

Informix Product Family
Informix Genero
Version 2.40

IBM Informix Genero Studio User Guide



Informix Product Family
Informix Genero
Version 2.40

IBM Informix Genero Studio User Guide



Note

Before using this information and the product it supports, read the information in "Notices" on page B-1.

Edition Notice

This edition applies to version 2.40 of IBM Informix Genero (product number 5725-D13) and to all subsequent releases and modifications until otherwise indicated in new editions.

Licensed Materials - Property of IBM.

All rights reserved. All information, content, design, and code used in this documentation may not be reproduced or distributed by any printed, electronic, or other means without prior written consent of Four Js Development Tools.

© Four Js Development Tools 1995, 2012

Contents

Introduction	xi
About this publication	xi
What's new in Informix Genero Studio, v 2.40	xi
Contacting IBM Software Support	xii
 Chapter 1. Setup and Configuration	 1-1
Setting Up Genero Studio	1-1
General	1-1
Local Studio environment	1-3
Remote Studio environment	1-4
Configuration Checklist	1-7
Genero Studio Configuration Dialogs	1-8
Genero Configuration Management Dialog	1-8
Changing the Active Configuration	1-10
GDC Configuration Management	1-11
Genero Hosts Management	1-12
Local Host Configuration	1-12
Remote Host Configuration	1-13
Genero Installations	1-13
GAS Configurations	1-13
Environment Sets	1-14
Default Environment Sets	1-15
Environment Set Variables	1-15
Configuring a Remote Genero Studio Server	1-16
Prerequisites	1-16
Define the Remote Server	1-16
Defining a Remote Configuration	1-18
Launching Genero Studio Server (TCP connections only)	1-20
Studio Text Encoding Settings	1-20
Text Encoding	1-20
The LANG environment variable	1-21
Genero Studio 2.x Migration Notes	1-21
Version 2.40.	1-21
Version 2.30.	1-22
Version 2.20.	1-22
Advanced SSH Configuration.	1-23
Configuring Public Key Authentication on Windows	1-24
Configuring public key authentication on Linux	1-26
Advanced Character Sets Management in Genero Studio	1-28
Text Encoding Plugins	1-28
Character Mapping Table (<i>GST-install-dir/conf/encodingMap.xml</i>)	1-28
Adding Plugins	1-29
Example.	1-29
Launching Web applications from Studio	1-31
Studio and XCF files.	1-31
XCF Custom Code Examples	1-31
Create and apply a custom XCF for your Web application	1-32
How Studio selects the custom XCF to use	1-33
Editing and validating a custom XCF file	1-33
Legal Notices	1-33
 Chapter 2. Getting Started	 2-1
Genero Studio New Features	2-1
Version 2.40	2-1
Version 2.31	2-3

Version 2.30	2-3
Beginning Work	2-4
Getting Started	2-4
Launching Studio Modules	2-5
Getting Help.	2-6
Sample Programs and Files	2-6
Genero Studio Modules	2-6
Overview	2-7
Application Modeling and Generation	2-7
Code Editor	2-7
Meta-schema Manager	2-7
Graphical Debugger	2-8
Form Designer	2-8
Project Manager	2-8
Report Writer	2-8
Source Code Management - SVN	2-8
File Browser	2-9
Adding Web Services	2-9
Graphical Differential.	2-9
Configuring Genero Studio	2-9
Preferences: Customizing Studio Modules	2-9
Code Analyzer.	2-10
Using Genero Studio	2-10
Main Window	2-10
Menus and Toolbars	2-12
File>>New	2-13
File>>Save/Save As/Save All.	2-15
Getting Help	2-16
Genero Studio Help window	2-16
Browser View	2-18
Genero Studio Error Messages	2-18
Form Designer Error messages	2-19
XML Validation Error messages	2-29
Metaschema Manager Error Messages	2-31
Business Application Modeling Error Messages.	2-37
SVN Error Messages.	2-41
Business Records Error Messages	2-41
Working With Existing Genero Applications	2-45
Using Studio in a Remote Environment	2-45
Editing the Project Files.	2-46
Using the Form Designer	2-46
Using the Graphical Debugger	2-46
Importing Existing Projects into Project Manager	2-47
Integrating your Tools into Studio	2-47
Chapter 3. Application Modeling	3-1
Introduction to Genero Studio Business Application Modeling	3-1
A Model-Driven Approach	3-1
BAM Diagrams	3-2
Application Generation TCL templates	3-6
BAM Tools	3-7
How Code Generation Works	3-7
Application Generation Requirements	3-8
Sample databases and files	3-8
Deploying the Generated Application	3-8
Your First Application (the dbApp template)	3-9
Step 1: Create a Managed Project	3-9
Step 2: Add the Database Schema to your Project	3-11
Step 3: Create the Business Application Diagram (.4ba)	3-11
Step 4: Create the Program and Form Definitions	3-12
Step 5: Define Properties for the Program.	3-14

Step 6: Define Properties for the Form	3-15
Step 7: Build and Execute the New Program.	3-17
Step 8: Add a Detail Table to the Form	3-18
Step 9: Define Properties for the Form	3-19
Step 10: Set the Relationship Between the Tables	3-20
Business Application Model Diagram	3-22
The Diagram	3-22
The Program Component and Menu	3-24
The Form Component (.4fdm) and Menu	3-24
Zoom and ReportData Objects	3-25
The Relation Component	3-26
Additional Views	3-26
Import Files into the Diagram (.4ba) from the Project	3-27
Creating CRUD Forms	3-28
Database Schema	3-29
Implement Form from Database	3-29
Form components	3-31
Records Tab	3-31
Genero Report Writer reports	3-32
Modifying the Business Application Diagram	3-33
Creating the Genero Report Designer report definition (.4rp)	3-35
Rebuilding the application.	3-35
Lookup fields and ComboBoxes	3-35
Using Lookup Fields.	3-35
Populate a ComboBox Dynamically.	3-39
Program Flow: Opening Zooms and Additional Forms	3-43
Opening a Zoom form	3-44
Opening A Second Form	3-47
Actions and Action Views	3-50
Default Actions	3-50
Default Action Views	3-51
Adding to the TopMenu	3-52
Adding to the Toolbar	3-53
Adding Buttons	3-54
Modifying Action Defaults (dbapp.4ad)	3-54
The Application Source Code and Files	3-55
Application Files	3-56
The Program .4gl File	3-57
The Module .4gl Files	3-58
Report .4gl Files	3-58
Zoom Module .4gl Files	3-58
The Library file (libdbappCommon.4gl)	3-59
Adding Code	3-59
Modifying the Application.	3-59
POINTS and BLOCKS	3-60
Adding your Own Functions	3-60
Adding Your Own Actions.	3-61
Adding Additional Variables	3-61
Customization	3-62
Creating a Custom Template - Overview	3-62
Use Case: Create a New File Type	3-63
XSLT Template for Code Generation	3-63
Application Generation Templates	3-66
Default Application Generation Templates	3-66
Generation scripts (Templates)	3-67
Associated Files	3-68
Build Rules for Templates	3-75

Chapter 4. Project Manager 4-1

Genero Studio Project Manager	4-1
Using Project Manager	4-1

Menu of Actions	4-9
gsmake - Command line option to Build Projects	4-11
Importing a Project	4-13
Quick Guide to Creating and Testing a Project	4-14
Build Rules and Advanced Properties	4-15
Build Rules	4-16
Advanced Properties	4-19
Setting Environment Variables	4-20
Predefined Node Variables.	4-23

Chapter 5. Code Editor 5-1

Genero Studio Code Editor	5-1
Introduction	5-1
Edit a file	5-2
Fold Text	5-2
Square Selection	5-3
Split Windows	5-3
Encoding (i18n).	5-3
Bookmarks	5-3
BDL Syntax help	5-3
XML Editing.	5-3
Customizing Code Editor	5-3
Code Editor Main Window	5-4
Document Editing Window.	5-4
Code Structure View	5-5
Output View.	5-5
Search/Replace View	5-5
Coding Assistance	5-5
Using Auto-Completion (Ctrl space)	5-5
Using Code Templates (Ctrl+T)	5-5
Search and Replace	5-6
The Search/Replace View	5-7
The Search Results View.	5-8
Search: Wildcards and Regular Expressions	5-9
Print a File	5-12
Keymaps	5-12
Cursor movement keymap.	5-13
Selection keymap	5-13
Editing Text keymap.	5-14
Clipboard keymap	5-15
Search and Replace keymap	5-15
Buffers and Files keymap	5-15
Code Completion keymap	5-15
Code Templates keymap	5-16
KDE keystrokes	5-16
vi Commands List	5-17
Studio XML Catalog Files	5-19
Overview	5-19
The XML Catalog file	5-20
Add, Edit, Delete XML Catalog Entries	5-21

Chapter 6. Form Designer 6-1

Genero Studio Form Designer	6-1
Overview	6-1
The Form Designer	6-2
Formfields	6-3
Action views.	6-4
Design the Form	6-4
Form Design Window	6-5
Handling Errors	6-7

Align/Layout	6-7
Tab Index.	6-8
Form Structure View	6-8
Properties View.	6-9
Build (Compile) the Form	6-9
Command-line syntax: gsform	6-10
Using Styles	6-11
Using Localized Strings.	6-13
Studio Form Containers and Widgets	6-13
Containers	6-14
Widgets	6-18
Additional Form Elements.	6-24
Database Interface	6-27
Data Control Wizard.	6-27
Column Selection page	6-27
Container Selection Page	6-27
Query Creation Page	6-28
Dynamic Properties Page	6-29
Properties List.	6-30
Properties of the form	6-30
Properties of widgets and containers	6-30
Using Web Components	6-35
Adding a WebComponent Widget to a Form.	6-35
WebComponent Setup (advanced)	6-35
Chapter 7. Meta-schema Manager	7-1
Genero Studio Meta-schema Manager	7-1
Overview.	7-1
Creating the Meta-schema file (.4db).	7-2
New Empty Meta-schema	7-2
Adding the Meta-schema to your Application.	7-2
The DB Schemas tab	7-3
The Database Menu options	7-5
Generate SCH files and Database Creation Scripts	7-5
The Studio Meta-schema File (.4db)	7-8
Open the File	7-8
Indexes and Constraints	7-10
New tables and columns	7-14
Revert changes	7-15
Missing status	7-16
Filter View	7-16
Extracting the Meta-schema from a Database	7-16
Extracting the Information.	7-16
Database Server and User Information for each Database Type	7-18
Chapter 8. Graphical Debugger	8-1
Genero Studio Graphical Debugger	8-1
Local or remote Debug	8-1
Launch the Debugger.	8-1
Open/Edit the Source File	8-1
The Output View	8-2
Testing your Program.	8-2
Run the Program	8-2
Exit the Debugger	8-3
Breakpoints	8-3
Set a Breakpoint	8-3
Breakpoints: contextual menus.	8-4
The Breakpoints View.	8-4
Conditions	8-5
History View	8-5

Watch Points.	8-5
Setting a WatchPoint	8-6
Conditional Watchpoints.	8-6
The Watchpoints View	8-7
Data View	8-7
Backtrace Tab	8-9
Debugger Commands.	8-9
Command View	8-9
Record commands	8-10
Debugger Command List	8-10
Supported Commands	8-10
Unsupported Commands	8-22

Chapter 9. Genero Report Writer 9-1

Getting Started	9-1
Introduction to Reports	9-1
Run Reports - the GRW Demo.	9-3
Steps to a Report	9-6
Configuring Fonts and Printers	9-9
BDL Programs	9-11
Writing the Genero BDL Report Program	9-11
Configuring for Labels	9-26
Report Data Wizard	9-27
Modifying the Output	9-33
Change Report Output Options	9-34
Using Localized Strings.	9-41
Report Design Document	9-43
Overview of Genero Report Designer	9-43
Designing a Report	9-45
Design How-To	9-62
Working with Business Graphs	9-74
Expressions in Properties	9-84
Reference	9-91
Report Design Elements (The Toolbox).	9-91
Report Element Properties	9-102
Reporting API Functions	9-119
Bar Codes	9-155
RTL Classes Overview.	9-185
Dimension Resolver	9-186
RTL Class Reference	9-190
The Boolean Class	9-190
The Color Class	9-191
The Date Class	9-195
The Enum Classes	9-196
The FGLNumericVariable Class.	9-202
The FGLStringVariable Class	9-203
The Numeric Class	9-204
The Runtime Class	9-213
The String Class.	9-215

Chapter 10. Code Analyzer 10-1

Genero Studio Code Analyzer	10-1
Sequence Diagrams	10-1
Overview	10-1
Creating the Diagrams	10-3
Dependency Diagrams	10-3
Displaying the Dependency Diagram	10-3
The Structure Panel	10-4
The Filter View	10-6

Chapter 11. Customization	11-1
Customizing Genero Studio: Preferences	11-1
Studio Preferences Menu	11-1
Saving changes to preferences	11-2
Import Preferences	11-2
Genero Studio Preferences	11-3
General preferences	11-3
Workspace configuration	11-4
User action configuration	11-4
File Associations configuration	11-5
Messages configuration	11-6
History configuration	11-6
User Interface Preferences	11-7
Toolbars configuration	11-7
Menubars configuration	11-7
Accelerators configuration	11-7
Compiler and Runtime Preferences	11-9
Preprocessor configuration	11-9
Compilation configuration	11-9
Run/debug configuration	11-9
Build Rule configuration	11-9
Form Designer Preferences	11-10
Form settings	11-10
Form Elements settings	11-11
Database Form settings	11-11
Code Editor Preferences	11-12
General Preferences	11-12
Behavior and Display Preferences	11-12
Color and Font Preferences	11-13
Template Preferences	11-14
XML Schema/DTD Preferences	11-15
Additional Preferences	11-17
Source Code Management Configuration	11-17
File Browser	11-18
Sequence Diagram	11-18
Application Generator Preferences	11-18
Graphical Differential Configuration	11-19
Meta-schema Manager Preferences	11-20
Welcome Page Preferences	11-20
Report Writer Preferences	11-21
 Chapter 12. Source Code Management - SVN	 12-1
What is Genero Source Code Management?	12-1
Specify a Subversion client	12-2
Getting Started	12-3
Checking out an existing project or project files	12-3
Checking out an existing project from the File Browser view	12-3
Checking out an existing project from the SCM Menu	12-3
Using the SVN Checkout dialog	12-4
The Process Log	12-5
SVN Log Select revision dialog	12-5
Refining the list of revisions displayed	12-7
Accessing Source Code Management	12-7
SVN Log	12-7
SVN Repository Browser	12-9
SVN Status	12-10
Working with SVN	12-11
Adding Files	12-11
Deleting Files	12-11
Update/Update All	12-11
Commit/Review Changes	12-12

Revert changes / Unadd files	12-13
Diff with Current Revision	12-13
Cleanup	12-13
Copying SVN Files and Directories	12-13
Copying directories and files from the File Browser	12-13
Copying directories and files from the SCM Menu	12-13
The SVN Copy dialog	12-13
Merge/Revert	12-14
Revert from a single revision	12-16
Merge and revert from a range of revisions.	12-16
Switch (Moving a working copy)	12-16
Create Patch	12-17
Apply Patch	12-18
Manage Properties	12-18
Chapter 13. Other Studio Modules	13-1
Using Web Services with Studio	13-1
Creating a Web Services program	13-1
Add Web Service	13-2
The Web Services Wizard	13-2
Building the Application	13-6
Update the WSDL	13-6
Genero Studio Graphical Differential	13-6
Graphical Differential tabbed page	13-6
Using an External tool	13-9
Genero Studio File Browser	13-9
File Browser	13-9
Selecting Files	13-11
Managing Files	13-11
Appendix. Accessibility	A-1
Accessibility features for IBM Informix products	A-1
Accessibility features.	A-1
Keyboard navigation.	A-1
Related accessibility information	A-1
IBM and accessibility.	A-1
Notices	B-1
Trademarks	B-3

Introduction

This publication contains information about IBM® Informix® Genero® Studio.

About this publication

This publication describes how to configure and use IBM Informix Genero Studio.

Genero Studio (GST) enables you to quickly create cross platform business applications. Model your application in an intuitive graphical schema and design user interfaces that work across multiple platforms: be they desktop, browser or mobile based.

Studio comprises an application generator that models applications from the database schema and generates hardware independent code; a code sensitive editor and analyzer for super efficient program structure; a graphical debugger; a forms designer for desktops, browsers and mobile devices; a web services generator to integrate with Service Oriented Architectures; a source code control system for team development; a project manager and a graphical report generator.

What's new in Informix Genero Studio, v 2.40

This publication includes information about new features and changes in existing functionality.

The following changes and enhancements are relevant to this publication.

Table 1. What's new in Informix Genero Studio, Version 2.40

Overview	Reference
Update settings in Source Code Manager have been updated. For example, you can now omit externals.	See "Update/Update All" on page 12-11.
You can now add multiple files.	See "Adding Files" on page 12-11.
The Meta-Schema Manager has been updated for easier editing and viewing. Options have been added in Edit mode, and document errors and warnings are displayed while editing.	See Chapter 7, "Meta-schema Manager," on page 7-1.
Multiple views of the same database can be created.	See "The schema (.4db)" on page 7-8.
With Business Application Modeling, new business records have been added for reports.	See "Rebuilding the application" on page 3-35.
With Business Application Modeling, an improved editor assists with search and document errors.	See "Introduction to Genero Studio Business Application Modeling" on page 3-1.
With Business Application Modeling, new templates for a new architecture and for the generation or report code are provided.	See "List of TCL Templates" on page 3-67.
With Business Application Modeling, advanced customization is now possible.	See "Customization" on page 3-62.

Table 1. What's new in Informix Genero Studio, Version 2.40 (continued)

Overview	Reference
With Business Application Modeling, an integrated code analyzer is provided.	See "Genero Studio Code Analyzer" on page 10-1.
With Report Writer, data matrix barcodes are supported.	See "data-matrix" on page 9-185.
With Report Writer, new properties for Text and Image fields allow them to behave as hyperlinks.	See "Using Hyperlinks in a Report" on page 9-72.
With Report Writer, additional options have been added for MapCharts.	See "Map Chart" on page 9-74.
With Report Writer, new report output format for Excel.	See "Sending Report Data to an Excel Spreadsheet" on page 9-38.
With Report Writer, new report output format for Word (.rtf).	See "Outputting Report Data in Microsoft RTF Format" on page 9-40.
With Report Writer, there is a new function to create a process-level data file and a document at the same time	See "fgl_report_setProcessLevelDataFile" on page 9-134.
With Report Writer, there are new functions to create document metadata for compatibility reports	See "Report Design Document Metadata" on page 9-58.
With Report Writer, there are new functions to provide generic report auto-formatting when no report design document (.4rp) is specified for a report	See "Auto-formatting Reports that have no .4rp (Report Design Document)" on page 9-23.
With Report Writer, there are additional new report API functions, indicated by a "New in 2.40" designation.	See "Reporting API Functions" on page 9-119.
Additional support for application rendering with the Genero Web Client.	See "Genero Configuration Management Dialog" on page 1-8.
You can now import configuration setup and preferences of earlier versions of Genero Studio when you first start up a new installation of Genero Studio.	See "General preferences" on page 11-3.
For ease of use, the Windows menu allows you to select the default layout, and you can toggle Full Size Documents or Full Screen to maximize your working space	See "Windows Menu" on page 2-13.
States of previous sessions are remembered.	See "Workspace configuration" on page 11-4.
Support of high contrast mode and of Windows screen readers for accessibility standards.	See Chapter 1, "Setup and Configuration," on page 1-1
Multiple instances of Genero Studio can be opened simultaneously.	See Chapter 1, "Setup and Configuration," on page 1-1
Various performance improvements have been implemented.	See "Version 2.40" on page 2-1

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM Software Support contract, and you must be authorized to submit problems to IBM. For information about the types of available support, see the Support portfolio topic in the *Software Support Handbook*.

Complete the following steps to contact IBM Software Support with a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For more information, see the Getting IBM support topic in the *Software Support Handbook*.
2. Submit the problem to IBM Software Support in one of the following ways:
 - Online through the IBM Support Portal: You can open, update, and view all your Service Requests from the Service Request portlet on the Service Request page.
 - By phone: For the phone number to call in your region, see the Directory of worldwide contacts web page.

Chapter 1. Setup and Configuration

Topics:

- Setting up Studio
- Configuring Studio
- Configuring remote Studio Server
- Text encoding settings
- Migration notes
- Advanced SSH Configuration
- Advanced character sets management
- Launching Web applications from Studio
- Legal Notices

Setting Up Genero Studio

Topics:

- General
 - Installation choices
 - Default Studio Configuration
 - Sharing Projects/Source Code Management
 - Database Access
 - Text Encoding
- Local Studio environment
 - Software requirements
- Remote Studio environment
 - Software requirements
 - Using Shared drives
 - Genero Studio Server
 - Connecting to Genero Studio Server
- Configuration Checklist (Changing the Studio SDK)

See also: Configuration Management Dialogs, The Remote Studio Server, Text Encoding and Character Sets

General

Installation choices

When you install the Studio software package, you can automatically install the following Genero components:

- Genero Business Development Language
- Genero Desktop Client - to be used for display
- Genero Web Client - to be used for display
- Genero Web Services
- Genero Application Server

Genero Studio can be installed for:

- a single developer working on his local machine (installation on a local host)
- a development team (installation on a remote server accessed by developers from their own client machines)

Default Studio Configuration

Genero Studio configurations provide the information that Studio needs in order to run on your local or remote hosts. A default configuration is provided, based on the software in the distribution package being installed in the default directories. See Configuration Checklist.

Sharing Projects/Source Code Management

You can share your Genero projects using a Version Control System (VCS):

- A network drive/directory must be set up on the remote server for the storage of the project and source files.
- All paths in the **.4pw** files must be relative paths; only the path of the project(s) can be absolute.
- The network drive/directory must be mapped on each client machine.
- You will check out a copy of the project file and source files from the VCS repository to your mapped network drive. After making changes, commit your files back to the VCS repository.

The Source Code Management (SCM) module included with Genero Studio requires that a **Subversion client of version 1.5.2 or later** must be installed on your local machine. Genero Studio for Windows is shipped with Collabnet's Subversion client. On GNU/Linux, you must install Subversion on your system. To specify the location of your SVN client, select **Tools>>Preferences>>Source Code Management>>Subversion** from the Studio main menu.

For more information about Collabnet's Subversion, see: <http://subversion.tigris.org/>

Database Access

To execute an application that accesses a relational database, Genero Studio uses the Genero runtime system and the specific database client software, which must be properly configured. Entries in a FGLPROFILE configuration file can be used to connect to the database. See the specific page for the supported database vendors' software in the Open Database Interface section of the Genero BDL documentation.

Genero Studio database schema files (.4db) allow you to use the database tables and columns that are listed in the schema to define variables and form fields in your application code and form definitions that are created in Studio.

In the Configuration Management dialog, default environment sets are provided for the various database vendors. Values for the environment variables, as well as additional variables, can be added in the Configuration Management Dialog as part of the Configuration process.

See Configuration Checklist .

Text Encoding

When writing a form or program source file, you use a specific character set. A default character set is specified in the Text Encoding field of Genero Studio General Preferences. This default **Text Encoding** method is the one that Studio

found on your system. If you want to use encodings for a different language, you can select a different Text Encoding choice from a list of all available encodings.

The current encoding is used during file operations of text documents, so the encoding must be correctly set before opening documents. Once loaded, Genero Studio uses Unicode for text management.

If you change the Text encoding, the LANG environment variable must be set accordingly in an environment set.

See Studio Text Encoding Settings for complete information.

Genero BDL and Genero clients have settings that must also be changed when you want to use encodings and character sets for a language that is different from that specified in the default text encoding for your system:

- See the *Genero Business Development Language User Guide*, section **Localization** for additional information about the settings for Genero Business Development Language.
- See the *Genero Desktop Client User Guide*, section **Localization** for additional information about settings for Genero clients.

Local Studio environment

The developers work individually, and all the Genero software is installed on their local machines. The database can be on their local machines, or they can access it from remote servers. Although each developer must have the Genero software installed on their local machine, some group development can be enabled through the use of an external Version Control System.



Figure 1-1. Single user

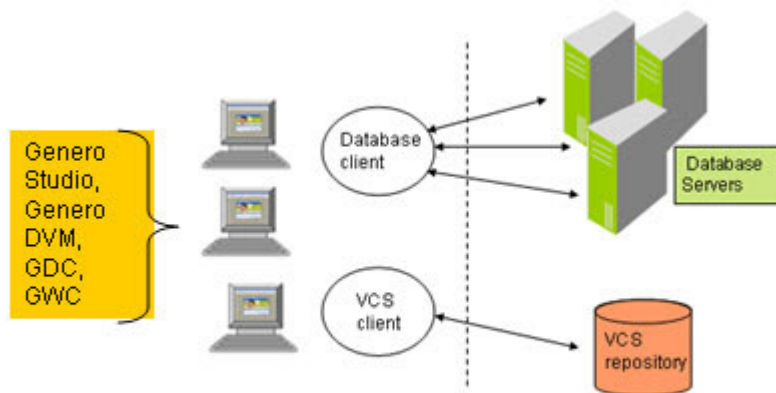


Figure 1-2. Multiple Users

Software requirements

- Genero Studio
- Installed Genero Business Development Language (BDL) compiler and runner (DVM)
- Installed Genero Desktop Client (GDC) and / or Installed Genero Application Server (provides the Genero Web Client - GWC)
- Installed relational database (Genero db, Informix, Oracle, etc.) and its accompanying database client (CSDK, Oracle client, etc.)

Each database software has its own client character set configuration. You must properly configure the database client locale in order to send/receive data to the database server, according to the locale used by your application. See *Localization: Database Client Settings* in the *Genero Business Development Language User Guide*.

As the project files (.4pw) in Studio Project Manager can not be shared, each developer must have his own copy of the Project definition. To share projects, use a VCS system.

See the installation documentation and release notes for additional information.

Remote Studio environment

Using a remote installation allows the developers to compile, debug, and execute Genero applications stored on a remote server. Genero Studio Server must be installed on the remote server.

We recommend that Studio clients connect to Genero Studio Server using SSH for better security and user management, since each user with an SSH connection works with their own server process. The relational databases that your application accesses can be stored on the same remote server as Genero Studio server, or on separate servers. A network drive must be set up for the remote storage of Genero source code files.

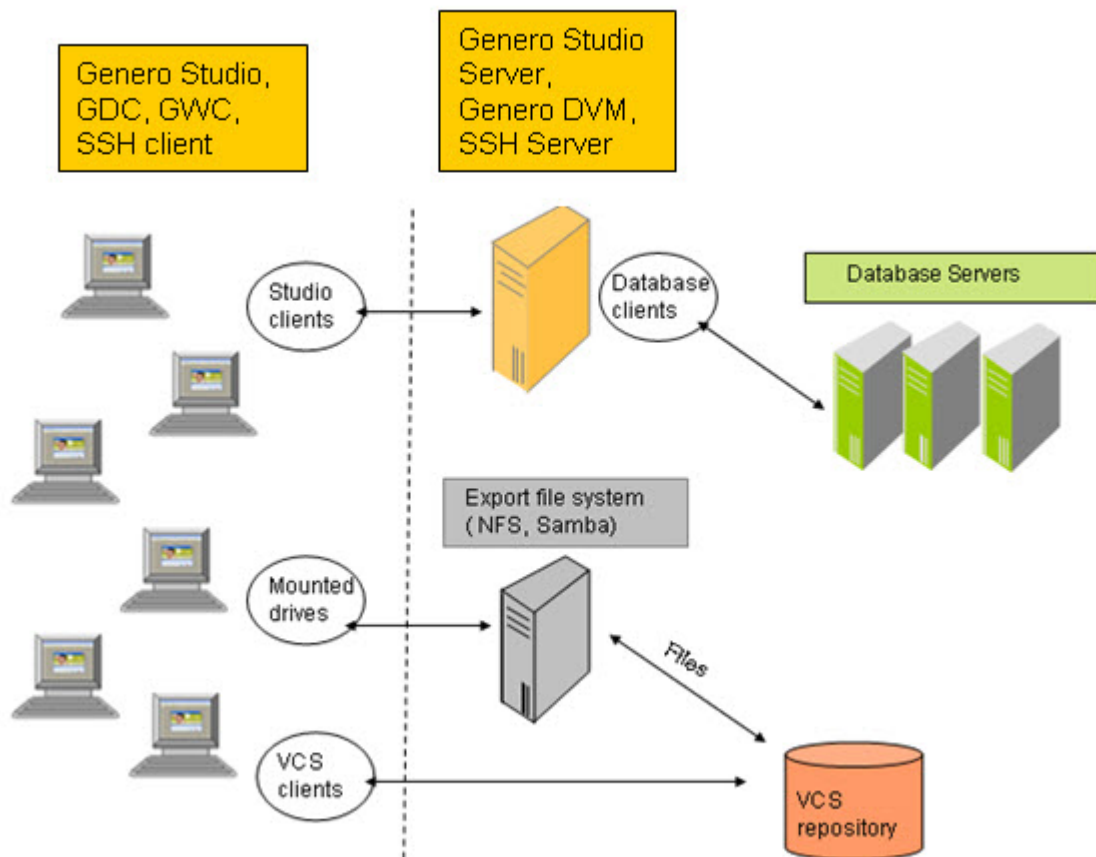


Figure 1-3. Remote Studio environment

Software requirements

On the server(s)

- Genero Studio Server (for Windows/Linux Genero Studio Server and Client are in the same software package)
- Installed Genero Business Development Language (BDL) compiler and runner (DVM)
- Installed relational database (Genero db, Informix, Oracle, etc.) and its corresponding client (CSDK, Oracle Client, etc.); **or** Database Client on this server and the corresponding Database on a different remote server.
- SSH server (if using the recommended SSH connections).

On the client

- Genero Studio
- Installed Genero Desktop Client or Installed Genero Application Server (provides the Genero Web Client - GWC)
- SSH client (part of Genero Studio), if using the recommended SSH connections.

Note: Each database software has its own client character set configuration. You must properly configure the database client locale in order to send/receive data to

the database server, according to the locale used by your application. See Localization: Database Client Settings in the *Genero Business Development Language User Guide*.

Using Shared drives

To work properly with the current version of Genero Studio, all program files accessed remotely must reside on a shared drive on a server accessible to your local network.

- You must share the drive using a standard tool, such as NFS (Network File System) for Linux networks. Samba, or a similar tool, can be used to allow Microsoft networks to share files stored on a Linux server.
- If Studio is installed on a Microsoft Windows client, you must map the shared network directory to a letter drive on the client, using the **Map Network Drive** option of Windows.
- You must specify the remote drive in the Genero Configuration Management dialog for remote servers.
- The character set will be specified in the environment of the **remote** FGL installation (set the **LANG** variable in the **remote** configuration variables). On the client machines, set text encoding in Studio Preferences, specifying the same character set.

Due to technical constraints in the current release, running the Genero Studio Server requires X11 to be installed on the system (but it can be run in TTY).

See the installation documentation and the installation notes in the release subdirectory of the Genero Studio software package for detailed Genero Studio installation information.

Genero Studio Server

Genero Studio remote environments must have Genero Studio Server installed on the remote server. See Configuring a Remote Studio Server and the installation notes in the release subdirectory of the Genero Studio software package for detailed instructions.

The Genero Studio client must add the remote Genero Studio Server to its list in order to access it. Genero Studio Server must be launched before you can configure Studio on your local machine to access it. If you are using an SSH connection, Genero Studio Server is launched automatically by SSH.

Genero Studio Server uses the current Genero environment and variables, so check your Genero environment before launching it.

Connecting to the Genero Studio Server

All remote configurations can be shared and are usable by each Genero Studio client that adds the remote Genero Studio Server to its list. Genero Studio can connect to the Genero Studio Server using either SSH or TCP. Using an SSH connection is recommended for better security and finer user access rights management.

- If you choose to connect via SSH, an SSH daemon must be running on the server host. There is no need to launch the Genero Studio Server on the host, **the SSH daemon will automatically launch it**.
- If you choose to connect via TCP, the Genero Studio Server must be launched before use.

See *Configuring a Remote Server* for additional information about Genero Studio Server.

Configuration Checklist

Configurations allow Studio to locate the installed Genero components necessary to create, compile, build, and execute Genero applications. A **default configuration** is provided, based on the software in the distribution package being installed in the default directories. If you have installed your own Genero software, or have not installed Genero Studio in the default directories, you will have to define a Genero Studio configuration.

Even if you use the default configuration, you must use Configuration Management to select the environment set appropriate for your Database Client software, and add any missing variables or values.

For instructions on defining a remote configuration, see *Configuring a Remote Genero Studio Server*.

Prerequisites

Prior to configuring Genero Studio you must:

- have access to an installed relational database that contains your business data
- have installed the database vendor's client software
- (for remote environments): have installed Genero Studio Server on your remote server
- If you wish to use a different character set than the default found on your system, use Preferences to change the Text Encoding. Then make the corresponding change to the LANG variable in the following Configuration Steps.

Configuration Steps

To change the default configuration or add a new one:

1. Select **Tools>>Configurations** from the Studio menu or the toolbar icon to define your Genero configuration or to make changes in existing configurations.



Figure 1-4. Configurations toolbar icon

A Genero configuration contains settings for:

- An installed Genero front end client - information about the location of the Genero Desktop Client or Genero Web Client software
- The local or remote host - information about the host machine.
- Genero installations - information about the location of the compiler and runtime version of Genero that will be used by Studio (FGLDIR)
- Environment variable sets for the database client setup - values for environment variables required by your database, or used by Studio or Genero BDL to connect to the database. If you have changed the Text Encoding in Preferences, make corresponding changes to the LANG variable in an environment set.
- Mounted drives information (remote server only) - information about your shared drives

Genero Configuration Management dialogs guide you through the configuration for each of these settings.

2. Select **Database>>New Schema** from the Studio menu or the toolbar icon to create a Database Schema for the installed databases to which you have access. See Database Access.



Figure 1-5. New Schema toolbar icon

3. Review requirements for Sharing Projects/Source Code Management, Text Encoding, and Connecting to a Studio Server, as applicable.

To add or make changes only to the environment sets:

1. Select **Tools>>Configurations** from the Studio menu or the toolbar icon.
2. Select the desired **Configuration** in the Configuration name list.
3. Click the **Configure button** at the bottom of the DisplayRuntime dialog.
4. Select the Environment set you wish to change from the tree displayed in the host panel;

- or -

Select the **Add an Environment Set** icon at the top of the host panel to add a new Environment set.

Genero Studio Configuration Dialogs

Topics:

- Genero Configuration Management Dialog
- GDC Configuration Management
- Genero Hosts Management
 - Local Host Configuration
 - Remote Host Configuration
- GAS/GWC Configuration Management
- Genero Installations
- Environment Sets
 - Default Environment Sets
 - Environment Set variables

See also: Setting Up Genero Studio, Configuring a Remote Studio Server

Genero Configuration Management Dialog

Genero Studio configurations provide the information that Studio needs in order to run on your local or remote hosts. A default configuration is provided, based on the software being installed in the default directories.

Note: For instructions on defining a remote configuration, see Configuring a Remote Genero Studio Server.

You can have multiple Genero configurations, distinguished by the name that you have assigned:

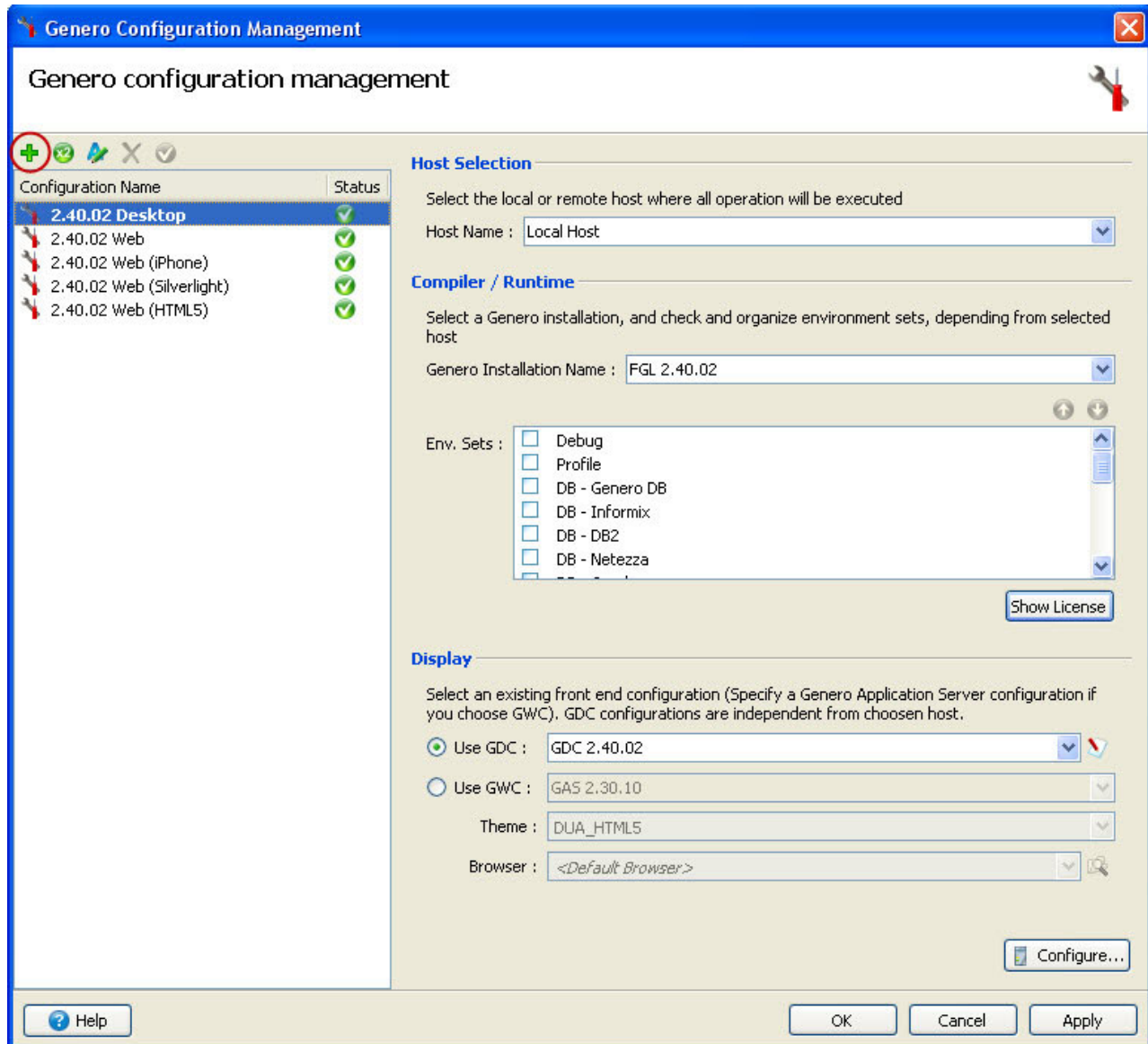


Figure 1-6. Genero Configuration management dialog

- **Configuration name** - The names of the available Genero configurations are displayed. Select the configuration name to view its settings.

Use the **toolbar icon** at the top of the table to **add** the name of a new configuration. Once added, you can enter its settings. The toolbar also has icons to **duplicate**, **rename**, and **delete** Genero, GDC and GAS configurations, Genero installations and environment sets.

A checkmark in the **Status** column indicates there are no missing settings.

- **Host Selection** - The **Host Name** field identifies the host where the application will be compiled and executed. The host could be local or remote. To view/edit the settings for the selected host, or to add a new remote host, select the **Configure** button (See Genero Hosts management).
- **Compiler/Runtime**

Genero Installation name: This field identifies which Genero Installation will be used to compile or run the application. A Genero Installation specifies the FGLDIR and the runner name. The names of the Genero installations that are available for the selected host are displayed in the dropdown box. Select the desired installation for your configuration. To view/edit the settings for the selected Genero Installation name, or to **add** a new Genero Installation, select the **Configure** button (See Genero Installations).

Env. Sets: An environment set is a collection of environment variables to be set prior to compiling or running the application on the specified host. The name of the environment sets available for the selected host are listed. Use the up/down arrows to set the priority of environment sets. A **checkmark** indicates the environment set is active. To view/edit the settings for the selected Environment set, or to **add** a new Environment set, select the **Configure** button. (See Environment Sets for complete information). Default environment sets have been created for the RDBMS supported by Genero. Default variables have been entered for each set, and values have been provided where possible. Select your environment set, and enter any missing values or variables.

- **Display**

Front Ends are used by Genero Studio to display forms and applications. Either a Genero Desktop Client (GDC) or a Genero Web Client (GWC) can be selected as the Front End. Select the radio button to specify the type of Front End.

GWC configurations can be viewed and edited by selecting the **Configure** button.

GDC configurations can be viewed and edited by selecting the edit icon to the right of the GDC list box.

When you select Use GWC, you can choose an available **Theme** to use when running an application. You can also override the web **browser** preferences setting and choose the web browser to use when running an application. If the theme field is left empty, then default theme will be used. If the web browser field is left empty, then the web browser defined in preferences will be used.

Confirm or cancel the changes:

- **OK** - Save and apply all modifications, then exit.
- **Cancel** - Undo all modifications and exit. The last saved values are restored.
- **Apply** - Confirms your updates, allowing you to test the new configuration.

Changing the Active Configuration

When you have multiple configurations defined, only one of the configurations will be active. The name of the active configuration will be displayed in bold in the Configuration table in bold. Select a different configuration name and click the **Set Active** icon at the top of the table, or select **Set Active** from the right-click menu.

You can also change the active configuration from the Genero Studio main window. The **combobox** at the bottom right corner of the main window displays the name of the currently active configuration. Click the down arrow to select a different configuration from the list of all available configurations. If you select the Configuration icon located to the right of the combobox, the Genero Configuration Management dialog displays.

GDC Configuration Management

GDC Configurations contain information about the available Genero Desktop Clients. You can create multiple GDC Configurations, each with a different name. See the GDC documentation for information about installing the GDC.

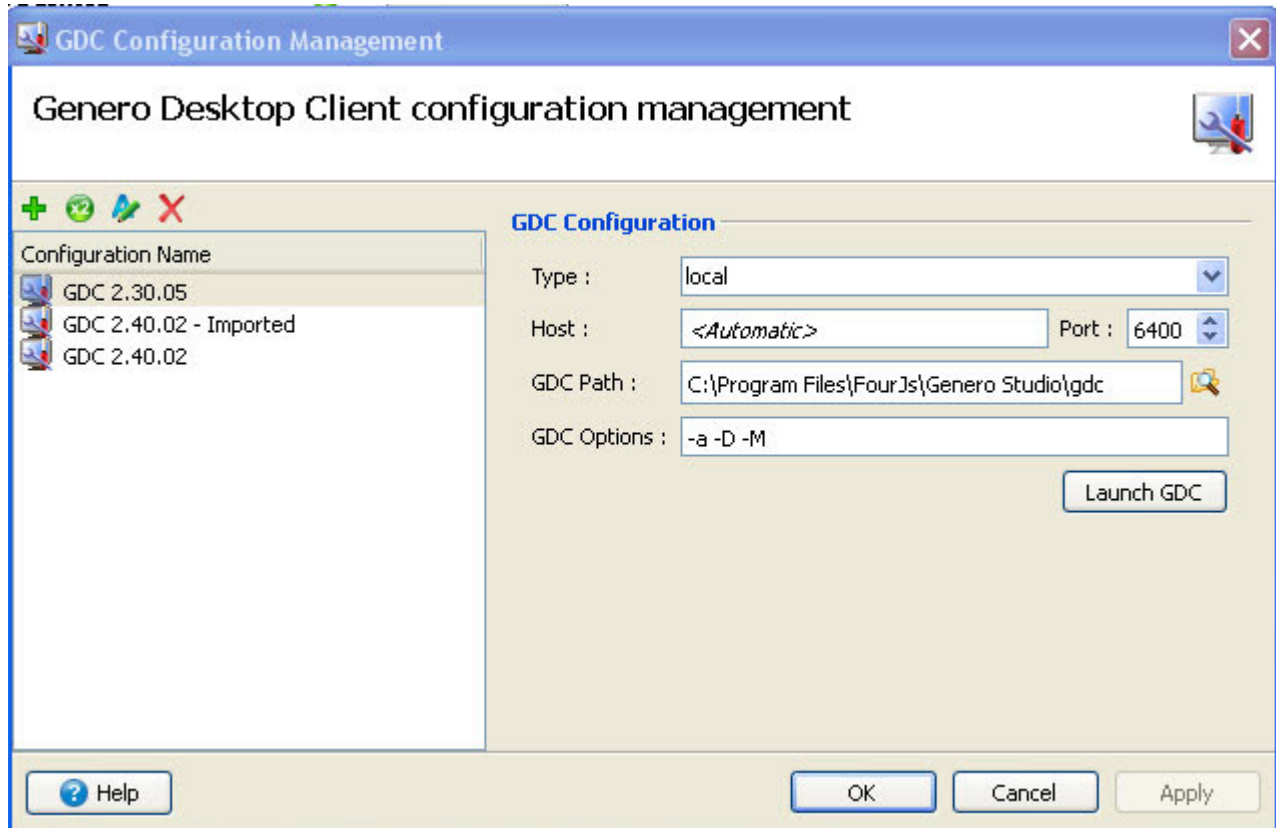


Figure 1-7. Genero Desktop Client configuration management dialog

Configuration name list - Select the GDC Configuration name from the list to modify its settings. Use the **toolbar icons** at the top of the list to **add** the name of a new Configuration or **remove** an existing one. Once a name is added you can fill in its settings.

Configuration settings

- **Type:** remote or local
- **Host:** hostname or IP address where the GDC client executes
- **Port:** Port from which GDC client is to be launched
- **GDC path:** path where GDC client is installed
- **GDC Options:** command line options for the execution of GDC

Click the **Launch GDC** button to test your settings.

Note: In local host mode, when the host field is empty, the client IP address will be automatically detected in order that GDC will always be accessible from fgldr. You can change this manually.

Confirm or cancel the changes.

Genero Hosts Management

Hosts Management contains information about the local or remote host(s) on which Genero Studio Server is running. The Host list contains the names of the hosts that have been defined. You can have multiple remote hosts, distinguished by the name that you have assigned. A green check icon in the **Status** column indicates there are no missing settings.

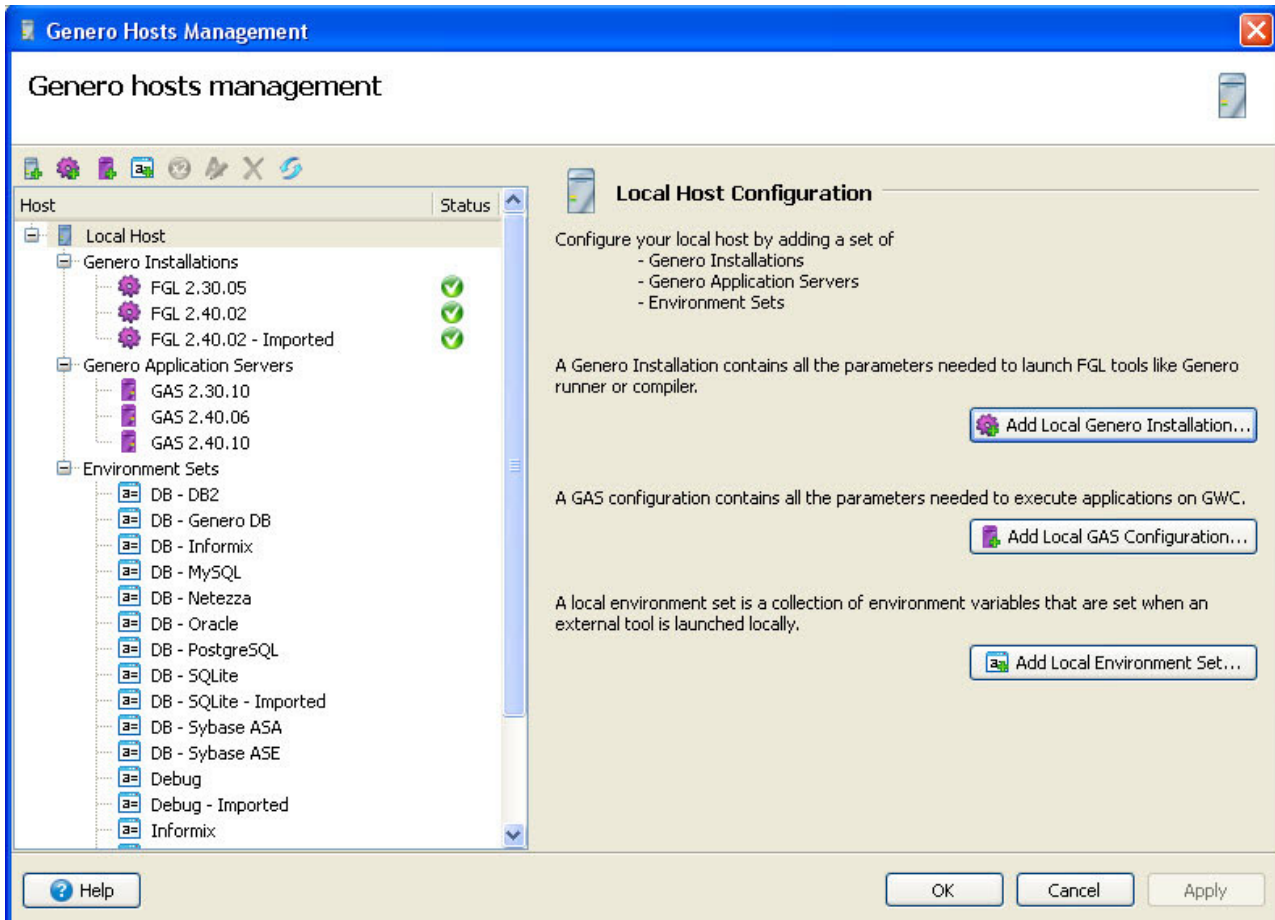


Figure 1-8. Genero Hosts Management dialog

Local Host Configuration

You configure your local host by adding Genero Installations for the compiler and runner settings, GAS Configurations for the parameters needed to run applications in the GWC and Environment Sets to organize environment variables needed when the application is launched.

- **Select** an item in the Host list to display its information.
- **Add** a new item in the Host list by using one of the buttons or the toolbar icons above the **Host** list.
- Complete the details for the added item:
 - Genero Installations
 - GAS Configurations
 - Environment Sets

Once you've completed your host configuration, confirm or cancel your changes.

Remote Host Configuration

If you are using Genero in client/server mode, you must use a remote Genero Studio Server to work with your remote Genero configurations from your Microsoft Windows / GNU Linux clients. The remote Genero Studio Server provides configuration information to your client machines, and allows them to send requests (compile, run, debug) back to the remote Genero Studio Server, which executes the requests.

Each Genero Studio client that wishes to use the remote Genero Studio Server must add the server to its list in Configurations. All remote configurations can be shared and are usable by each Genero Studio client that adds the remote Genero Studio Server to its list. Remote host configurations can be renamed, but not duplicated.

See Configuring a Remote Genero Studio Server for complete information.

Genero Installations

A Genero Installation contains information about the compiler and runtime version of Genero that will be used by Studio. You can have multiple installations configured for each defined host, distinguished by the name that you have assigned. Based on the Studio package that you installed, a default local Genero configuration is created.

Select the name of the Genero Installation in the Host list to view or edit the **Genero Installation settings**:

- **FGLDIR**: the installation directory of the Genero BDL software
- **FGL runner**: the name of the Genero executable
- **FGL options**: options given to the FGL runner when an application is launched.
Select the **Check Installation** button to test the settings.
- **Version**: The version information of the Genero installation is displayed.

Confirm or cancel your changes.

GAS Configurations

Genero Application Server (GAS) Configurations contain information about the available Genero Web Clients (GWC). You can create multiple GAS Configurations, each with a different name. If you did not install the entire Studio package initially, you must install and set up the Genero Application Server (GAS), which includes the GWC rendering engine. See the GAS documentation for details on installation.

When you select a GAS configuration, Studio will dynamically generate an external application to enable you to run, debug, or preview an application or file.

You can specify the browser to use with GWC in Genero Studio Preferences (**Tools>>Preferences**) or in the Configuration Management dialog.

Select the name of the Genero Application Server in the Host list to view or edit the **GAS Configuration settings**:

- **Host**: Specify the GAS IP address, or leave empty (automatic) to use your server address.
- **Port**: Port on which the Genero Application Server is running.

- **Connector:** Indicate the connector to be used if the GAS is running behind a Web server, or leave this setting blank if you are using a stand-alone GAS for development purposes (direct connection).
- **Fetch:** Click this button to provide lists of **Application Groups** and **Parent Applications**. This feature uses GAS monitor mode. By default, monitor mode is only accessible from localhost, so fetching works only if **Host** field is left empty.
- **Application Group:** The name of the Application Group defined in the GAS as.xcf file. Enter the value "_default" for the group to use the default configuration file, unless you have an explicit group to specify.
- **Group Directory:** The directory of the Application Group, the same path as the one referred to in the as.xcf file. If using the default application, the corresponding directory (usually "<GAS installation directory>/app") has to be entered in the "Group Directory" field.
- **Parent Application(optional):** The identifier of the parent GWC application; if no identifier is given, the default GWC application will be used.

Confirm or Cancel your changes.

Tip:

An application defining custom snippets should be used as the parent application in the GWC configuration. This way, custom snippets will be integrated in the application you wish to run.

Example configuration:

- **Host:** <empty> **Port:** 6394
- **Application Group:** mygroup
- **Group Directory:** c:\work
- **Parent application:** defaultgwc

In this configuration, GAS applications will be run using the GAS located on the current computer at port "6394". The group "mygroup" in the as.xcf file points to the directory "c:\work"; the corresponding fields in the GWC configuration dialog are filled with the same information.

Note:

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

See the GAS documentation for information on setting up GWC and GAS and adding applications.

Environment Sets

You may define sets of variables to be added to, or to overwrite, the default environment. These environment settings are also usable by each Genero Studio client that adds a remote Genero Studio Server to its list.

When you launch any Studio subprocess (run, GDC, compilation, debug, and so on) Genero Studio sets all redefined environment variables according to the current context (the context from Projects may differ from the File Browser context).

After pressing the **Configure** button on the Genero Configuration Management dialog, the Genero Hosts Management page lists the default environment sets. Default variables and some values are provided for the RDBMS supported by Genero.

Default Environment Sets

- **Various RDBMS sets** - contains some variables required by database clients from the various vendors. The values for the variables must be entered before using these environment sets. See Database Client Environment for information about the specific variables for each RDBMS.
- **Debug** - contains variables that set the debug level (FGLSQLDEBUG for FGL/SQL, FGLWSDEBUG for Web Services, and GREDEBUG for Report Engine.) The values are already set to 9, the highest debug level, but can be changed to any number between 1 and 9. Debug files will be generated in the current directory.
- **Profile** - contains the environment variable FGLPROFILE; for the value, enter the path of the current configuration (fglprofile) file to be used by the system before enabling this set on the Genero Configuration Management page; the default location is *<BDL-install-dir>/etc/fglprofile*. If you are using a remote Studio Server, the value should be set to the location on the remote host.
- **Report Writer 2.40** - contains the environment variables set by default to the installation directory of Genero Report Engine. This set is automatically enabled, as the environment variables are required when working with reports.

Select the name of an environment set in the tree list to view/modify the **Environment Set variables** table. If there are no environment sets listed, use the **Add Local/Remote environment set** button in the toolbar at the top of the tree list, or select **Environment Set** in the tree list and right-click for a menu option to **add** a new set of environment variables.

Environment Set Variables

- **Type**: valid choices are **value**/**directory**/**directory list**/**file**
- **Name**: the name of the variable
- **Value**: value of the variable

Use the buttons at the top of the Variables table to **add** a new variable to an existing set, and to **edit** or **delete** a selected variable. When entering the value, if the type is **directory**, **directory list**, or **file**, select the ... button to browse for the correct value. See Setting Environment Variables for a complete discussion of environment variables and how they work.

Confirm or cancel your changes.

Important: If you have set Text Encoding in **Genero Studio preferences** to change the default charset, you must set the corresponding LANG environment variable in your current environment set. Invalid settings of **LANG** may cause compilation errors. See Studio Text Encoding Settings.

Configuring a Remote Genero Studio Server

Topics:

- Prerequisites
- Defining the Remote Server
 1. Define the Remote Server
 2. Define the Genero Installation
 3. Add Genero Application Servers
 4. Add Environment Sets
 5. Define Mount Points
- Defining a Remote Configuration
 1. Add a new Configuration
 2. Define the Configuration
- Launching Genero Studio Server (TCP connections only)

See Also: Setting Up Genero Studio

Prerequisites

Before beginning to configure your remote server, make sure:

- Genero Studio Server is installed on the remote server, and the Genero Studio Client is installed on your PC; See Remote Installation. The versions of Genero Studio running locally and running on remote servers must be the same.
- Any BDL application files that you will access remotely reside on a shared drive, on a server accessible to your local network. See Using Shared Drives for additional information.

Note: Genero Studio Server opens a port on the server; make sure that adequate security has been set.

Define the Remote Server

For a Genero Studio client to connect to a remote Genero Studio Server, it must add the remote server to the **Hosts** defined in Studio Configurations, and must have a remote Configuration defined for that server.

All remote configurations can be shared and are usable by each Genero Studio client that adds the remote Genero Studio Server to its list.

1. Add the Remote Server.

First, decide on your connection type. Genero Studio can connect to the Genero Studio Server using either SSH, or a socket connection (TCP). Using an SSH connection is recommended for better security and finer user access rights management.

- **SSH connection** - Each user with an SSH connection works with their own server process, as one instance of Genero Studio server is created for each SSH connection. If you choose this connection type, an SSH daemon must be running on the host. There is no need to launch Studio Server on the host; **the SSH connection will automatically launch it.** See Advanced SSH Configuration for additional information.
- **TCP connection** - If you choose to connect via TCP, the Genero Studio Server must be launched on the host before use. On Gnu/Linux and Unix systems It can also be configured to run as a service (init.d, cron).

Next, from the Studio client on your PC, select **Tools>>Configurations**. From the **configuration management** dialog, click the **Configure** button in the lower right corner.

When the **hosts management** dialog displays, right-click on the **Local Host** listing in the Host tree and select **Add new remote host**. Enter the name for your new Studio server; the server name will be appended to the Host tree.

Select your new Studio Server in the Host tree, and complete the **Host Configuration** information.

- **Host:** hostname or IP address where Genero Studio Server is running
- **Port:** Port number used when Genero Studio Server was started
- **Connection type:** Select **SSH** (default) or **TCP**.
- **Installation Directory:** specify the Genero Studio installation directory on the remote machine; enter the complete path, which will always end with **/gst**
- **Login using a user name and password:**
 - **User Name:** enter valid user name
 - **User Password:** enter password
- Select the **Connect** button to check the validity of the host configuration.
- Use the **Import** button to import Genero installations, GAS configurations and environment sets from previous Genero Studio installations. You must connect to the server before importing.
- Confirm or cancel your changes.
- When you add a new host to the Host tree, categories for the details of the remote host are automatically added to the tree.

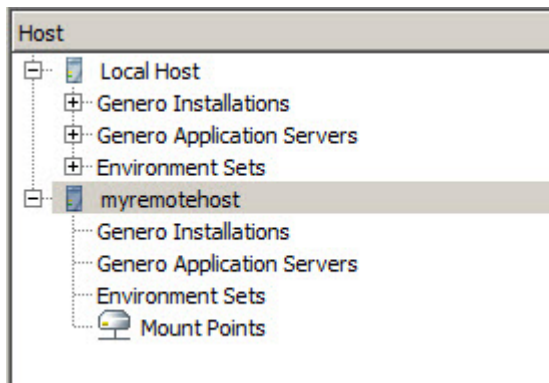


Figure 1-9. Remote Host details in Host treeview

2. Define the Genero Installations
3. Add Genero Application Servers
4. Add Environment Sets
5. Define Mount Points.

Shared Drives allow you to compile, run, and debug files that are on the remote server. Genero Studio uses mapped paths to allow simultaneous access to two machines.

Select **the Mount Points** category under the new remote host in the Host tree to display the Mount Point Table; enter the necessary information.

Mount Point Table details for *Windows Studio client*:

- **Mounted Point:** the mounted drive (network drive that has been mapped to a letter drive on your local machine).
- **Remote path:** the related remote path. For example, if your local computer has a mounted drive **G:** that is mapped to **/usr1/public** on the server **zebra** , the remote path for drive **G:** is; **/usr1/public**.

Mount Point Table details for *Linux Studio client*:

- **Mounted Point:** the location of the file system where the device is attached.
- **Remote path:** the related remote path. For example, if your local computer has a mount point **/mnt/work** that is mapped to **/usr2/public** on the server **zebra**, the remote path for **/mnt/work** is **/usr2/public**.

To **add** a new mount point, click the **Add** button at the top of the Mount Point table. A dialog lists the mount points available for this server; select the desired mount point and enter the remote path.

To **edit** a mount point, select it and click the **Edit** button at the top of the table.

To **delete** a mount point, select it and click the **Delete** button at the top of the table.

Confirm or cancel your changes.

Defining a Remote Configuration

1. Add a New Configuration

Each Genero Studio client that wishes to use a remote Genero Studio Server must have a configuration that accesses the remote server. You may define a number of different remote configurations. All remote configurations can be shared and are usable by each Genero Studio client that adds the remote Genero Studio Server to its hosts list.

In the Configuration Management window, use the **toolbar icon** at the top of the **Configuration Name** table to add a new configuration name:

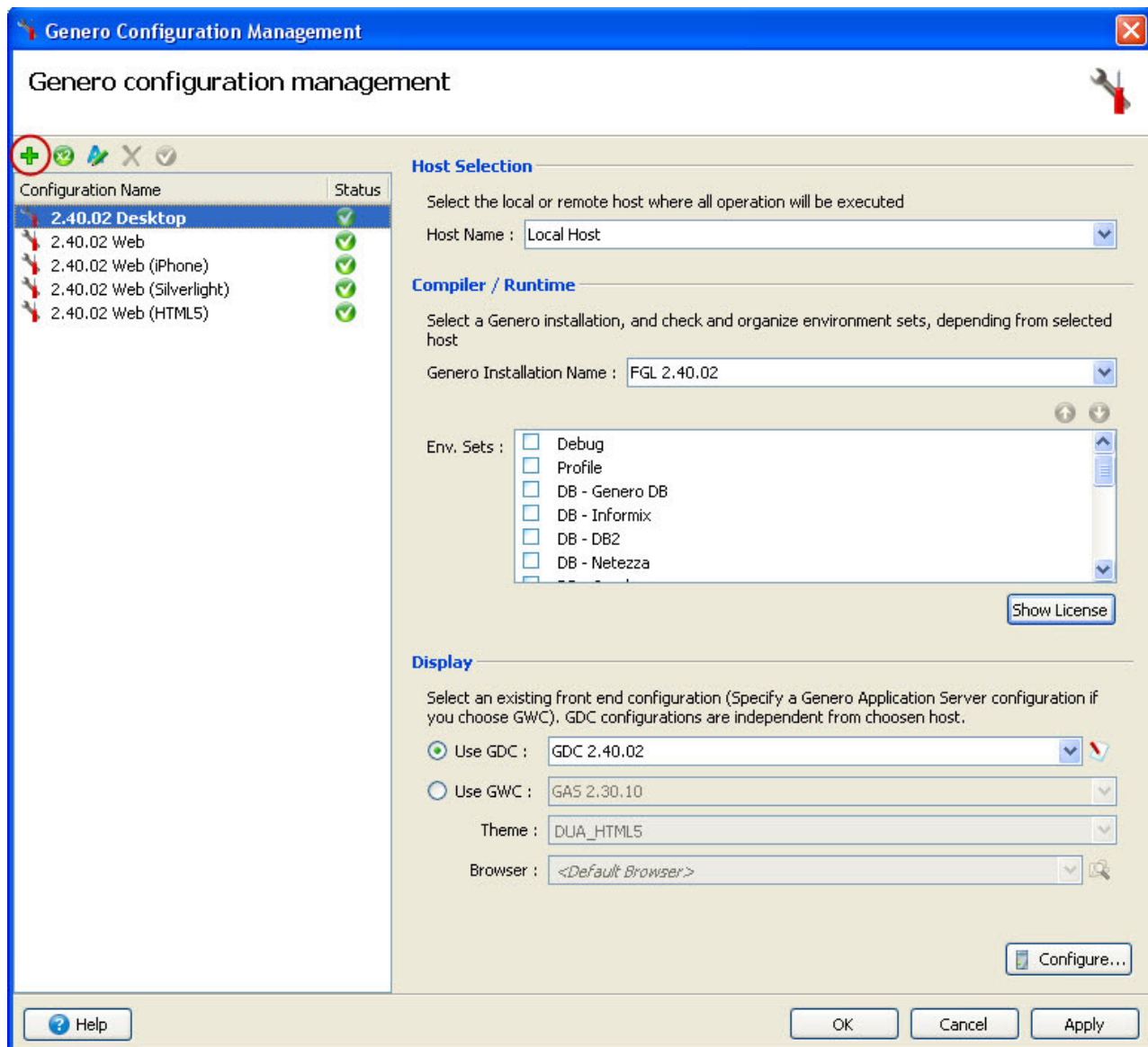


Figure 1-10. Genero Configuration Management window

2. Define the Configuration

Select the new configuration in the Configuration Name list and define the details for the configuration:

- In the **Host Selection** section, select your remote host in the Host Name list box.
- In the **Compiler/Runtime** section:
 - The available Genero Installations will be displayed in the **Genero Installation Name** list box; select your remote Genero installation from that list box.
 - The available **Environment sets** will **Host Selection** be displayed; check the desired sets that you wish to use.
- In the **Display** section, select the Front-End Client that your PC will use.

Click **OK** to save the changes for your new configuration.

Launching Genero Studio Server (TCP connections only)

If you use a TCP connection, Genero Studio Server must be launched before use.
(If you use an SSH connection, Genero Studio Server is launched automatically.)

This program can run as a service and has no user interface; it must be run from a console or terminal. On the server machine, from within *<GeneroStudio installation dir>/bin*, execute the command for your operating system. The default port value is 5321.

- On Windows:
gsserver.exe
- On GNU/Linux:
gsserver

If you need it to listen to another port, you have to use the command-line argument **--port** each time you start Genero Studio Server.

- On Windows:
gsserver.exe --port 1234
- On GNU/Linux:
gsserver --port 1234

For further information, start Genero Studio Server with the **--help** option, and see the **installation notes in the release subdirectory** of the Genero Studio software package.

Studio Text Encoding Settings

Topics:

- Text Encoding
- The Lang Environment Variable

See also: Advanced Character Sets Management

Text Encoding

Text encodings specify the character set to be used by Genero Studio Preferences contains a Text Encoding choice, containing a list of all available encodings. The selected encoding is used during file operations of text documents (load, save, parsing, highlighting, and so on), so the encoding must be correctly set before opening the documents. Once loaded, Genero Studio uses Unicode for text management.

- If you are using Studio in a remote configuration, your encoding method must match the encoding on the Studio server.
- If you are using CVS or other version control systems, the encoding method must be the same for all text stored in the system.
- The database client locale (character set) and application locale (character set) settings must match.

If you change the Text Encoding in Studio, the corresponding LANG environment variable must be set.

The LANG environment variable

While the Text Encoding specifies the character set, the LANG environment variable allows you to compile the Genero files that are using the encoding. Genero Studio Configuration (Tools>>Configuration) allows you to create environment sets containing variables for a specific configuration.

The LANG variable is used while launching Genero internal processes to determine which encoding the file uses.

With the LANG environment variable, you define the language, the territory (country) and the character set (codeset) to be used. The format is:

`<language>_< country>.<codeset>`

On Windows systems, `< language>_< country>` is optional. The system must be configured to support the language defined by the LANG variable.

When a remote Genero Studio Server is used, the LANG variable must match the server LANG variable.

Example LANG settings:

- On Windows: `English_United States.1252`
- On Unix: `en_US.IS08859-1`

Genero Studio 2.x Migration Notes

Topics:

- Version 2.40
- Version 2.30
- Version 2.20

This page outlines some issues that may affect migration in Genero Studio 2.x versions.

Version 2.40

The format of all Studio XML files (.4fd, .4rp, .4pw etc.) has been modified, and you will receive a warning if you execute an application compiled using Studio versions 2.2x or 2.3x. For Genero Report Writer reports only, an error will occur if you have not converted the .4rp file. Opening a file in Studio and saving it will convert it.

Form Designer

For Studio form definition files (.4fd), the `gsform` command-line tool can be used for a batch conversion of the files to version 2.40:

`gsform -c <files>`

Application Generation

The format of the `settings.agconf` file in the templates directory (`<Studio-install-dir>/gst/bin/src/ag/tpl/dbapp/settings.agconf`) has changed. If you have previously modified this file, you must make comparable changes in the 2.40 version of this file.

Version 2.30

Report Writer

- TOP Margin is ignored in compatibility reports.
TOP MARGIN is now ignored when ASCII reports are run in compatibility mode. Margins can be set by using the API function `fgl_report_setPageMargins()`.
- Parameter changed in `fgl_report_configureCompatibilityOutput()`.
The **reportName** value for the `fgl_report_configureCompatibilityOutput` function is now automatically set to the name of the currently running report. When using the function, however, you must pass NULL as the value, for compatibility reasons.
- 4GL Boolean type is now interpreted as Numeric instead of String.
The 4GL **Boolean type** is now regarded as a Numeric type, instead of String as in previous versions. Existing expressions that contain references to boolean variables may require modification accordingly.
- START REPORT syntax must be TO XML HANDLER.
The BDL statement `START REPORT <repname> TO XML HANDLER handler` is now the only valid syntax for starting graphical reports using a .4rp design document.

Version 2.20

Important: Studio Form files (.4fd), Project files (.4pw), and Report files (.4rp) are converted to the Studio 2.20 format when opened. We strongly advise making a backup of your files before migrating them.

Meta-Schema Manager (formerly Database Browser)

1. Database Browser is replaced by Meta-schema Manager.
2. Schema availability has changed.

Form Designer

1. Form files (.4fd) will be converted to the Studio 2.20 form format when opened.
2. The Matrix container replaces the MFArray container.

Project Manager

1. Project files (.4pw) will be converted to the Studio 2.20 project format when opened.
2. The Genero variable `FGLRESOURCEPATH` defines the path to resource files.

Report Writer

1. Report files (.4rp) will be converted to the Studio 2.20 project format when opened.
2. The Reporting API is now linked as one library, `libgre.42x`.
3. The callback function `fgl_report_getFieldCaption()` is no longer called by default.
4. The Configuration menu has moved to `File>>Report Properties`.

Meta-schema Manager (formerly Database Browser)

1. The tool Database Browser has been replaced by Meta-schema Manager; see The Meta-schema Manager.

2. The availability of the schema (.4db) you've extracted from a database, or imported from a SCH file, has changed. (In previous versions of Genero Studio, the schema was always available for all projects you're working on.) Beginning with Studio version 2.20:
 - You can bind schemas to a specific workspace by adding the schema file to an application or library node in the workspace, as you would do with any other source file.
 - You can make the schema available at all times by adding it to the Meta-schema Manager preferences.

We advise adding the schema file (.4db) to your project workspace in a library node. If you use source code control management, the .4db file should be part of the versioned files.

Form Designer

1. On opening Form files (.4fd), they will be converted to the Studio 2.20 form format. We strongly advise making a backup of your files prior to migrating them.
2. The **Matrix container** replaces the **MFArray container**.

Project Manager

1. On opening project workspace files (.4pw), they will be converted to the Genero Studio 2.20 project format. We strongly advise making a backup of your files prior to migrating them.
2. Project Manager now uses only the Genero variable FGLRESOURCEPATH to define where resource files (4ad, 4st, and so on). are located. All occurrences of the Genero variable DBPATH are changed automatically to FGLRESOURCEPATH during conversion to the new format. You must now use FGLRESOURCEPATH instead of DBPATH. However, if Project Manager is used with Genero BDL versions prior to 2.11.06, DBPATH is also used for legacy support.

Report Writer

1. When you open a report file (.4rp) it will be converted to the Studio 2.20 report format. We strongly advise making a backup of your files prior to migrating them.
2. The Reporting API is now linked as one library, libgre.42x, which can be entered in the external dependencies property instead of each individual API filename.
3. Users who overloaded the callback function **fgl_report_getFieldCaption()** from **CaptionCustom.4gl** must now call the API function **fgl_report_setCallbackLocalization(true)** to get the previous behavior. The function **fgl_report_getFieldCaption()** is no longer called by default to provide the values for the "caption" member variable; instead, the lookup in the ".42s" files is performed without calling this function.
4. The Configuration main menu option for configuring a report design document's output has moved to File>>Report Properties.

Advanced SSH Configuration

Topics:

- Configuring Public Key Authentication on Windows
- Configuring Public Key Authentication on Linux

Configuring Public Key Authentication on Windows

Genero Studio can support public key authentication on Windows if you use **Pageant**. You have to install **PuTTY** on your client machine to use it. **Pageant** is a utility that stores unencrypted private keys in memory and provides digital signatures when needed.

You configure it by giving the path of the encrypted private key file (See PuTTY documentation for more information). You enter the passphrase just once when Pageant is launched; then you will not have to bother with it again, until you launch a new Windows session.

Genero Studio uses **plink** to initiate an ssh connection. If Pageant is launched, plink automatically uses Pageant to get a signature. When using public key authentication in Genero Studio, don't fill in the password field in the SSH connection dialog, as it not used.

Using Pageant

First, you have to generate a pair of private\public keys. If you already have a SSH\OpenSSH key pair, see the How to use openSSH key with Pageant section.

- Launch PuTTYgen, and verify that the **SSH-2 RSA radio button** is selected.
- Select the number of bits you want as the size for the key and click the **Generate** button. During generation, move cursor into the blank area to randomize the keys.
- Enter a comment to identify the key pair more easily.
- Enter a passphrase. A passphrase is used to decrypt the private key (Private key is encrypted on the disc for more security). Don't forget it, you will have to enter it each time you begin a new Windows session.
- Once you have entered the passphrase, you can save the private and public keys on the disc.

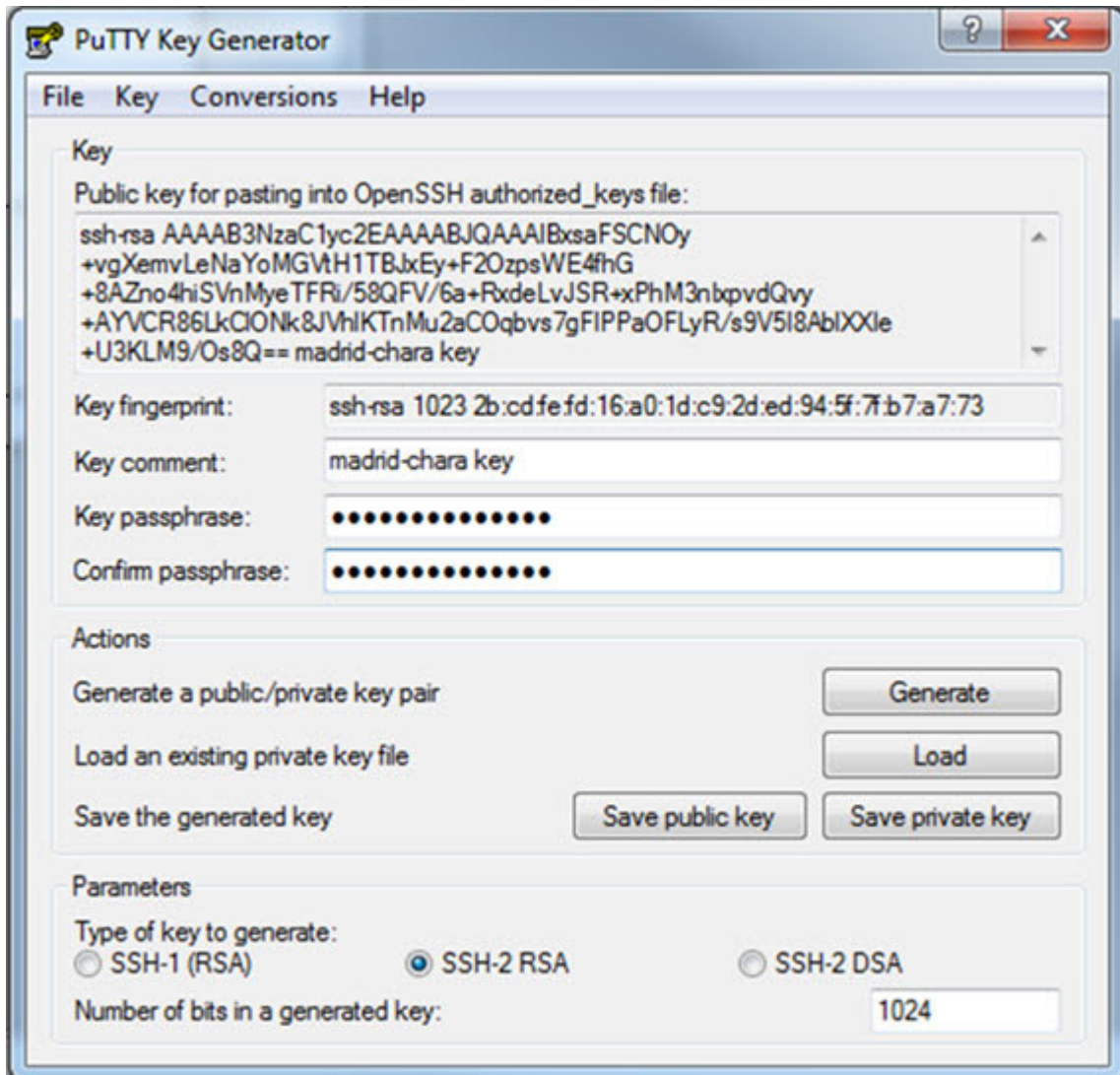


Figure 1-11. PuTTY Key Generator

- Next, go to your home directory (**/home/username**) on your host machine, and edit the **.ssh/authorized_keys** file with your preferred text editor. If the file (or **.ssh** directory) doesn't exist, create it.
- Copy your generated public key in the **authorized_keys** file. The key must be in exactly the same format as displayed in PuTTYgen (**ssh-rsa **** ...****==keyname**) and must occupy only one line. If several lines were already present, enter a line return before writing the key.
- Now launch Pageant. It will appear in the systray as a desktop computer wearing a hat. Right click on it and select **Add key**.

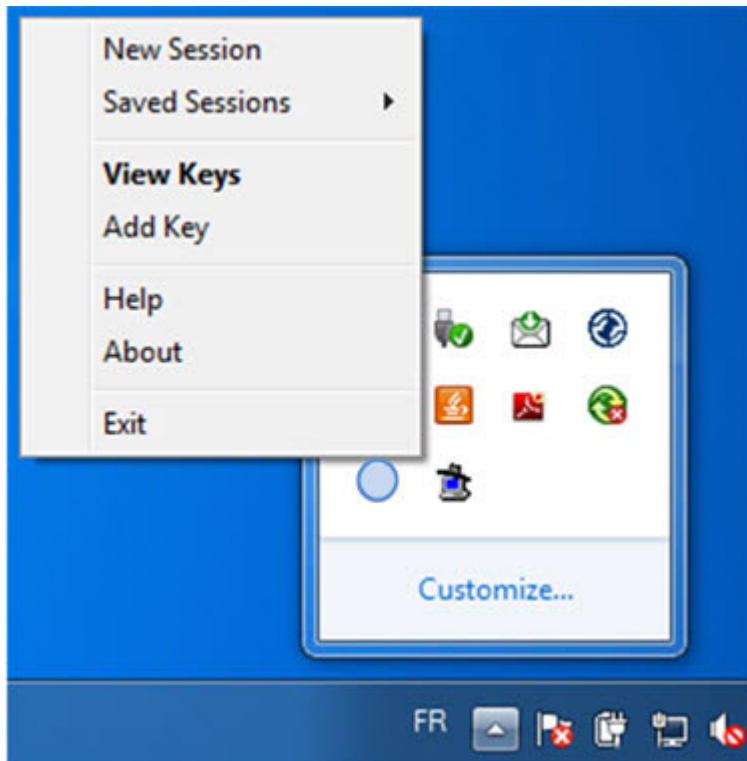


Figure 1-12. Pageant pop-up menu.

- An open file dialog lets you select a ppk (Private key) file. After selecting it, you will be asked to enter the corresponding passphrase.
- You can verify that the key has been successfully added by selecting "View keys" in the Pageant icon's context menu. The key signature is displayed instead of the private key, but you can recognize it by its name.

You should now be able to initiate an ssh connection with Genero Studio without entering a password.

How to use openSSH keys with Pageant

The ssh-keygen generated private key files cannot be directly used by Pageant; you have to convert them into .ppk files. To do so, open PuTTYgen and select **Conversions>>Import key menu item**; select your OpenSSH format private key.

Configuring public key authentication on Linux

Genero Studio uses **plink** ssh client to initiate an ssh connection; plink is traditionally used with the Pageant authentication agent. Since Pageant doesn't exist on Linux, we will use **ssh-agent**, which is released with OpenSSH clients.

ssh-agent stores the private key unencrypted in memory, and provides the private key signature when asked. You configure it by giving it the path of the encrypted private key file (using the ssh-add command). When ssh-agent is launched, plink will automatically ask ssh-agent to get a signature to send to ssh server.

Using ssh-agent

First, you have to generate a private\public key pair. If you already have a PuTTY generated key pair, see the How to use PuTTY keys with ssh-agent section; otherwise, we will generate it with ssh-keygen.

- Type **ssh-keygen -t rsa** in a terminal to generate a key pair using rsa encoding.
- You will be prompted for the file path where you want to save your private key. If you press enter without giving a path, the key will be saved as `~/.ssh/id_rsa`.
- Next, enter the pass phrase (needed to decrypt the private key). Don't forget it as you will have to enter it again at each logon session. The public key file will be saved with the same path as the private key file, but with the `.pub` extension.
- Go to your home directory (`/home/username`) on your host machine, and edit the `.ssh/authorized_keys` file with your preferred text editor. If the file (or `.ssh` directory) doesn't exist, create it.
- Copy your generated public key in `authorized_keys` file. The key must be exactly in the same format as displayed in PuTTYgen (`ssh-rsa ****...****==` keyname) and must occupy just one line. If several lines were already present, enter a line return before writing the key.

Next, configure **ssh-agent** to automatically provide signatures from the private key.

- **ssh-agent** must be started at each logon session. Execute it if it isn't already running.
- Next, execute **ssh-add** with the private key file path as the parameter (For example `ssh-add ~/.ssh/id_rsa`). This will tell **ssh-agent** to test your private key each time you start an ssh session.
- Verify that the key has been successfully added by typing `ssh-add -l`. It should display you your private key fingerprint.

You can now use public key authentication with Genero Studio. For that, create an SSH connection without entering a password:

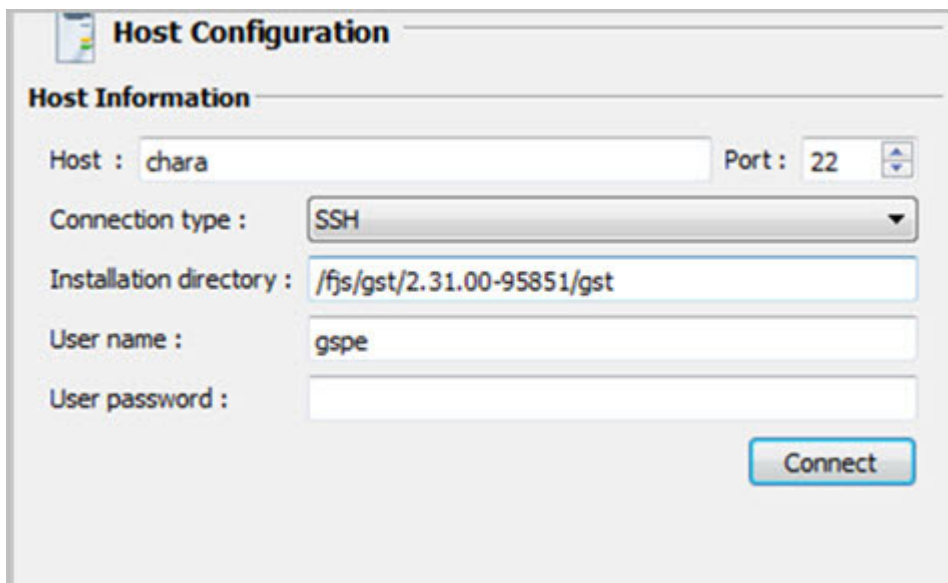


Figure 1-13. Host Configuration

How to use PuTTY keys with the ssh-agent

If PuTTY is installed on Windows, start PuTTYgen, load your private key (ppk file), and select the **conversions >> Export OpenSSH key** menu item. It will ask you where you want to save the file.

If PuTTY is installed on Linux, execute `puttygen input_key.ppk -O private -o output_key`.

Advanced Character Sets Management in Genero Studio

Topics:

- Text Encoding Plugins
- Character Mapping Table
- Adding Plugins
- Example

See also: Studio Text Encodings Settings

Text Encoding Plugins

To support *internationalization* (**I18N**), characters typed at the keyboard are intercepted and changed automatically, based on the Text encoding method selected. The default charset used by Genero Studio is the charset of your local system. Genero Studio Preferences allow you to choose from a list of default supported Text Encodings. In addition, text encoding plugins can be loaded at startup to provide new encoding support. The new names will appear in the Text Encoding choice.

Since the default encodings are much faster than plug-ins, however, users are strongly advised to use the provided encodings unless there are no other solutions.

Character Mapping Table (*GST-install-dir/conf/encodingMap.xml*)

A default list of text encodings is provided in Genero Studio. The mapping between each encoding, the LANG variable in each platform, and the name (alias) commonly used to refer this encoding is defined in the `encodingMap.xml` file in the `Studio-installdir/conf` directory.

For some of the encodings there is only a partial match between Unix and Windows platforms.

This file contains a table used to map each encoding name (alias) to:

- a LANG qualified **country** for Unixes (`unixCountry` attribute)
- a LANG **encoding** for Windows platforms (`win` attribute)
- a LANG **encoding** for UNIX platforms (`unix` attribute)
- an **implementation** name in Genero Studio (`impl` attribute)

```
<Alias name="IBM852" unix="IBM852" win="852" impl="IBM852" unixCountry="pl_PL"/>
```

This means that:

- Under UNIX: `LANG = ${unixCountry}.${unix}`
- Under Windows: `LANG = .${win}`

The encoding is named `${impl}` under all platforms.

The **impl** attribute is defined within the POSIX2 charmap file after the `code_set_name`. It appears in the Genero Studio preferences combobox:

```
<code_set_name> IBM852
```


The **name** attribute is the name Genero Studio should use for this text encoding. By default it is the character set name (code_set_name); when an alias to an existing encoding is needed, the name attribute should contain the alias name.

```
<Alias name="CP852" unix="IBM852" win="852" impl="IBM852" unixCountry="pl_PL"/>
```

Adding Plugins

The **POSIX2 charmaps** are text files describing the text encoding that can be loaded in Genero Studio to add a new encoding for new character sets. During startup, the files in `<GST-install-dir>/conf/charmaps` are loaded as new encodings in Genero Studio, and are available in the General Preferences.

If the corresponding encoding is used for compilation / runtime, `<GST-install-dir>/conf/encodingMap.xml` should be updated.

The plugins should be plain **ASCII POSIX2** charmap files (If the extension is .Z or .gz, they are compressed; you should uncompress them with the correct tool).

Note: Only single byte POSIX2 charmap files are supported in Genero Studio 2.12 and 2.20.

The Format is the following (example for IBM852):

```
<code_set_name> IBM852
<comment_char> %
<escape_char> /
% version: 1.0
% source: IBM NLS RM Vo12 SE09-8002-01, March 1990

% alias CP852
% alias 852
CHARMAP
<U0000> /x00 NULL (NUL)
<U0001> /x01 START OF HEADING (SOH)
<U0002> /x02 START OF TEXT (STX)
<U0003> /x03 END OF TEXT (ETX)
<U0004> /x04 END OF TRANSMISSION (EOT)
```

Example

This example will explain **how to set up Genero Studio client and servers** to be able to use the Polish language. The client will run on a Windows32 platform running in CP1252 character set (Western European languages), and the server on a Linux running in English with utf8 character set.

1. Install

- Choose the encoding and install it: IBM852.
- Check that IBM852 is installed on your linux system: `$ locale-a | grep IBM852`

If you get no result, install IBM852 support; refer to your system documentation.

- Install the following Genero products:
- On Linux (server) :
 - Genero BDL (with web services if required by the project)
 - Genero Studio (GS server)
 - Database (if required by your project)

On Windows (client) :

- Genero Studio

- Genero Desktop Client (GDC)
 - Install support for Polish languages:
In Regional and language options, check 852 (OEM - Latin2)
 - Share a directory between your Linux and Windows hosts. For example:
 - Linux: /shared
 - Win32: Z:
2. Check encoding support
 - Start Genero Studio on Windows.
 - Open Studio Preferences and check the available encodings. Since IBM852 is not supported, you need to add a new codec.
 - Close Genero Studio.
 3. Install the charmap plugin
 - Search for IBM852 POSIX2 charmap on the internet, or on your linux system: (it's often in /usr/share/i18n/charmaps on linux systems). if the file is compressed, uncompress it and copy it to the **conf\charmaps** subdirectory of *<GST install-dir>*. (If the charmaps subdirectory doesn't exist, create it.)
 - Open the encodingMap.xml file in the conf directory and add a new entry for IBM852 (both platforms) :

```
<Alias name="IBM852" unix="IBM852" win="852" impl="IBM852"
  unixCountry="pl_PL"/>
```
 4. Launch Genero Studio
 - Launch the GS server on port 5321.
 - Launch the Genero Studio client
 - Open Studio Preferences and select IBM852. If it's missing :
 - the IBM852 charmap is not present
 - your charmap file is not valid
 - the charmaps dir is not present
 5. Set up the Genero Configuration
Close Studio Preferences and open the Genero Configuration dialog:
 - Set the GDC configuration
 - Open the Genero Hosts Management dialog
 - Add a new host for your linux GS server
 - Set the host and port
 - Set the mount point
 - Add a new environment set
 - Add a variable named LANG with pl_PL.IBM852
 - Close Hosts Management and check the environment set for your configuration
 6. Create a project and compile it
 - Add a new project in Project Manager
 - Add a 4gl file (HelloWorld.4gl on your mount point)
 - Compile the project, and run it

Congratulations, you've added a new character set to Genero Studio clients and servers.

Launching Web applications from Studio

Topics:

- Studio and XCF Files
- XCF Custom Code Examples
 - Custom Snippet
 - Allowed Theme
 - Additional Parameters
- Create and apply a custom XCF for your Web application
- How Studio selects the custom XCF file to use
- Editing and validating a custom XCF file

Studio and XCF files

When launching a Web application, information needed to launch that application is provided in an external application configuration (XCF) file. An XCF file is an external XML file that provides the configuration details for a Web application executed using the Genero Web Client (GWC) front end.

When Studio launches a GWC application, it generates the XCF file and places it in the GAS group directory. If you have explicit changes that need to be made to the generated XCF file - such as specifying a specific snippet to use or parameters to pass to the application - you must create a custom XCF file.

The custom XCF file and procedure described in related topics are for launching a Web application from within Studio. Refer to the Genero Application Server User Guide for information on creating an XCF file and launching Web applications outside of Studio.

XCF Custom Code Examples

These examples show the xml code for specifying specific snippets, themes, or parameters.

Specify a Custom Snippet

Adding the custom snippet ComboBox for an application using the Silverlight theme requires the following additional code in an OUTPUT element:

```
<OUTPUT>
  <MAP Id="DUA_SL">
    <THEME>
      <TEMPLATE Id="_default">c:\mypath\main2.xhtml</TEMPLATE>
      <SNIPPET Id="ComboBox">c:\mypath\SL\combobox.xaml</SNIPPET>
    </THEME>
  </MAP>
</OUTPUT>
```

Specify an Allowed Theme

Specifying an allowed theme requires the following additional code in a MAP element:

```
<MAP Id="DUA_SL" Allowed="TRUE" />
```

Specify additional Parameters

Defining additional parameters requires an EXECUTION element containing the following PARAMETERS element:

```
<EXECUTION>
  <PARAMETERS>
    <PARAMETER>--test</PARAMETER>
    <PARAMETER>-o</PARAMETER>
    <PARAMETER>outputFile.txt</PARAMETER>
  </PARAMETERS>
</EXECUTION>
```

These code segments would be included in your custom XCF file.

Note: DMV and ENVIRONMENT_VARIABLE elements are always generated and will replace any elements you define in a custom XCF file.

Create and apply a custom XCF for your Web application

Follow these steps to create and apply the custom configuration necessary for launching your application as a Web application within the Studio framework.

1. Create a custom XCF file. Include only those elements that you need to add for customization purposes. Example custom XCF file (myHelloWorld.xcf):

```
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.xsd">
  <EXECUTION>
    <PARAMETERS>
      <PARAMETER>--test</PARAMETER>
      <PARAMETER>-o</PARAMETER>
      <PARAMETER>outputFile.txt</PARAMETER>
    </PARAMETERS>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_SL" Allowed="TRUE">
      <THEME>
        <TEMPLATE Id="_default">c:\mypath\main2.xhtml</TEMPLATE>
        <SNIPPET Id="ComboBox">c:\mypath\SL\combobox.xml</SNIPPET>
      </THEME>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

2. Add the custom XCF file to your application's project structure. Your file can have any valid file name, and must end with an xcf extension. Add the file to an application node or dependant library.



Figure 1-14. Project View with Custom XCF file displayed in treeview

3. Launch the application in Studio. The information in the custom XCF file will be applied to the generated XCF file.

How Studio selects the custom XCF to use

Studio searches for a custom XCF file in the application node. If there is no custom XCF file in the application node, Studio will use the first custom XCF file found in the libraries that are dependencies of the application. If multiple custom XCF files are located, the first found file is used.

Editing and validating a custom XCF file

Create the custom XCF file using Code Editor. The Code Editor provides syntax highlighting and auto-completion while editing an XCF document. Validation will be done using the corresponding xsd, and any errors will be displayed.

Legal Notices

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by CollabNet (<http://www.Collab.Net/>).

This product includes software developed by the University of California, Berkeley and its contributors.

This product includes software developed or owned by Caldera International, Inc

Chapter 2. Getting Started

Topics:

- “Genero Studio New Features”
- “Beginning Work” on page 2-4
- “Genero Studio Modules” on page 2-6
- “Using Genero Studio” on page 2-10
- “Getting Help” on page 2-16
- “Genero Studio Error Messages” on page 2-18
- “Working With Existing Genero Applications” on page 2-45

Genero Studio New Features

Topics:

- Version 2.40
 - Code Editor
 - Debugger
 - Project Manager
 - Source Code Manager
 - Database Manager
 - Business Application Modeling
 - Report Writer
 - General
- Version 2.31
 - Report Writer
- Version 2.30
 - Application Generator
 - Report Writer
 - Code Analyzer
 - Code Editor
 - Project Manager
 - Form Designer

Version 2.40

Code Editor

- Better optimization for 4GL and for its use in Remote configurations
- Support for XML Catalogs
- Recognition of [BLOCK], [POINT] for the Application Generator
- Preview and print, on a single page or multiple pages

Debugger

- Optimized for programs with large data in variables, and for use in remote configurations

Project Manager

- Automatically adds dependency information when a project is imported
- View of intermediate files when build rules use these files

Source Code Manager

- Update options, to omit externals for example
- Add mode for adding multiple files

Meta-Schema (Database) Manager

- Easier editing and viewing; options have been added in Edit mode, and document errors/warnings are displayed while editing.
- Multiple views of the same database can be created.

Business Application Modeling

- New Business Records for Reports
- Improved editor with Search, Document Errors
- New templates for a new architecture and for the generation of Report Code
- Advanced customization
- Integrated Code Analyzer

Report Writer

- New Data matrix barcodes
- New properties for Text and Image fields allow them to behave as Hyperlinks
- Additional options for MapCharts
- New report output formats for Excel 2007 (.xlsx) and Word (.rtf)
- New function to create a process level data file and a document at the same time
- New functions to create Document Metadata for compatibility reports
- New functions to provide generic report auto-formatting when no report design document (.4rp) is specified for a report
- Additional new Report API functions, marked "New in 2.40"
- Memory usage reduced by setting a disk cache for documents
- Improved optimization for Report Viewer

General

- Additional Web Client support
- Import setup and preferences of earlier versions at first startup
- Support of high contrast mode and of Windows screen readers for accessibility standards
- Ease of use - the "Windows Menu" on page 2-13 menu allows you to select the default layout, and you can toggle Full Size Documents or Full Screen to maximize your working space
- Multiple instances of Studio can be opened
- States of previous session are remembered
- Export diagrams as an image

Version 2.31

Report Writer

- Support for custom paper sources is provided.
- New RTL class Date provides methods for date formatting and parsing.
- Additional SVG preview options are available.

Version 2.30

Application Generator

- A new Business Application Modeling approach to design is provided.
- Application Diagrams aid design and development of generated applications.
- Design documents are customizable.

Report Writer

- Any report can be output as an Excel or HTML document.
- Subreports can be included in a master report.
- Arbitrary XML data sources can be specified for a report.
- The total number of pages can be displayed on each page.
- Images, and Images from blobs, can be embedded in reports.
- A new element, HTML BOX, has been added to the toolbox.
- Business Charts can be output as tables.
- A new RTL class, the Runtime Class, simplifies the creation of dynamic designs that change behavior based on the runtime setup.
- Additional SVG preview options are available.
- Text for the WordWrapBox report element can be justified.
- Extended scope for variables allows totals to be printed at the beginning of a report.

Code Analyzer

- Code of existing Genero BDL applications can be analyzed.
- Component diagrams and sequence diagrams can be generated.

Code Editor

- XML documents are recognized and can be edited in Code Editor.
- Bookmarks can be added and managed.
- The mode can be set so that you can use vi Editor commands.

Project Manager

- Subprojects - you can now add a library from a different workspace (.4pw) to your current project, using the **Add external dependencies** right-click menu option.

Form Designer

- A new widget, Web Component, can be used in forms.
- A Records tab in the Form Design Window displays the record(s) associated with the form and the properties.

Beginning Work

Topics:

- Getting Started
 1. Choose a Project
 2. Genero Configuration
 3. Add Database Schemas
- Launching Studio Modules
- Getting Help
- Sample Files and Databases

Getting Started

Note: Genero Studio must be first be set up and configured.

When you launch Studio, a **Welcome Page** opens in the central Work Area.

The open Tab, **Projects**, allows you to open a recent Project (file extension .4pw), or to open a recently opened document. Additional tabs at the top of the window give access to:

- **Tutorials and Samples** - sample files distributed with Studio; contains links to open the sample projects
- **News** - RSS feeds containing news about Genero; enable this using the Studio main menu option *Tools>>Preferences*, *Welcome Page*; check the *RSS Section* checkbox
- **Support** - access to Support and Find Answers

1. Choose a Project

In addition to opening recent projects from the Studio Welcome Page, you can choose *File>>Open* to open a new or existing Project.

A **Project** is a file that contains a map of the structure of a project, allowing you to organize the source files and resources of your projects. You can have as many Projects as you need.

When you select a Project, the Project View displays a tree of the project and its components. Right-click on a project node to display a menu of options, or select from the Studio main menu options - select *File>>Open* to continue work by opening a file, for example.

2. Genero Configuration

Genero Studio can be installed:

- by each developer on his machine (local installation)
- on a remote server and accessed by developers from their own client machines (remote installation).

Configurations allow you to define the Studio environment. A default Genero configuration is automatically created and is active at startup. You can change the active configuration, or define additional configurations, using *Tools>>Configurations*.

The first time you start up Genero Studio, the default configuration must be configured, specifying the location and type of the Genero compiler/runtime system, and the location of your Genero Desktop Client. See Configurations.

3. Add Database Schemas

Genero database schema files contain information about the structure of database tables, allowing you to:

- Define formfields in a Genero form definition by referencing a database column
- Define variables in Genero applications as LIKE a database column and have the compiler validate them

To add the schema for an installed database server to your project, see Meta-schema Manager.

Launching Studio Modules

Toolbar icons, or the following menu options, allow you to launch a Studio module:

- Project Manager - from the main menu: *File>>New/Open Project*; automatically launched if you select a project in the Project View
- Form Designer - by opening a new or existing Genero form from the *File* menu; by opening a Genero form file in *File Browser* or *Project Manager*
- Meta-schema Manager- from the main menu: *Database>>New Schema*, or *File>>Open* to open an existing schema
- Code Editor - by opening a new or existing text file from the *File* menu; by opening a text file in *File Browser* or *Project Manager*
- Report Writer - by opening a new or existing Genero Report Design document from the *File* menu; by opening a Genero Report Design document in *Project Manager* from the Welcome Page when Studio opens
- Application Modeling and Generation, by creating or opening a Managed Project from the *File* main menu option; by opening a new or existing Business Application Model diagram from the *File* menu or Project View
- Graphical Debugger - from the *Debug main* menu option; the *Project Manager* Run and Build menus or the contextual menu; from the *File Browser* contextual menu
- Configuration management - from the main menu: *Tools>>Configurations*
- Preferences - from the main menu: *Tools>>Preferences*
- Code Analyzer - right-click an existing Studio application or node in Project Manager and open the desired diagram from the context menu
- Using Web Services - right-click an existing Studio application or project node in Project Manager, and select *Add Web Service* to add a web service to your application.
- Source Code Management - if a Subversion client is installed on your machine, use *Tools>>Preferences* to specify the location of your SVN client. Use the SCM main menu option and right-click context menu in Project Manager to manage version control for your files.
- File Browser - from the Studio main menu: *Tools>>File Browser*
- Graphical Differential - from the main menu: *Tools>>Graphical Differential*

Getting Help

Online help text is available from the **Help>>Help** menu option, the toolbar icon, help buttons on various dialog windows, or by pressing the F1 key. See the Help page for additional information.

Sample Programs and Files

The samples directory (default location is **Documents and Settings/<user name>/My Documents/My Genero Files/samples**) contains the following folders, which have example files and programs, as well as sample databases,

- **Hello World** - contains the **HelloWorld.4pw** project, which has the source code for the very simple Hello World application.
- **BDL tutorial** - contains the **BDLTutorial.4pw** project, which has a set of tutorial programs that illustrate the use of the Genero Business Development Language (BDL), to be used in conjunction with the Tutorial section of the BDL manual.
- **Reports** - contains the **Reports.4pw** project, which has a set of example programs that create reports using Genero Report Writer.
- **DSConfig** - contains the **DSConfig.4pw** project, which has the program source code to complete an fglprofile configuration file with the necessary information for a database selected by the user. An fglprofile file already exists for the sample databases, but you must create one if you choose to use your own database.
- **databases** - contains directories for the sample databases **custdemo** and **officestore**, in both SQLite and Genero db versions. The directories also contain Genero Studio and BDL schema files, as well as a program of SQL commands to recreate the databases for your own database software, if desired. These files are grouped in the appropriate **OfficeStoreSetup.4pw**, and **CustDemoSetup.4pw** projects.

The necessary data for the sample programs is stored in the provided SQLite and Genero db databases.

You can open the project for one of the samples to examine and execute the sample programs. From the Studio Welcome Page, select the **Tutorials and Samples tab**. Links to open the sample projects are displayed under **Samples and Demos**.

Genero Studio Modules

Topics:

- Overview
- Application Modeling and Generation
- Code Editor
- Meta-schema Manager
- Graphical Debugger
- Form Designer
- Project Manager
- Report Writer
- Source Code Management - SVN
- Adding Web Services
- File Browser
- Graphical Differential

- Configuring Genero Studio
- Genero Studio Preferences
- Code Analyzer

Overview

Genero Studio provides a graphical integrated development environment dedicated to Genero users, to help you quickly and easily:

- work locally or across remote resources, with multiple defined configurations
- design a user interface that interacts with a database
- create application forms based on database tables
- create application code that uses the correct syntax
- organize projects
- compile, run, profile, and debug Genero applications
- use version control to manage files
- automatically generate the logic and source code for a database application
- analyze the code and generate function diagrams for existing applications

Application Modeling and Generation

You can create a Genero application without using the Genero programming language, Genero Business Development Language (**BDL**). Genero Studio allows you to automatically generate the logic and source code for a database application to query, add, update and delete rows in database tables. The application handles master-detail relationships between tables, and includes features such as lookup tables. If you are familiar with BDL, you can also add your own logic to the modules.

The Business Application Model is the starting point for your application. Working with existing Studio components, such as Form Designer, Project Manager, and Meta Schema Manager, you can create an Application Diagram for your new application that can be used to communicate with others. Genero forms (application screens) and applications are generated from the Business Application Diagram.

See Chapter 3, “Application Modeling,” on page 3-1.

Code Editor

A fundamental tool of Genero Studio, the *Code Editor* is a smart editor which recognizes the structure of the file being processed (for .4gl files and for other types such as .xml) and highlights it based on the customizable highlighting rules.

Code Editor parses your file and displays a list of errors while you are working on your code. It provides important assistance with syntax (structure highlighting, code completion, and customizable templates). The folding feature makes it easier to work with any text.

See Chapter 5, “Code Editor,” on page 5-1

Meta-schema Manager

The Meta-schema Manager allows you to view the structure of databases associated with your application. You can extract the structure using a database connection, and create database schema files that can be used by your program.

See Meta-schema Manager

Graphical Debugger

The *Graphical Debugger* allows you to test and control the behavior of your Genero application. You can navigate through the functions, and create and manage breakpoints on functions and code lines.

You can choose and group together any variables you specifically want to watch. This gives you the ability to follow a number of variables easily, and even to alter their values while the application is running, for testing purposes.

See Graphical Debugger

Form Designer

The *Form Designer* provides a graphical interface to design and create forms. Widgets, containers, and objects can be added to the forms and their properties can be specified. An interface with databases is provided, allowing you to generate and manipulate form fields from a database schema.

A form created with *Form Designer* can be compiled into the **.42f** format, which is executable by your Genero programs. Existing **.per** forms may be imported and then updated with *Form Designer*.

More on Chapter 6, “Form Designer,” on page 6-1

Project Manager

The *Project Manager* allows you to organize projects into subprojects, nodes, and files. Source files are grouped in program or library nodes, which are used when you build or run applications.

From within Project Manager, you can create or import projects, open source files in Code Editor, open forms in Form Designer, compile and execute programs, execute using the Genero Profiler, and launch Graphical Debugger. A Project file maps the relationships between the nodes in a project.

Creating a new Project file, or opening an existing one, launches Project Manager.

More on Chapter 4, “Project Manager,” on page 4-1

Report Writer

Genero Report Writer introduces several innovative technologies providing power, flexibility and performance for business reports for mission critical applications. It includes the Genero Report Designer (GRD), which creates graphical report layouts, the Genero Report Engine (GRE), which takes raw processed output produced by Genero Business Development Language (BDL), and the Genero Report Viewer (GRV), which takes that output and renders it graphically.

See Chapter 9, “Genero Report Writer,” on page 9-1.

Source Code Management - SVN

The files in a project can be placed under version control, enabling collaborative sharing and maintenance of the files in your Genero projects. Options in the SCM Studio main menu allow you to manage the files using the product **subversion**.

Current and historical versions of the files are stored in a Subversion repository, allowing you to recover older versions of your data, when necessary.

See Chapter 12, “Source Code Management - SVN,” on page 12-1

File Browser

The *File Browser* allows you to delete or rename files, and to create new folders or files. File Browser manages files based on the kind of information that they contain (MIME type file management). *File Browser* will launch the specific action / executable defined as the default for each file type, or you may choose between the actions available for that file type. For example, *Code Editor* opens for a .4gl file and *Form Designer* for a .4fd file.

See “Genero Studio File Browser” on page 13-9

Adding Web Services

You can create a Genero application that calls a Web Service. The *Web Service Wizard* guides you through the process of adding a Web Service to your Genero application.

See “Using Web Services with Studio” on page 13-1

Graphical Differential

The *Graphical Differential* provides a convenient way to display the differences between two files. Select two files (which may be from the same or different folders) from the *File Browser*, and ask for the comparison. The two files are displayed in a split window, which makes it easy for you to navigate from one difference to another.

See “Genero Studio Graphical Differential” on page 13-6

Configuring Genero Studio

Using Genero Studio, you can work locally or across remote resources, with multiple defined configurations.

Configure Genero Studio to match your work environment. You can define:

- Genero Desktop Client configurations, and specify which one is active
- Remote servers to access remote Genero
- local and remote Genero Configurations, and specify which one is active

See “Genero Studio Configuration Dialogs” on page 1-8

Preferences: Customizing Studio Modules

Genero Studio lets you customize the default behavior and layout parameters of Studio modules to match the way you usually work. You can customize parameters specific to each module, as well as general Genero Studio features such as accelerators and menus. The changes can be applied and used immediately, and/or saved as the new defaults.

See “Customizing Genero Studio: Preferences” on page 11-1

Code Analyzer

The Code Analyzer reverse engineers your existing applications and generates Dependency and Sequence diagrams, providing you with an overview of your application.

See Chapter 10, “Code Analyzer,” on page 10-1

Using Genero Studio

Topics:

- Main Window
 - Central Work area
 - Structure View
 - Output View
 - Properties View
 - Bookmark View
 - Listbox in Status bar
- Menus and Toolbars
 - “Windows Menu” on page 2-13
 - All Views
 - Workspaces
 - Context Menus
- File>>New
- File>>Save/Save As

See also: Getting Help

Main Window

When a Genero Studio module is launched, the Main Window is displayed. All other windows and views, the menus, toolbars, and icons used by Genero Studio, are contained within this Main Window.

Right-click in an empty spot in the framework of the top of the window, or use **Window>>Views**, to display a complete list of the views that can be displayed in panels of the window.

Central Work Area

The Central Work Area is where you work with the various Studio modules - creating documents for forms or source files, or displaying documents containing connection or schema information, for example. Each document has its own tabbed page. The Central Work Area may contain as many tabbed views as needed.

- Click a tab to **bring a document to the front**.
- Click the Select a desired document icon at the top right corner to **select a desired document** from the dropdown list.



Figure 2-1. Select a desired document icon

- Click the close icon to **close** the current document.

- Right-click a tab to display the following



Figure 2-2. Close icon

menu:

- Close the **current document**
- Close **all other documents**
- Close **all documents**

The “Windows Menu” on page 2-13 allows you to display various views, providing additional information.

Structure View

This contains a tabbed page for each current Genero document, displaying a structure tree. When you close a document, all related tabs are removed from Main Window.

Drag and drop the Structure view to change its position in the Main Window.

Double-click on the title of the tabbed module page to undock it. This allows the page to float and you may move it aside temporarily. Double-click again to restore the page to its previous location.

Output View

This view is used by any module to display its output. The output of the various modules (messages, errors, results of commands, and so on) is displayed as tabs in the view.

For example:

- **Output tab:** displays all the messages from the execution of Studio. You can filter the messages by checking the **Filter Messages** box at the top of the view, and selecting one of the icons to hide all error messages, all warnings, or all information messages.
- **Document Errors tab:** displays the document error messages. Select the error number and press the F1 key to display additional information about the error. Use Tools>>Preferences, Studio Preferences, Messages to hide a specific information or warning message.
- **Function Search:** allows you to search for a specific function; displays a list of the files containing that function.
- **Task Manager tab:** displays a list of currently running tasks.
- **SVN Status:** displays information about changes to your SVN repository.

You can open any unopened tabs using the **Window>>Views** main menu option.

Use the icons at the top right of the view to display a tab in its own window, or to close a tab.

This view may be also be undocked or re-docked by double-clicking on the title bar.

Properties View

In Studio Form Designer and Project Manager, select an item in the structure view or central work area to display its properties in the Properties View; the property values can be viewed and/or changed.

Bookmark View

This view lists all the bookmarks that a user has added to documents in Studio. You can navigate through them, and add or remove bookmarks, using the toolbar integrated into the view or the new **Edit >>Bookmarks** menu.

The available actions are:

- **Toggle bookmark** - adds or removes a bookmark in the current document, provided the current module supports the feature
- **Previous / Next bookmark** - activates the previous / next bookmark in the view (this should open or raise the associated document)
- **Remove bookmark** - removes the current bookmark in the view
- **Remove all bookmarks** - removes all defined bookmarks

Toggle Document Editing - shortcut to switch to a Document Workspace

Toggle Full Screen display - display the full screen, without changing the Workspace

Listbox in status bar

Select the available Genero configurations from the listbox. The configuration named **default** is displayed at startup.

Click the button at the right of the listbox to display the Genero Configuration Management dialog, allowing you to view and edit the configuration details, or to add/delete configurations.

Menus and Toolbars

Menus and Toolbars are constructed dynamically depending on the context and the currently active module. In addition to the menu options, toolbar icons allow you to perform the tasks related to the various Studio modules:

- The **File>>New** menu option, or the corresponding toolbar icon, automatically launches Form Designer, Code Editor, or Project Manager, for example, when you create new documents.
- **Tools** menu options, or the corresponding toolbar icons, launch Configuration Management, Database Browser, or Preferences, for example.
- The **Help** menu, corresponding toolbar icon, or F1 key launches context-sensitive help documentation.
- The **Window** menu options allow you to close documents or dock/undock views, and to decide the window layout that you prefer.

Other menus and menu options are enabled depending on the context of the Studio module in which you are working.

Toolbars can be dragged and dropped in a different location using the dotted vertical line at the beginning of the toolbar section.

Menus and toolbars are customizable using **Tools>>Preferences**.

Windows Menu

All Views - This menu option allows you to select the views that you wish displayed. Selecting a view from the list toggles the display of the view.

Workspaces - use this option to choose the layout of the workspace:

- **Normal** - displays the usual views - Project, Output, Structure, Properties.
- **Debug** - removes some of the normal views to focus on the Debugger. Studio automatically switches the workspace when starting and stopping a debug session. It keeps track of the current workspace and the one used while debugging, and switches between them when the user starts or stops debugging an application.
- **Document** - displays the Central Workspace containing the document and the Project View, to focus on document editing.

When you select a layout, it is checked. When you select a different layout, the checkmark is moved accordingly.

Context Menus

Right-click a node in the Structure or other View, to display a menu of the actions that are appropriate at that particular point in a Studio module. For example, if you right-click an application node in the Projects View, the menu of actions includes **Execute**, **Build**, and **Debug**.

File>>New

From the Studio File menu, New option, choose the category and the type of item within that category that you wish to create. The corresponding Studio module will be opened (for a .4gl Genero Source file, for example, the Code Editor would be opened.)

Table 2-1. File >> New options

Category	Item Type
Design (Model Driven Architecture)	Project • Managed Project (.4pw) - project for a generated application Application Modeling • Business Application Diagram (.4ba) - application design diagram, used in application generation Database • Schema (.4db) - database schema • Schema from Database (.4db) - extracts schema from database Managed Code - related to application generation • CRUD Form (.4fdm) - create an empty CRUD (Create, Read, Update, Delete) form • CRUD Form from Database (.4fdm) - create a CRUD form generated from database tables • Report data (.4rd) - create empty business record for generated report • Report data from database (.4rd) - create business record for generated report from database tables • Zoom Form (.4fdz) - create an empty zoom form • Zoom Form from Database (.4fdz) - zoom form generated from a database tables Resources • Action defaults (.4fd) - a Genero action defaults file • Toolbar (.4tb) - a Genero toolbar definition file • TopMenu (.4tm) - a Genero TopMenu definition file • Style (.4st) - a Genero Style definition file

Table 2-1. File >> New options (continued)

Category	Item Type
Genero Sources	Project • Simple project (.4pw) Sources • Source (.4gl) - a Genero BDL source code module • • Form (.4fd) - a Genero Studio form definition file • Form as text (.per) - a Genero BDL form definition file in text format • Form from database (.4fd) - calls the Form from Table wizard Resources • Action defaults (.4fd) - a Genero action defaults file • Toolbar (.4tb) - a Genero toolbar definition file • TopMenu (.4tm) - a Genero TopMenu definition file • Style (.4st) - a Genero Style definition file • Startmenu (.4sm) - A Genero Start Menu definition file • Localized String (.str) - Genero Localized String file
Other files	New Files • With no extension • HTML (.html) • Text (.txt) • XML (.xml) • XHTML (.xhtml)
Reports	Report Data • Report from database (.4gl) - to generate BDL code to extract data from database Report Designs • Empty report (.4rp) - a blank report design document • List report (.4rp) - template of a report design document formatted for a list
SOA	• Soap Server - create a .4gl application and server stub from wsdl file

File>>Save/Save As/Save All

When you save a file for the first time, or select **File>>Save As**, the dialog **Save as** opens.

- If a project is open, you can also choose to create a link to the file in a project node or virtual folder.

- **Path:** where the file should be saved in the file system. Click the magnifying glass icon to browse for the path and enter the filename.
- **Insert the file in the project:** if checked, you can choose the application or library node, or a virtual folder, from the list that is displayed in the dialog. Use the plus icons to expand the list.
- If a project is not open, the file can be saved in the file system only. No link can be created in a project node.
 - **Save in:** select the path where the file should be saved in the file system
 - **Filename:** enter the name of the file
 - **Save as type:** change the type of the file, if desired.

If you have previously saved the file, choosing **File>>Save** saves it again with the same parameters. No dialog opens.

The **Save All** option saves the project and its contents.

Getting Help

Topics:

- Genero Studio Help window
 - Toolbar icons
 - Searching the Documentation
 - Search tab
 - Find in text icon
- Browser View
 - Contents tab
 - Index tab
 - Bookmarks tab

Online help text is available from the **Help>>Help** menu option, the toolbar icon, or the F1 key. The relevant documentation page displays in the Help window, depending on your context in the active Studio module. The entire set of documentation is displayed as a tree of links in the Help Browser View. You can synchronize this tree with the currently displayed page using the Sync with Table of Contents toolbar icon.

Genero Studio Help window

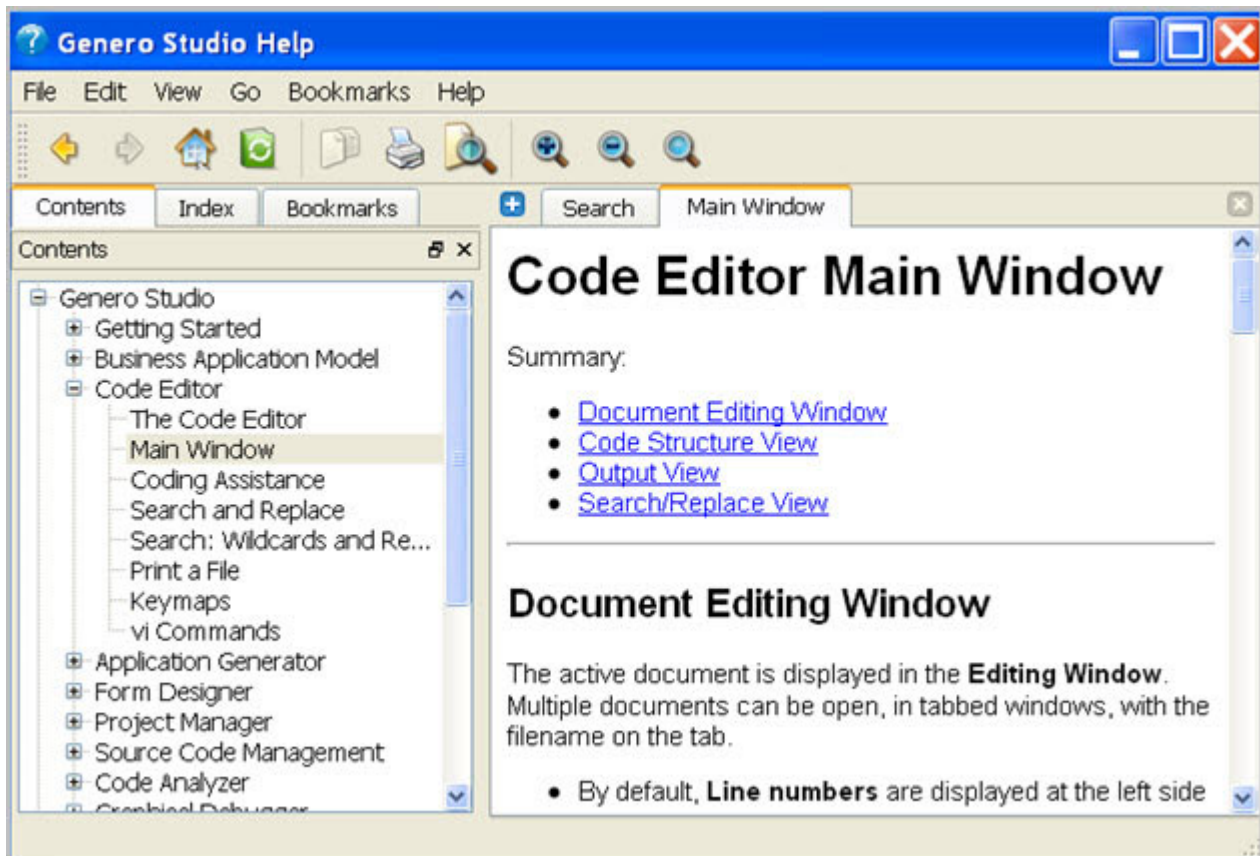


Figure 2-3. Genero Studio Help window

This opens as a separate, movable window, with a Browser on the left.

The main window has two tabs:

- The documentation pages display in one tab, named for the currently displayed page.
- The Search tab allows you to search the entire documentation set for pages containing the text that you specify. Use the Find in Text icon to highlight specified text on the currently displayed page.

Toolbar icons - to help you navigate the documentation

- **Back/Forward** - arrow keys to move back and forth between pages that have been displayed
- **Home** - return to the Studio manual main page
- **Sync with Table of Contents** - expand the tree to display the node for the currently displayed page
- **Copy selected text** - copy text from the currently displayed page
- **Print** - print the page current displayed
- **Find in Text** - search displayed page
- **Zoom In** - enlarge the font size
- **Zoom Out** - decrease the font size
- **Normal size** - return to the default font size

Searching the documentation

Search Tab

This tab is at the top of the documentation window. Enter the text you wish to search for. The entire documentation set is searched. The plus icon allows you to specific advanced search options.

The pages containing the text are displayed in a list on the tab. Select a document to display it in the Help Window.

Find in Text icon

This icon opens a field at the bottom of the currently displayed page, allowing you to search that page for text that you specify. Enter the text to move to the area of the page where the text is found; the text will be highlighted on the page.

Browser View

Contents tab

The links in the tree allow you to navigate through the documentation books and pages. Expand the + icon to display the pages in a book. In addition to documentation for each Studio module, other Genero documentation sets are included.

Index tab

Select from the list of indexed terms, and the corresponding page will be displayed in the Help window. If there is more than one page containing the term, a popup window displays a list of the matching pages so you may select the page that you wish to display in the Help Window.

You can also enter your own text in the field at the top of the tab, and only the indexed terms containing that text will be displayed in the list.

Bookmarks tab

Press **Add** to set a bookmark for the page that is currently displayed in the Help window.

All bookmarks are displayed in a list; select a bookmark to move quickly to the corresponding page.

Select a bookmark and press **Delete** to remove it from the list.

Genero Studio Error Messages

Topics:

- Form Designer Error Messages
- XML Validation Error Messages
- Meta schema Manager Error Messages
- Business Application Modeling Error Messages
- SVN Error Messages
- Business Records Error Messages

Form Designer Error messages

Table 2-2. Form Designer Error messages

Number	Description
GS-10001	Field widget used when no database is attached to form. <i>Description:</i> The current form does not use a database. The Field element is an abstract Formfield that can exist only when a database is attached to a form (the widgets depends on the database column datatype). <i>Solution:</i> • Change the form databaseName property to the corresponding database, • Change the Formfield widget from Field to another widget (Edit, for example).
GS-10002	Database table referenced when no database is attached to form. <i>Description:</i> The current form does not use a database. The Formfield is associated to a database table column, which is invalid. <i>Solution:</i> • Select the Formfield and change fieldtype attribute to non_database • Select the managedForm and set databaseName to the corresponding database
GS-10003	Columns with 'aggregate' must have a widget of type 'Edit'. <i>Description:</i> The column using an aggregate is not an 'Edit' widget type. <i>Solution:</i> • Change the widget type to 'Edit'. • Remove the aggregate.
GS-10004	Error due to fgl not set well. <i>Description:</i> When using gsform, the fgl compiler is not working correctly. <i>Solution:</i> • Check whether FGLDIR is set. • Check whether fgl is licensed and working.
GS-10008	Children are out of container bounds. <i>Description:</i> The children of the parent are out of its boundaries. <i>Solution:</i> • Increase the size of the parent • Decrease the size of the children
GS-10011	Widget shares border or intersects with another widget. <i>Description:</i> Widgets share border or overlap with each other. <i>Solution:</i> • Move them so they no longer intersect or are adjacent. • Delete them.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10012	Invalid widget, upgrade Genero version. <i>Description:</i> The SpinEdit, TimeEdit, Slider and Field widgets were not available prior to Genero version 2.00. The widget webComponent was introduced in Genero version 2.30. The Tree container and Phantom widget were introduced in Genero version 2.20 <i>Solution:</i> • Change version to corresponding valid version or later. • Change widgets to existing widgets with respect to your current version (Edit, for example), or do not use these widgets.
GS-10013	Invalid property, upgrade Genero version. <i>Description:</i> The unitWidth/unitHeight property was added in Genero version 2.00. The doubleClick/splitter/minWidth/minHeight properties were added in Genero version 2.10The image/ contextMenu properties were added in Genero version 2.20. The valueMin/valueMax properties were added in Genero version 2.21. The justify/style properties were added in Genero version 2.30. <i>Solution:</i> • Reset to a supported property. • Change version to corresponding valid version or later.
GS-10016	Layout required for containers. <i>Description:</i> Cannot have two or more containers without a layout in a parent container, except for layout tags (containers in grids). <i>Solution:</i> Layout the containers vertically or horizontally.
GS-10017	Nested grid/scrollgrid cannot contain containers. <i>Description:</i> Nested grid / ScrollGrid (layout tag ScrollGrid) cannot contain following children: Table, Group, ScrollGrid, GridNested Group /Scrollgrid (layout tag Group) cannot contain containers other than: HRec, Table, Tree Solution: Re-arrange your elements using layouts.

Table 2-2. Form Designer Error messages (continued)

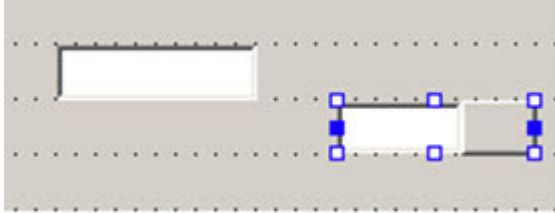
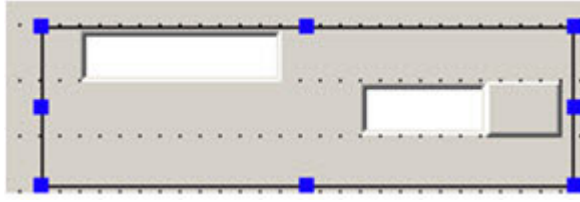
Number	Description
GS-10019	Parent container required. <i>Description:</i> A container is required between the form and the element:  <i>Figure 2-4. Form without container</i> <i>Solution:</i> • Draw a grid or scrollgrid around the element. • Move the element into a Grid, Group, scrollGrid.  <i>Figure 2-5. Form with container</i>
GS-10020	Character not supported in '%1' encoding . <i>Description:</i> The character is not supported by a particular encoding. <i>Solution:</i> • Change the characters to a supported one • Change the encoding.

Table 2-2. Form Designer Error messages (continued)

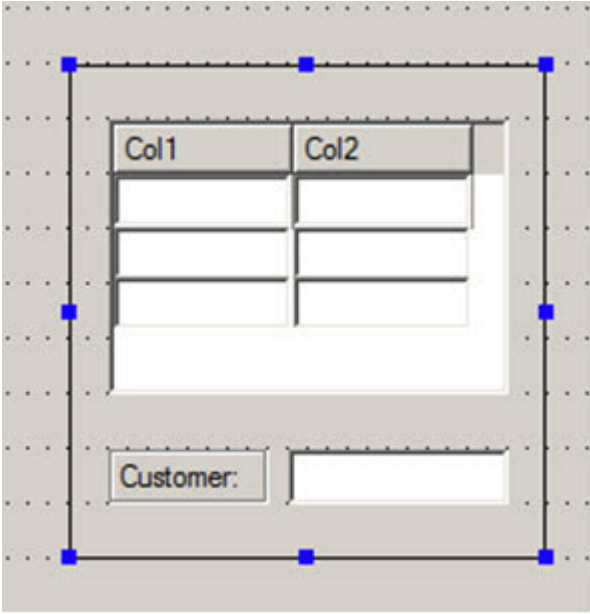
Number	Description
GS-10021	No element below layout tag before Genero 2.01.  <i>Figure 2-6. Form with element inside grid below layout tag</i> <i>Description: You cannot have elements inside the grid below a layout tag (container inside grid.)</i> <i>Solution: Move the elements to another grid and layout both grids vertically.</i>
GS-10022	Layout tag width is name length plus 4. <i>Description: Layout tag minimum size depends on name attribute (name length + 4).</i> <i>Solution:</i> • Increase layout tag width. • Shorten name attribute.

Table 2-2. Form Designer Error messages (continued)

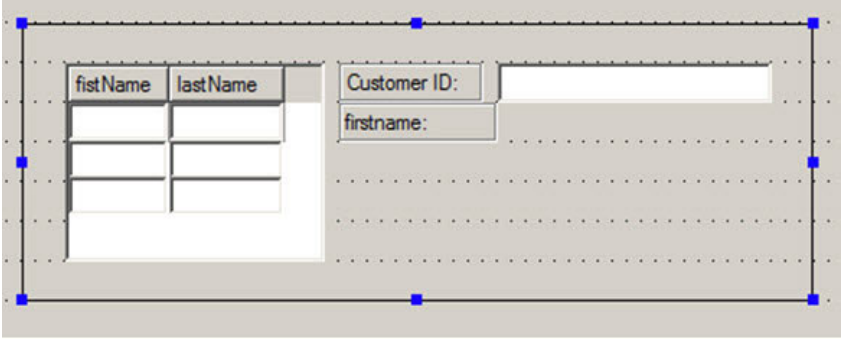
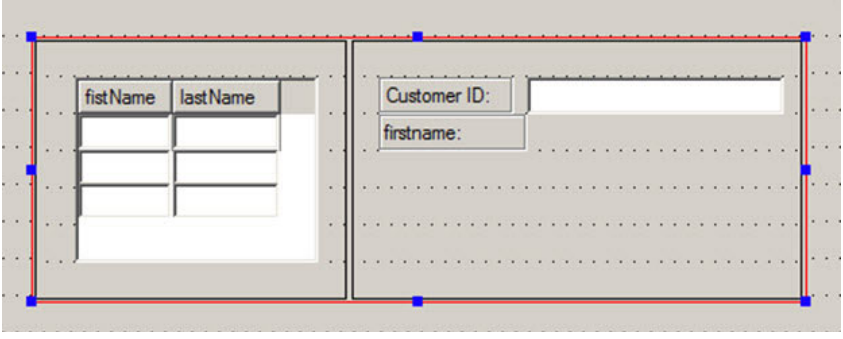
Number	Description
GS-10026	Non container widgets not beside table in grid before Genero 2.10. <i>Description:</i> Cannot have widget beside layout tag table, this feature has been implemented in fgl version 2.10.  <i>Figure 2-7. Form with widget beside layout tag table</i> <i>Solution:</i> • Change Genero configuration fgl to 2.10 or later • Move the widgets into another grid and layout both grids horizontally  <i>Figure 2-8. Form with widgets into another grid and layout both grids horizontally</i>
GS-10027	Empty container <i>Description:</i> This node is missing its required children: • Item is missing in the <i>TopMenu</i> or <i>TopMenuGroup</i> or <i>ToolBar</i> or <i>ActionDefaultList</i> • Widget is missing in group, hrec etc <i>Solutions:</i> • Add an valid child. • Remove the container.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10028	ActionDefault %1 doesn't contain modified attributes applicable to current Genero version. <i>Description:</i> ActionDefaultGroup is defined without any attribute applicable to current Genero version. <i>Solution:</i> • Delete it. • Set one of the following attributes: title, image, accelerators ... • The contextmenu attribute is available since version Genero 2.20, but not before that. • The validate attribute is available since Genero 2.1x, not before that. • The Accelerator4 is available since Genero 2.00, not before that.
GS-10029	Redundant table records. <i>Description:</i> There are two or more records of the table container that are exactly the same, i.e. they have the same order of recordFields. <i>Solution:</i> • Reorder the recordField(s) in record that is duplicate • Delete the duplicate record
GS-10031	Left edge cannot be shared with table right edge. <i>Description:</i> Widget left edge cannot be shared with table right edge. For any Genero version, Genero form compiler needs a space to the right of the table. <i>Solution:</i> Enter a space to the right of the table.
GS-10033	Requires minimum Location.
GS-10034	Requires minimum Size.
GS-10035	Old widget '%1' was removed during import. <i>Description:</i> An old widget (canvas) that is no longer supported was present in the imported form and removed from the document. <i>Solution:</i> Create another element for this widget, or ignore the error.
GS-10036	Old widget '%1' was transformed to '%2' during import . <i>Description:</i> An old widget that is no longer supported was present in the imported form. It's been transformed into the form element. <i>Solution:</i> Check that the transformation was correct, or fix it (some attributes may be incorrectly set).
GS-10040	Expected ValueChecked to be different from valueUnchecked. <i>Description:</i> Checkbox valueChecked and valueUnchecked attributes cannot have the same value. <i>Solution:</i> Change one of the attribute values.
GS-10042	Requested operation cannot be performed, please check the Genero configuration settings. <i>Description:</i> The Genero configuration is not valid. <i>Solution:</i> Check Genero license, directory, and Genero Desktop Client.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10044	Internal error. <i>Description:</i> An unexpected internal error occurred during compilation. <i>Solution:</i> contact your local support Center.
GS-10045	Could not delete temporary .per file. <i>Description:</i> A temporary .per file cannot be deleted during the compilation process, probably because the user opened the file or changed the access rights. <i>Solution:</i> Close the temporary file, or set the access rights to read write.
Gs-10046	Cannot generate unique tag Names. <i>Description:</i> All available field identifiers are used during per file generation. <i>Solution:</i> Enlarge some field's gridwidth.
GS-10047	Unable to create per file during form compilation . <i>Description:</i> During form compilation it is not possible to create a per file.
GS-10049	Value not compatible with data Type. <i>Description:</i> The Value is not compatible with data type set. This error occurs when defaultValue and <i>include</i> property value are incompatible with dataType property. <i>Solution:</i> Change <i>defaultValue/include</i>; enter a value compatible to data type format.
GS-10052	Record cannot start with phantom field. <i>Description:</i> A record cannot start with a phantom field. <i>Solution:</i> Rearrange record fields.
GS-10054	idColumn/parentIdColumn must be defined for Tree. <i>Description:</i> The tree container's properties idColumn or parentIdColumn are undefined, i.e. empty. <i>Solution:</i> Set a valid value, i.e. a name of one of the tree's children, for these properties.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10055	Invalid value set for %1 property. <i>Description:</i> The property value is incorrect. <i>Solution:</i> Check the documentation for the property, and set a valid value. For example, a valid value for the name property is comprised of alphanumeric characters and underscores, but can not start with a number: - For accelerator, accelerator2, accelerator3, accelerator4; value should be a valid accelerator name. - For imageColumn, idColumn, parentIdColumn, expandedColumn, isNodeColumn; value should be one of the children of the table/tree. - For expandedColumn, isNodeColumn, imageColumn; value could be blank, i.e. reset to default. - For Action; value should be a valid action name. - For FieldType; the Field widget cannot be used with FORM_ONLY or FORM_LIKE fieldType; the expected fieldType is TABLE_COLUMN or TABLE_ALIAS. - For formfields; value should be <tableName>.<colName> - For tableAlias type formfields; value should be <tableAliasName>.<colName> - For formonly formfields; value should be formonly.<name> - For non-formfields; value should be a valid name .
GS-10056	Include property doesn't support dataType. <i>Description:</i> An include string is set while dataType is not supported (for example, interval). <i>Solution:</i> Change data type, or remove include string.
GS-10057	Value is out of data Type range. <i>Description:</i> Some data types have a limitation of size (for example, integer). <i>Solution:</i> Change data type (for example, from integer to bigint), or modify value so that it is valid.
GS-10061	Screen record array has different component sizes. <i>Description:</i> Matrices corresponding to the Screen Record need to have the same repeat count (number of rows * number of columns). <i>Solution:</i> - Change the row count or the column count. - If the current screen record is an array and any of the screenRecordItems correspond to a FormField, convert it to a matrix. - If the current screen record is not an array, and any of the screenRecordItems correspond to a Matrix, convert it to a FormField.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10062	All members of \'Record\' must reference the same container. <i>Description:</i> The formfields corresponding to the record field in the same record must belong to the same container. <i>Solution:</i> • Move the record field to the correct record. • Move the formfield to the right container. • Remove the record field or formfield.
GS-10063	First tree column must have a widget of type 'Edit' or 'Label'. <i>Description:</i> The first Tree widget column is not an Edit or a Label. <i>Solution:</i> • Insert an Edit or Label as the first column. • Change the first column to an Edit or Label. • Re-order the columns to have an Edit or Label as the first column.
GS-10064	Invalid/Duplicate name transformed during import. <i>Description:</i> An invalid name that was present in the imported form was transformed to a valid and unique name. <i>Solution:</i> A valid name is not blank, and is made up of the following characters: A-Z, 0-9, "_". You may edit the transformed name.
GS-10065	Multiple line Form title. <i>Description:</i> Genero clients don't support multiple lines. <i>Solution:</i> Remove the carriage return.
GS-10070	Cannot save 4fd file. <i>Description:</i> The .4fd file cannot be saved due to file system permissions. <i>Solution:</i> Check permissions, or move the .per file and import it again.
GS-10076	File not found . <i>Description:</i> A file is missing.
GS-10077	Error loading file . <i>Description:</i> An unexpected error occurred loading the file.
GS-10079	Cannot import file, %1 already exists. <i>Description:</i> The form file already exists, import is cancelled (gsform). <i>Solution:</i> • Delete the .4fd form file • Force overwrite (-i option).
GS-10081	Container is too small, user defined properties will be ignored. <i>Description:</i> The layout tagged container is too small to hold identifier. Any user defined properties will be ignored. This is a warning message. <i>Solution:</i> Increase width of container.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10082	Record has only phantom fields. <i>Description:</i> The record fields are all phantom formfields. <i>Solutions:</i> • Change widget type of a record field Formfield from phantom to some other type. • Drag-drop some record field that is not a phantom type into this record. • Delete record.
GS-10083	valueMin must be lower than valueMax. <i>Description:</i>The valueMin property value must be lower than valueMax property value. <i>Solutions:</i> • Change valueMin so that it is less than valueMax. • Change valueMax so that it is greater than valueMin.
GS-10084	Invalid text. <i>Description:</i> The text property value cannot contain newline characters, '\n' or '\r'. <i>Solution:</i> Delete or replace these characters.
GS-10085	Wrong file extension for import. <i>Description:</i> The file chosen for import has an incorrect extension. <i>Solution:</i> Select a file with a .per extension to import.
GS-10086	Cannot create compilation task. <i>Description:</i> Compilation task creation failure in the build rule. <i>Solution:</i> Check the build rule; if correct, contact your FourJ's Support center.
GS-10087	Invalid widget position. <i>Description:</i> Widget position is invalid for current FGL version used. <i>Solution:</i> Change position of widget.
GS-10088	StepX needs to be greater than 1. <i>Description:</i> The property stepX must have a value greater than 1. <i>Solution:</i> • Edit stepX property. • Uncheck repeat property.
GS-10089	Invalid nested table position. <i>Description:</i> The nested table position is invalid. <i>Solution:</i> Change table position.
GS-10090	Invalid file, mismatching tags . <i>Description:</i> The file is invalid; it contains mismatching tags and XML parsing is not possible.

Table 2-2. Form Designer Error messages (continued)

Number	Description
GS-10091	Invalid file, contains orphan tags . <i>Description:</i> The file is invalid. It contains orphan tags.
GS-10092	Empty space required on all sides of layout tag. <i>Description:</i> Minimum of one unit empty space is required on all sides of layout tag containers. <i>Solution:</i> Resize layout tag container or its children.
GS-10095	componentType property must be defined. <i>Description:</i> The componentType property of webComponent must be defined. It is not set/empty currently. <i>Solution:</i> Set a valid value from available webComponents.
GS-10097	Invalid webComponent. <i>Description:</i> The webComponent is invalid as it does not satisfy XSD file rules. <i>Solution:</i> Open wcsettings file in code editor and rectify in accordance with XSD validation.
GS-10098	Invalid webComponent schema. <i>Description:</i> The webComponent XSD file that is used for validating webComponent XML files is invalid. <i>Solution:</i> • Take a clean copy from of file. • Undo all changes done manually
GS-10099	Empty text widget will be ignored. <i>Description:</i> The text property value will be ignored as it is blank. This is a warning message <i>Solution:</i> Since property value is blank, it plays no role, and is ignored.
GS-10100	Property '%1' conflicts with 'id' or 'parentid'. <i>Description:</i> The value of property 'expandedColumn' or 'isNodeColumn' is same as either 'idColumn' or 'parentIdColumn' of tree widget. <i>Solution:</i> Change the property value of either of the properties so that expandedColumn and isNodeColumn do not have the same value as either idColumn or parentIdColumn.

XML Validation Error messages

Table 2-3. XML Validation Error messages

Number	Description
GS-10400	Invalid root element. <i>Description:</i> The root of the file is not valid. <i>Solution:</i> Edit the root to be <ManagedForm>.

Table 2-3. XML Validation Error messages (continued)

Number	Description
GS-10401	Invalid child %1. <i>Description:</i> The element is not a valid child of the parent. <i>Solution:</i> Edit the child to be an acceptable child of the parent.
GS-10402	'%1' occurs more than once. <i>Description:</i> An element is defined more than once, eg: Topmenu, toolbar can appear just once <i>Solution:</i> Remove the duplicate occurrences to keep only a single valid definition.
GS-10403	Property '%1' occurs more than once. <i>Description:</i> A property for an element is defined more than once. <i>Solution:</i> Remove the duplicate occurrences to keep only a single valid definition.
GS-10404	Unexpected property %1 is ignored <i>Description:</i> This warning is displayed when a property that does not belong to the element is defined. This is ignored. <i>Solution:</i> Remove the property that does not belong.
GS-10405	Incomplete %1 definition, missing property(s) %2 <i>Description:</i> Some mandatory property or properties of an element have not been defined. <i>Solution:</i> Contact your support center - provide the 4fd file, the Genero Studio version, and how this file was created.
GS-10407	Invalid recordField, corresponding formfield not found. <i>Description:</i> The recordField does not correspond to any formfield, i.e. no formfield is present whose fieldId property matches the fieldIdRef property of recordField. <i>Solution:</i> • Check if recordField definition is invalid, and make it valid. • Add a valid formField. • Remove the recordField.
GS-10408	Incomplete Table definition for '%1', missing record. <i>Description:</i> The table does not have a corresponding record defined. <i>Solution:</i> • Check if any existing records are invalid and could be table's record. • Add a valid record. • Remove table.

Table 2-3. XML Validation Error messages (continued)

Number	Description
GS-10409	Invalid record, corresponding table not found. <i>Description:</i> The record does not correspond to any table. <i>Solution:</i> • Check if record definition is invalid and make it valid. • Add a valid table. • Remove the record if not needed.
GS-10410	Incomplete %1 definition, missing mandatory child. <i>Description:</i> The element's mandatory child has not been defined. <i>Solution:</i> Add the missing child.
GS-10411	Invalid geometry for %1. <i>Description:</i> The geometry of element is invalid. <i>Solution:</i> Edit the geometry properties posX, posY, gridWidth, gridHeight such that the element lies within the boundaries of its parent.
GS-10412	Incomplete FormField definition for '%1', missing recordField . <i>Description:</i> The formField does not have a corresponding recordField defined. <i>Solution:</i> • Check if any existing recordFields are invalid and could be corresponding to formfield. • Add a valid RecordField. • Remove FormField.
GS-10413	Malformed XML. <i>Description:</i> The xml structure is invalid. <i>Solution:</i> Check for missing tags.

Metaschema Manager Error Messages

Table 2-4. Metaschema Manager Error Messages

Number	Description
GS-11000	Unexpected error. <i>Description.</i> An unexpected error occurred.
GS-11001	Error while reading < file > schema: < description > <i>Description.</i> An error occurred while reading the file < file> . <i>Solution.</i> The details about the error can be found in the description part of the message.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11002	Cannot find codec. <i>Description.</i> The appropriate codec cannot be found to read a file.
GS-11003	Cannot find codec for < encoding> encoding. <i>Description.</i> The appropriate codec for encoding < encoding > cannot be found.
GS-11050	File not found. <i>Description.</i> The file cannot be found.
GS-11051	Failed to load file < file> . <i>Description.</i> An error occurred while loading the file <file> .
GS-11052	Failed to open file <file> for writing. <i>Description.</i> The file <file> cannot be opened for writing. <i>Solution.</i> Check the file permissions and those of its owner directory.
GS-11053	Failed to open file < file> for reading. <i>Description.</i> The file < file> cannot be opened for reading. <i>Solution.</i> Check the file permissions and those of its owner directory.
GS-11100	Failed to locate node. <i>Description.</i> The specified database object cannot be found in the schema.
GS-11101	Unexpected schema object. <i>Description.</i> An unexpected schema object has been found in the 4db document. The schema file is corrupted and cannot be read.
GS-11102	Missing schema name. <i>Description.</i> The schema name cannot be found. This property is required and the loading of the schema file cannot continue.
GS-11103	Missing schema property < property >. <i>Description.</i> The schema property < property > is required but it cannot be found. The loading of the schema file cannot continue.
GS-11104	Invalid schema property < property >. <i>Description.</i> The schema property < property > has an invalid value. The loading of the schema file cannot continue.
GS-11105	Schema contains no table or column. <i>Description.</i> No valid tables have been found in the schema. A valid table is a table in which "is not flagged" has been removed and which contains at least one column in which "is not flagged" has been removed.
GS-11106	Empty schema file. <i>Description.</i> The schema document contains no information.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11107	File is saved using a newer version of Genero Studio. Please upgrade Genero Studio. <i>Description.</i> The schema document has been saved with a newer version of Genero Studio. You will need to upgrade Genero Studio to be able to open it.
GS-11108	Schema name is not lowercase, its use in 4GL source will generate errors. <i>Description.</i> The schema name contains uppercase characters. Using it in 4GL programs might generate compilation errors. <i>Solution.</i> Use lowercase for the Schema name.
GS-11111	Unexpected table. <i>Description.</i> An unexpected table object has been found in the 4db document. The schema file is corrupted and cannot be read.
GS-11112	Missing table name. <i>Description.</i> The table name cannot be found. This property is required and the loading of the schema file cannot continue.
GS-11113	Missing table property < <i>property</i> > . <i>Description.</i> The table property < <i>property</i> > is required but it cannot be found. The loading of the schema file cannot continue.
GS-11114	Invalid table property < <i>property</i> > . <i>Description.</i> The table property < <i>property</i> > has an invalid value. The loading of the schema file cannot continue.
GS-11115	Schema already contains a table < <i>table</i> > . <i>Description.</i> The table < <i>table</i> > . has already been found in the 4db document. The document cannot contain two tables with the same name.
GS-11116	Table < <i>table</i> > is not found. Creating temporary table. <i>Description.</i> The table < <i>table</i> > cannot be found, and it is required by another database object. <i>Solution.</i> A surrogate table will be created.
GS-11117	Table contains no columns. <i>Description.</i> The table doesnt contain any columns.
GS-11121	Unexpected column. <i>Description.</i> An unexpected column object has been found in the 4db document. The schema file is corrupted and cannot be read.
GS-11122	Missing column name. <i>Description.</i> The column name cannot be found. This property is required and the loading of the schema file cannot continue.
GS-11123	Missing column property < <i>property</i> > . <i>Description.</i> The column property < <i>property</i> > is required but it cannot be found. The loading of the schema file cannot continue.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11124	Table < table > already contains a column < column >. <i>Description.</i> The column < column > has already been found in the table < table >. The document cannot contain two columns with the same name within the same table.
GS-11125	Index < index > already contains a column < column >. <i>Description.</i> The column < column > has already been found in the index < index >. The document cannot contain two columns with the same name within the same index.
GS-11126	Foreign key < key > already contains a column < column >. <i>Description.</i> The document cannot contain two columns with the same name within the same foreign key.
GS-11127	Column < column > is not found in table < table >. Creating temporary column. <i>Description.</i> The column < column > cannot be found in the table < table >, and it is required by another database object. <i>Solution.</i> A surrogate column will be created,
GS-11128	Column < column > in table < table > has unsupported type: < type >. <i>Description.</i> The type < type > is not supported for column < column > within table < table >. An unsupported type is of form encoded=(type, length), and can be generated while extracting database schemas.
GS-11129	Column < column > in table < table > has invalid type: < type > <i>Description.</i> The type < type > is invalid for column < column > within table < table >. An invalid type is not handled by the 4GL language.
GS-11131	Unexpected foreign key. <i>Description.</i> An unexpected foreign key object has been found in the 4db document, The schema file is corrupted and cannot be read.
GS-11132	Missing foreign key name. <i>Description.</i> The foreign key name cannot be found. This property is required, and the loading of the schema file cannot continue.
GS-11133	Missing foreign key property < property >. <i>Description.</i> The foreign key property < property > is required, but it cannot be found. The loading of the schema file cannot continue.
GS-11134	Schema already contains a foreign key < key >. <i>Description.</i> The foreign key < key > has already been found in the 4db document. The document cannot contain two foreign keys with the same name.
GS-11135	Schema already contains a foreign key < key >. Renaming user-defined foreign key to < name >. <i>Description.</i> The foreign key < key > has already been found in the 4db document. The document cannot contain two foreign keys with the same name. <i>Solution.</i> The foreign key will be renamed to < name >.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11136	Foreign key < key > is not found. Creating temporary foreign key. <i>Description.</i> The foreign key < key > cannot be found, and it is required by another database object. <i>Solution .</i> A surrogate foreign key will be created.
GS-11137	Table < table > is not found for foreign key < key >. <i>Description.</i> The table the foreign key refers to cannot be found. The loading of the schema file cannot continue.
GS-11138	Column < column > is not found in table <table> for foreign key < key >. <i>Description.</i> The column column the foreign key key refers to cannot be found in the table table. The loading of the schema file cannot continue.
GS-11139	Foreign key < key > does not contain any table column. <i>Description.</i> The foreign key < key > doesnt contain any columns.
GS-11140	Foreign key < key > columns < column > and < reference > have different data types. <i>Description.</i> The foreign key columns have mismatching data types.
GS-11141	Unexpected index. <i>Description.</i> An unexpected index object has been found in the 4db document. The schema file is corrupted and cannot be read.
GS-11142	Missing index name. <i>Description.</i> The index name cannot be found. This property is required and the loading of the schema file cannot continue.
GS-11143	Missing index property < property >. <i>Description.</i> The index property < property > is required but it cannot be found. The loading of the schema file cannot continue.
GS-11144	Table < table > already contains an index < index >. <i>Description.</i> The index < index > has already been found in the table < table >. The document cannot contain two indexes with the same name within the same table.
GS-11145	Table < table> already contains an index < index> . Renaming user-defined index to < name >. <i>Description.</i> The index < index> has already been found in the table < table> . The document cannot contain two indexes with the same name within the same table. <i>Solution:</i> The index will be renamed to < name >.
GS-11146	Index <index > is not found. Creating temporary index. <i>Description.</i> The index <index > cannot be found in the table < table >, and it is required by another database object, <i>Solution .</i> A surrogate index will be created.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11147	Column < column > is not found in table < table > for index < index >. <i>Description.</i> The column < column > the index < index > refers to cannot be found in the table < table >. The loading of the schema file cannot continue.
GS-11148	Table < table > contains more than one primary index. <i>Description.</i> The table < table > contains more than one primary index. Only one primary index can be defined for a table.
GS-11149	Index < index > does not contain any table column. <i>Description.</i> The index < index > doesn't contain any columns.
GS-11151	Unexpected column validation attributes. <i>Description.</i> An unexpected column validation object has been found in the 4db document. The schema file is corrupted and cannot be read.
GS-11152	Missing column name for validation attributes. <i>Description.</i> The column name the validation attributes refer to cannot be found. This property is required and the loading of the schema file cannot continue.
GS-11553	Can't convert order for column < column > of table < table >. <i>Description.</i> The column < column > of table < table > has an unsupported order value. The import of the sch file cannot continue.
GS-11600	Compilation generated an empty SCH file. <i>Description.</i> The compilation of the schema document generated an empty sch file. This can happen when no valid table columns have been found in the document.
GS-11650	The extraction process task can't be generated. <i>Description.</i> An error occurred while trying to create the database extraction process.
GS-11651	Database name can't be empty. <i>Description.</i> An empty database name has been found while trying to set the database connection up.
GS-11652	Schema file name can't be empty. <i>Description.</i> An empty file name has been found while trying to generate the schema document.
GS-11653	Preparing schema extraction... <i>Description.</i> This message indicates the schema extraction process is being set up.
GS-11654	Schema extraction started... <i>Description.</i> This message indicates the schema extraction process has started.
GS-11655	Merging extracted schema with existing one... <i>Description.</i> This message indicates the schema extraction process tries to merge the existing schema document with the extracted one.

Table 2-4. Metaschema Manager Error Messages (continued)

Number	Description
GS-11656	Generating schema id... <i>Description.</i> This message indicates a new schema identifier is being generated.
GS-11657	Schema extraction successful. <i>Description.</i> This message indicates the schema extraction process has successfully completed.
GS-11658	Schema extraction failed. <i>Description.</i> The schema extraction process has failed at some point. Refer to other messages to get additional information.
GS-11659	Schema extraction aborted. <i>Description.</i> The schema extraction process has been aborted by the user.
GS-11660	Failed to move temporary schema file to destination. <i>Description.</i> An error occurred while trying to move the temporary schema file to its destination.
GS-11661	Failed to create schema backup file. <i>Description.</i> An error occurred while trying to create a backup of the schema file.
GS-11700	The connection test process task can't be generated. <i>Description.</i> An error occurred while trying to create the database connection test process.

Business Application Modeling Error Messages

Table 2-5. Business Application Modeling Error Messages

Number	Description
GS-13001	Cannot load file. <i>Description:</i> The file cannot be loaded; depending on the error, the message can change: - Unknown node: an unknown node is present in the settings file - Cannot register node: a node cannot be added to the file format (it's invalid or already present) - Empty extension: a required file extension is empty. <i>Solution:</i> Check file path, format and permissions.
GS-13002	Cannot save file. <i>Description:</i> The file cannot be saved. <i>Solution:</i> Check file path and permissions.
GS-13003	Template not found. <i>Description:</i> The template path doesn't exist. <i>Solution:</i> Check template directory path.

Table 2-5. Business Application Modeling Error Messages (continued)

Number	Description
GS-13004	Unknown item %1. <i>Description:</i> The setting.agconf template file format is incorrect. <i>Solution:</i> Validate it using the xml schema (or open it in CE) and fix the errors
GS-13006	settings.agconf version %1 is not supported, use version %2. <i>Description:</i> The setting.agconf version 1 is deprecated. <i>Solution:</i> Modify setting.agconf to match version 2 xml schema.
GS-13007	%1 BLOCK(S) or POINT(S) end tag is(are) missing. <i>Solution:</i> Add the missing block / points.
GS-13008	Incorrect end BLOCK or POINT tag type. <i>Solution:</i> Fix the block / point.
GS-13009	End BLOCK or POINT tag does not correspond to an open tag. <i>Solution:</i> Fix the block / point.
GS-13010	%1 with name \"%2\" is already defined. <i>Description:</i> Multiple block/ point definition. <i>Solution:</i> Rename the block / point.
GS-13011	A BLOCK or POINT cannot be a child of a POINT tag. <i>Description:</i> Multiple block/ point definition. <i>Solution:</i> Rename the block / point.
GS-13012	Changes outside BLOCK or POINT ignored between line %1 and %2. <i>Solution:</i> Add a block point in the templates or remove your code.
GS-13014	Adding entries outside BLOCK or POINT not allowed. <i>Solution:</i> Add a blockpoint in the templates or remove your code.
GS-13015	\$(agcomp) is deprecated, please prefer using \$(generate),\$(tcl) and \$(blockpoint) commands. <i>Solution:</i> Modify the build rule to use the new commands.
GS-13016	Modified %1: %2. <i>Description:</i> Prepro block modified.
GS-13017	Unknown property. <i>Description:</i> An unknown property has been set in the settings file. <i>Solution:</i> Remove the property or change its name.

Table 2-5. Business Application Modeling Error Messages (continued)

Number	Description
GS-13100	The 'Open Mode' property is mandatory. <i>Description:</i> The 'Open Mode' property is missing when a relation is defined between modules (.4fdm to .4fdm) and the relation type='Relation'. This should not happen with the provided default settings.agconf. <i>Solution:</i> Check settings.agconf and ensure that the 'openMode' <DynamicProperty> exists in the <BusinessApplication> section and that the initial value is not empty. Check if the 'Type' property of the relation is 'Relation'.
GS-13101	The 'Action' property is mandatory. <i>Description:</i> The 'Action' property is mandatory when a relation is defined between modules (.4fdm to .4fdm) or between module and zoom (.4fdm to 4fdz) and the relation type=Relation. <i>Solution:</i> Check that the 'Type' property of the relation is 'Relation'. Check that the 'Action' property of the relation is not empty.
GS-13102	No report file is defined. Default layout will be used. <i>Description:</i> If the 'Report File' property is empty, the default layout (ASCII mode) will be used when a relation is defined between a module and a report (.4fdm to .4rd) and the relation type='ReportRelation'. This is a warning message. <i>Solution:</i> Check that the 'Type' property of the relation is 'ReportRelation'. Check that the 'Report File' property of the relation is not empty.
GS-13103	The 'Type' property of the relation is invalid. <i>Description:</i> The type of relation is not valid between entities: module (.4fdm), program (.4prg), zoom (.4fdz), report (.4rd). <i>Solution:</i> Check the 'Type' property of the supported relation and ensure that the relation is supported. See Table 2-6 on page 2-40
GS-13104	The relation is not supported. <i>Description:</i> Only certain relationships between entities are supported. <i>Solution:</i> Check that the entity relations, as defined in GS-13013, are correct.
GS-13105	Any relation to a program is not supported. <i>Description:</i> A relationship from any entity to a program is not supported. <i>Solution:</i> Check that the entity relations, as defined in GS-13013, are correct.
GS-13107	A unique key field must point to the master table of the record. <i>Description:</i> All fields making up the unique key of a record must be fields of the master table of the record. <i>Solution:</i> Uncheck the Unique Key property of fields which are not of the master table of the record.

Table 2-5. Business Application Modeling Error Messages (continued)

Number	Description
GS-13108	Duplicate table name in the FROM clause. <i>Description:</i> The joins between a pair of tables must have the same join operator. <i>Solution:</i> Check that there are not multiple join operators for the same pair of tables.
GS-13109	Field updated via ascending lookup is mandatory in the record. <i>Description:</i> The field that is updated via an ascending lookup is mandatory in the master table of the record. <i>Solution:</i> Add the field to the master table of the record. Currently, the field is only defined in the 'Query' clause of the record. If you don't want to see this field on your form, use the 'phantom' widget. Or, delete the 'lookup' property of the field which triggers the ascending lookup.
GS-13112	The 'zoom' action in the toolbar/topmenu won't be generated, an action is already named 'zoom'. <i>Description:</i> An action opening a zoom form is already named 'zoom', therefore the 4GL code for the global 'zoom' action of the toolbar/topmenu will not be generated. <i>Solution:</i> Rename your action to a name other than 'zoom'.
GS-13113	Several relations use the same action name. <i>Description:</i> Outgoing relations of a form can not have the same action name. <i>Solution:</i> Rename action names to use unique names.
GS-13114	SQL statement 'FULL OUTER JOIN' is vendor proprietary SQL syntax.

Table 2-6. Expected relationship between source and destination entities:

Source	Destination: Program (.4prg)	Destination: Module (.4fdm)	Destination: Zoom (.4fdz)	Destination: Report (.4rd)
Program (.4prg)	N/A	R	N/A	N/A
Module (.4fdm)	N/A	R	R	RR
Zoom (.4fdz)	N/A	N/A	N/A	N/A
Report (.4rd)	N/A	N/A	N/A	N/A

Legend:

- R: type='Relation'
- RR: type='ReportRelation'
- N/A: Not Available

SVN Error Messages

Table 2-7. SVN Error Messages

Number	Description
GS-22001	SVN Tool error.
GS-22003	Cannot start SVN Tool. SVN Tool is not properly set in preferences. <i>Solution:</i> in Tools>>Preferences, check the Source Code Management >> Subversion page to set and check the SVN tool path.
GS-22004	SVN Add command failed.
GS-22005	SVN Commit command failed.
GS-22006	SVN Export command failed.
GS-22007	SVN import command failed.
GS-22008	SVN Info command failed.
GS-22009	SVN mkdir command failed.
GS-22010	SVN remove command failed.
GS-22011	SVN rename command failed.
GS-22012	SVN rename URL command failed.
GS-22013	SVN resolved command failed.
GS-22014	SVN revert command failed.
GS-22015	SVN log command failed.
GS-22016	SVN status command failed.
GS-22017	SVN update command failed.
GS-22018	Apply patch failed.
GS-22019	Server certification error. GST displays a dialog to get the user input. You can select to: • Accept permanently - You will not be asked for the server certificate for the same repository again. • Accept Once - accepts the certificate only for the current operation. For the next operation, this dialog will appear again. • Reject - to cancel the operation.
GS-22020	SVN Authentication failed error. <i>Description:</i> The current user does not have access to the SVN repository. <i>Solution:</i> Provide a valid user name and password in the popup dialog box.

Business Records Error Messages

Table 2-8. Business Records Error Messages

Number	Description
GS-24001	Error loading file <file>. <i>Description.</i> An error occurred loading the file. <i>Solution:</i> Check the file name and permissions .

Table 2-8. Business Records Error Messages (continued)

Number	Description
GS-24002	Malformed xml. <i>Description:</i> xml file content is invalid. <i>Solution:</i> Select the correct file
GS-24003	File <file> not found in Business Application Diagram (<BA file name>). <i>Description:</i> A referenced element is not found in the business application diagram. <i>Solution:</i> Select the correct file.
GS-24004	Invalid value for "%1" property. <i>Description:</i> An invalid value is set for the given property. <i>Solution:</i> Check the property syntax in the documentation and correct any errors.
GS-24201	Missing master Record. <i>Description:</i> The document must contain at least one record that is master. <i>Solution:</i> Create a record.
GS-24202	Unique query key must be set. <i>Description:</i> At least one field must be declared the unique key in a record. <i>Solution:</i> Set a unique key.
GS-24005	Conflicted item. <i>Description:</i> The Business Application diagram and the Business Record unique id do not match. <i>Solution:</i> Open the 4ba file and resolve the conflict.
GS-24203	Record used in relation must be active. <i>Description:</i> Records linked with a relation must be active. <i>Solution:</i> Change the record to active, or remove the relation.
GS-24204	Empty relation. <i>Description:</i> This relation has no field definition. <i>Solution:</i> Add foreign/primary (or source/destination) fields.
GS-24205	Field <field name> is present in multiple active records. <i>Description:</i> A field appears in multiple active records in the same document. <i>Solution:</i> Use table alias field type, or rename the field. If the field is not used by AG, make the record inactive or delete it.
GS-24206	Missing master table for <record> record. <i>Description:</i> The record has no master table. <i>Solution:</i> Set the master table property.

Table 2-8. Business Records Error Messages (continued)

Number	Description
GS-24207	Table %1 unused in query. <i>Description:</i> A database table is used in the record, but there is no join for it in the query. <i>Solution:</i> Add a join for the table in the record query.
GS-24208	No schema attached, <i>Description:</i> A database is required for the business records. <i>Solution:</i> set the database name property to an existing database.
GS-24209	Invalid relation, <primaryField> and <foreignField> must have same number of fields. <i>Description:</i> A relation must have the same number of primary and foreign fields. <i>Solution:</i> Modify the relation fields so that a primary field corresponds to each foreign field.
GS-24210	Invalid query, left and right join must have same number of columns. <i>Description:</i> A query must have the same number of left and right fields . <i>Solution:</i> Check the query fields so that a left field corresponds to each right field.
GS-24211	Relation types don't match exactly. <i>Description:</i> The type of a foreign key doesn't match the corresponding primary key's type. <i>Solution:</i> Check the relation, change the field types, or fix the primary / foreign key.
GS-24212	Relation field <fieldname> not found. <i>Description:</i> The relation refers a field that is missing in the record. <i>Solution:</i> Modify the relation or add the field to the record.
GS-24213	Relation field <fieldname> not found. <i>Description:</i> The relation refers a field that is missing in the record. <i>Solution:</i> Modify the relation or add the field to the record.
GS-24214	Nonexistent table <table> referenced in document. <i>Description:</i> The document uses a database table that is not present in the schema. <i>Solution:</i> Update the schema, or change the field using the table.
GS-24215	Nonexistent column <table>.<column> referenced in document. <i>Description:</i> The document uses a database column that is not present in the schema. <i>Solution:</i> Update the schema or change the field using the table column.

Table 2-8. Business Records Error Messages (continued)

Number	Description
GS-24216	Database table defined in "no database" document. <i>Description:</i> A database field is used in a no-database document. <i>Solution:</i> Set the database property to an existing schema or remove the field.
GS-24217	Invalid alias <alias>. <i>Description:</i> An invalid table alias name is used; the alias must be a valid symbol, it cannot be empty or an existing table name. <i>Solution:</i> Change the table alias name.
GS-24218	Alias <alias> referenced for different tables in document. <i>Description:</i> The same alias is used for different tables. <i>Solution:</i> Rename the alias so that it refers to the same database table.
GS-24219	Alias Lookup field is ignored on "%1" as it is the master table of the record. <i>Description:</i> A lookup field is dedicated to foreign field update in the master table; do not set it on a master table column. <i>Solution:</i> Remove the lookup property value or update the field DB settings.
GS-24220	Name value "%1" is already used. <i>Description:</i> Duplicate name used in the document. <i>Solution:</i> Rename the element so that the name is unique.
GS-24221	Invalid INTERVAL qualifier. <i>Description:</i> The qual1 or qual2 set for the INTERVAL sqlType is not valid. It should belong to the INTERVAL classes (i.e., YEAR-MONTH or DAY-TIME) <i>Solution:</i> Change either qual1 or qual2 to fit the respective class range or change the sqlType property from INTERVAL to another sqlType.
GS-24222	Startfield of DATETIME or INTERVAL qualifiers must come earlier in the time-list than its endfield. <i>Description:</i> The qual1 value should be greater than the qual2 value, when the sqlType is either DATETIME or INTERVAL. <i>Solution:</i> Ensure qual1 > qual2 or change the sqlType property from INTERVAL to another sqlType.
GS-24223	Query properties set without attached database schema. <i>Description:</i> Some query properties (join, order, additional tables or where) are defined without a database schema attached. <i>Solution:</i> Clear the query properties or attach a database schema to document.

Table 2-8. Business Records Error Messages (continued)

Number	Description
GS-24224	Table '<table>' referenced by '<property>' is not present in the record. <i>Description:</i> A table is referenced in a query property (join or order), but is not referenced in any record's field or additional tables. <i>Solution:</i> • Remove the join or order that references this table. • Add a field referencing a column from this table to the record. • Add the table to the 'additional properties' property. • In the join or order, change the table to another one that is present in the record.

Working With Existing Genero Applications

Topics:

- Using Studio in a Remote Environment
 - Accessing the files
 - Compiling on the server
 - Using a VCS to share files in a team
- Editing the files
- Using the Form Designer
- Using the Debugger
- Importing Existing Projects into Project Manager
- Integrating Your Tools into Studio

See Also: Setting Up Studio, Configuring Studio, Studio Project Manager

Using Studio in a Remote Environment

Genero Studio can be used in client/server mode accessing remote servers. The files of existing Genero projects stored on the remote server can be accessed to create, compile, and build project files, as well as execute and debug applications. Studio Project Manager provides an integrated environment that makes it easier for you to manage your projects; the Studio tools can also be used individually.

Accessing the Files

To access the files within Studio:

1. Export the file system on your server using samba/NFS/Windows Share
2. Mount the file system on your client

Your files will now be available to Studio.

Compiling on the server

1. Start Genero Studio server (gsserver) on the server machine
2. Set up the remote Genero configuration
3. Configure the mount drive table; the path entered is the path as viewed from the Studio server

You can now compile files on the server.

Using a VCS to share files in a team

Set up a VCS tool on a server and integrate your sources in the VCS repository. Your Genero Studio projects (**.4pw** and **.deps**) should be in the repository as well.

Check out the sources of your application on the server in the directory that you have exported (using samba, NFS, Windows Share). Each user has their own checkout.

See Studio Source Code Management.

Editing the Project Files

If your files are stored on a remote server, you must set up access to them. Then, the Studio Code Editor is automatically launched when you use the **File>>Open** option from the Studio main menu and browse for a file that is editable (**.4gl**, **.txt**, etc.). Or, you can right-click the filename from File Browser or Project Manager and select **Open** from the menu.

Using the Form Designer

If you want to take advantage of the features of Genero Form Designer, you must convert your existing Genero **.per** form definition files to the Studio format. Choose **File>>Import Form** from the Studio main menu to import the form.

The converted file will have an extension of **.4fd**, and will compile into the usual **.42f**. The **.per** file is no longer the source file and the file extension has been automatically changed to **~per**. We recommend that this file be removed from the directory. You can keep a backup copy of your **.per** file of course, but the **.4fd** and **.per** files cannot both be present.

Once the form file is converted, you can launch Form Designer from the Studio main menu option **File>>Open**, or right-click the filename to display a menu of options.

Using the Graphical Debugger

The Studio Graphical Debugger can be used on any project as long as you have the binaries (**.42r**, **.42m**, **.42f**) and the sources (**.4gl**) available.

If the project source files have been imported into Project Manager, build the project to create the binaries, and then right-click the Application node and select **Debug**.

To debug from the File Browser:

- Set up the appropriate **FGLLDPATH**, **FGLDBPATH**, **FGLSOURCEPATH** and **DBPATH** environment variables in the Genero Configuration so Studio can locate the binaries and sources.
- Use File Browser to browse the file system for the project files.
- Launch the debugger by right-clicking on the **.42r** file in the Folder contents tab and selecting the **Debug** menu option.

Importing Existing Projects into Project Manager

1. Create a database schema for the database tables referenced in the project, using the **Database** main menu option; you can import an existing Genero **.sch** file, or set up a connection to the database to extract the schema directly.
2. Create an empty Project to manage the files in the project using Studio Project Manager.
3. Use Project>>Import Project from the Studio main menu; to **import the directory structure as a new project** use the **Import** icon at the top of the Project Manager (applications will be detected automatically). To make sure that only sources appear in the project, either clean all binaries and generated files from the project prior to importing, or set the correct import file filters.

Once the project is successfully imported:

- If your sources include files that are preprocessed to generate other files, set up the appropriate Build Rules.
- Check the appropriate dependencies in Advanced Properties for each application or library.
- The Target Directory property of the project nodes is set by default to *ProjectDir/bin*. Reset this property, and set any additional properties (Database Schema, for example), if needed.
- Build the project to check for dependency errors.

Integrating your Tools into Studio

To call your tools and scripts directly from Genero Studio, you can customize global menus and toolbars and also the context menus that are used in the Project Manager and File Browser.

You can set up your commands to ask for parameters (using input variables) or use the preset parameters proposed by the Studio environment (current file, current directory, and so on)

All commands run in the current environment set by the Genero Configuration or the Project. The working directory is set to the location of the file to which the command applies.

Chapter 3. Application Modeling

Topics:

- “Introduction to Genero Studio Business Application Modeling”
- “Your First Application (the dbApp template)” on page 3-9
- “Business Application Model Diagram” on page 3-22
- “Creating CRUD Forms” on page 3-28
- “Genero Report Writer reports” on page 3-32
- “Lookup fields and ComboBoxes” on page 3-35
- “Program Flow: Opening Zooms and Additional Forms” on page 3-43
- “Actions and Action Views” on page 3-50
- “The Application Source Code and Files” on page 3-55
- “Modifying the Application” on page 3-59
- “Customization” on page 3-62
- “Application Generation Templates” on page 3-66

Introduction to Genero Studio Business Application Modeling

Topics:

- A Model-Driven Approach
- BAM Diagrams
- Application Generation templates
- BAM Tools
- How Code Generation Works
- Application Generation Requirements
- Sample databases and files
- Deploying the Application

A Model-Driven Approach

Business Application Modeling allows you to develop business applications from design diagrams rather than writing some code. Use Studio Business Application Modeling (BAM) to model the application's features and behavior:

- Model your business applications using diagrams describing the functionality of the applications.
- Model the components of your applications and their interconnection in a high level business diagram.
- Model the detailed design of your database, forms, services in sub-diagrams
- Model the structure of the database table columns in a subdiagram to be used in generating code for reports.
- Automatically generate an application matching the behavior described in the diagrams without coding.
- Enhance the generated code with your own logic.
- Generate and output reports to a printer or in various output formats.

In Studio we have provided a complete framework for design and development, incorporating the entire life cycle of an application:

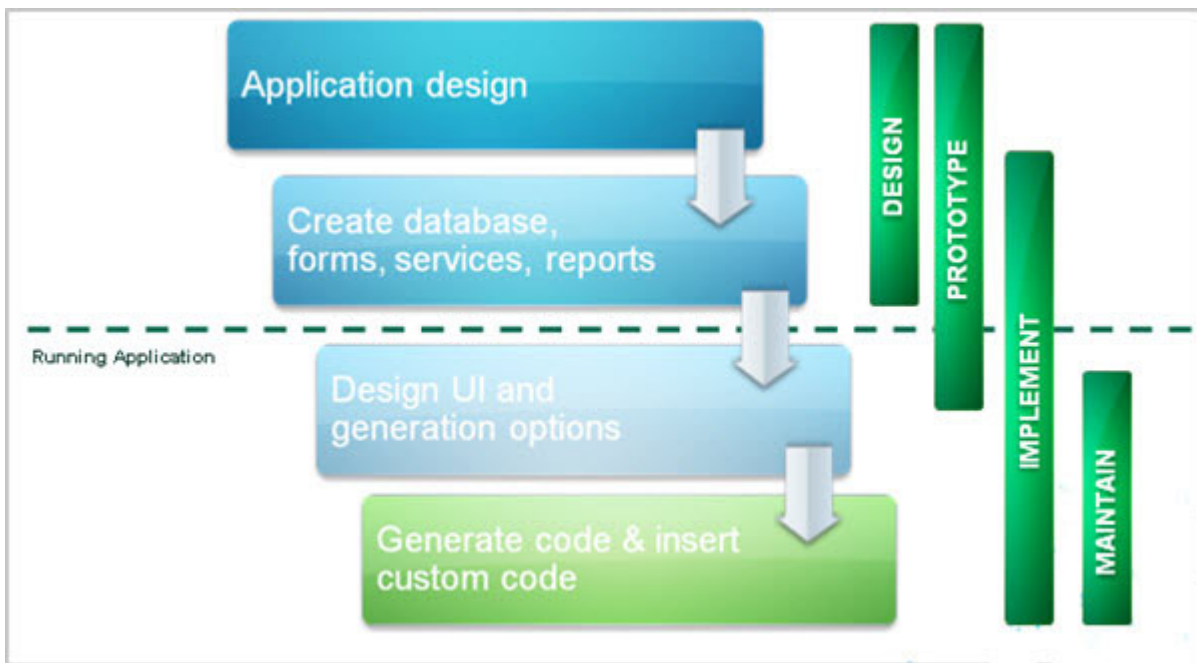


Figure 3-1. Application Lifecycle

Process:

1. Application Design - create the diagram to model the application design
2. Create database, forms, services, reports - create the diagrams that model these components of the application
3. Design UI and generation options - select the options for your application
4. Generate code and insert custom code - generate code based on the models, and customize it if desired

BAM Diagrams

Diagrams are used at all stages in the development. They serve as inputs for code generation, and are considered source code for the application. They are always up to date.

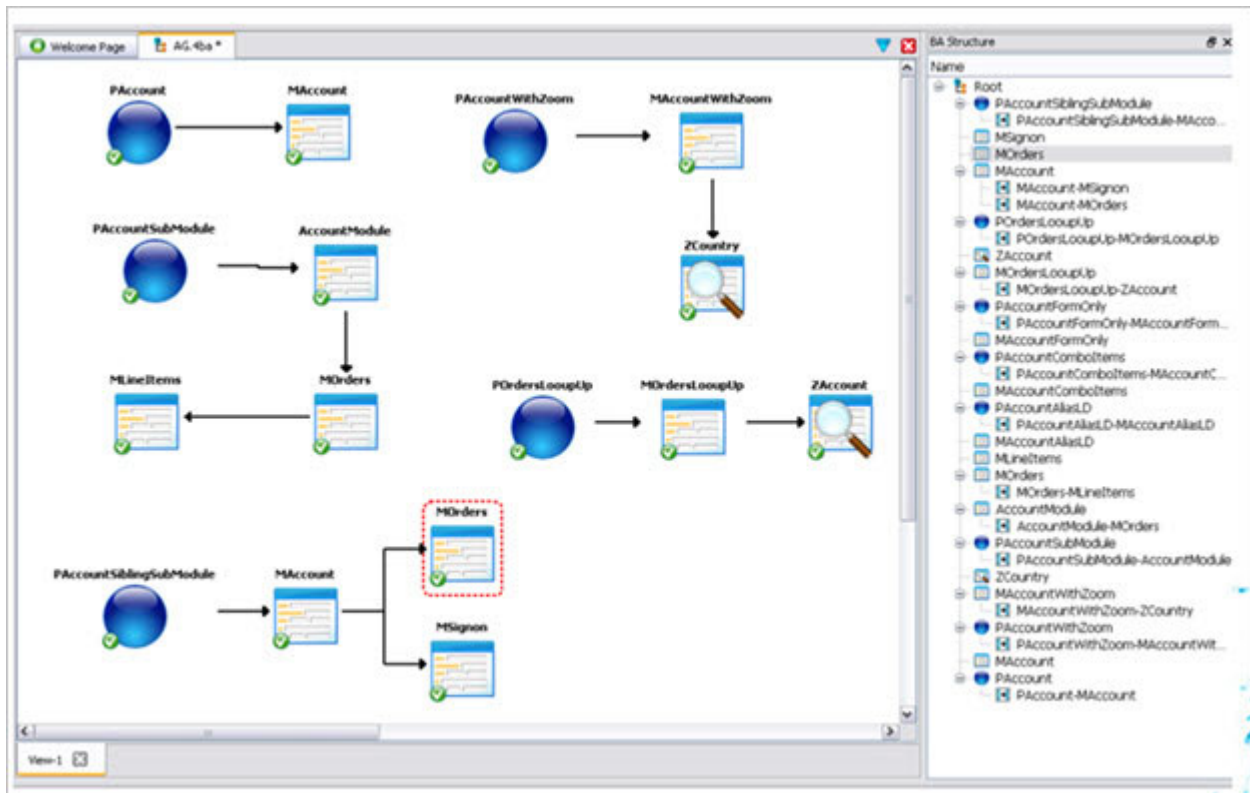


Figure 3-2. Business Application Model diagram

The Business Application Model diagram models the application flow and provides a high-level overview of the application

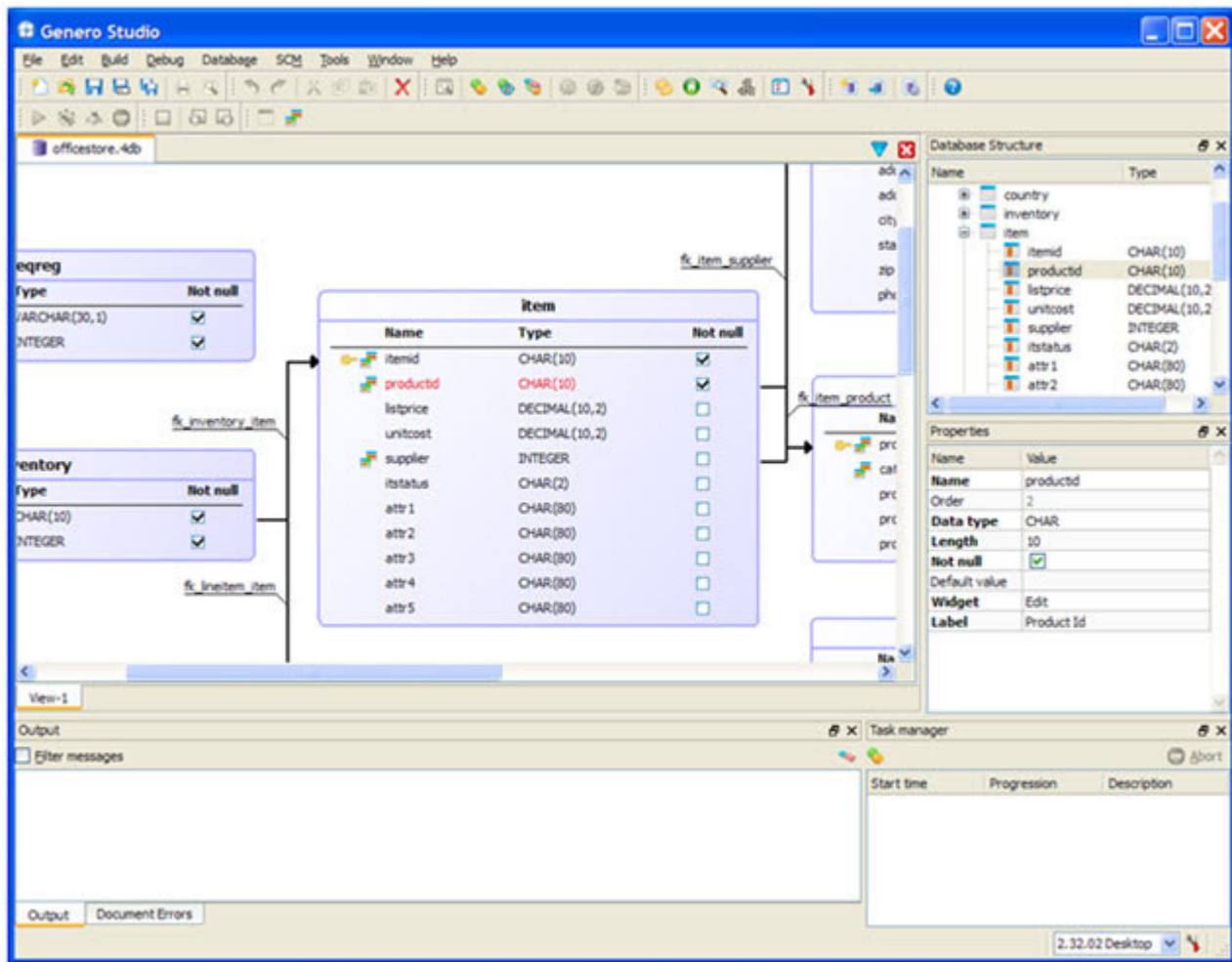


Figure 3-3. Meta-schema Manager diagram

The Meta-schema Manager diagram models the database design and constraints

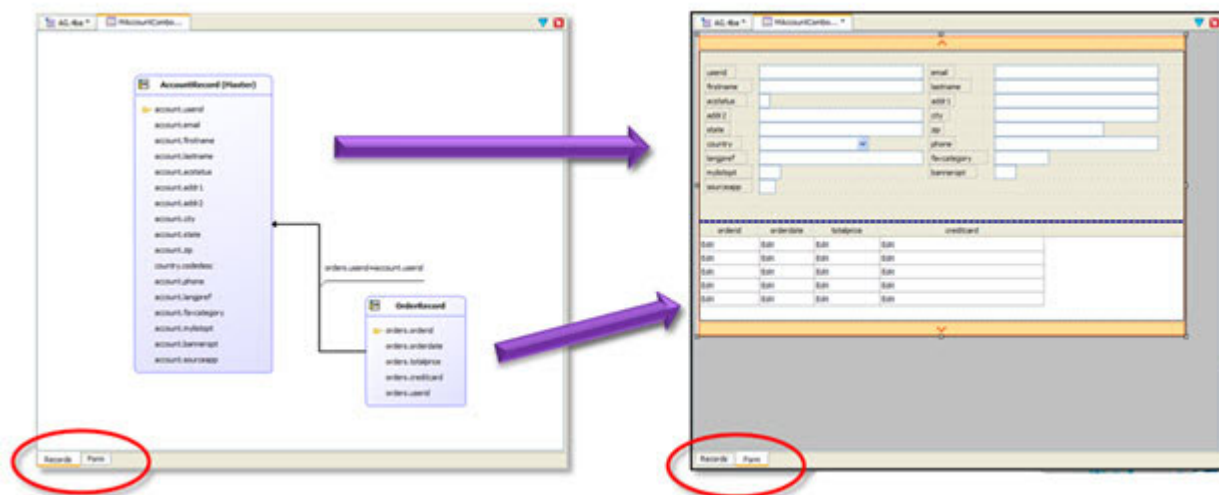


Figure 3-4. Form Designer diagram

The Form Designer diagram models an application's forms and screen records

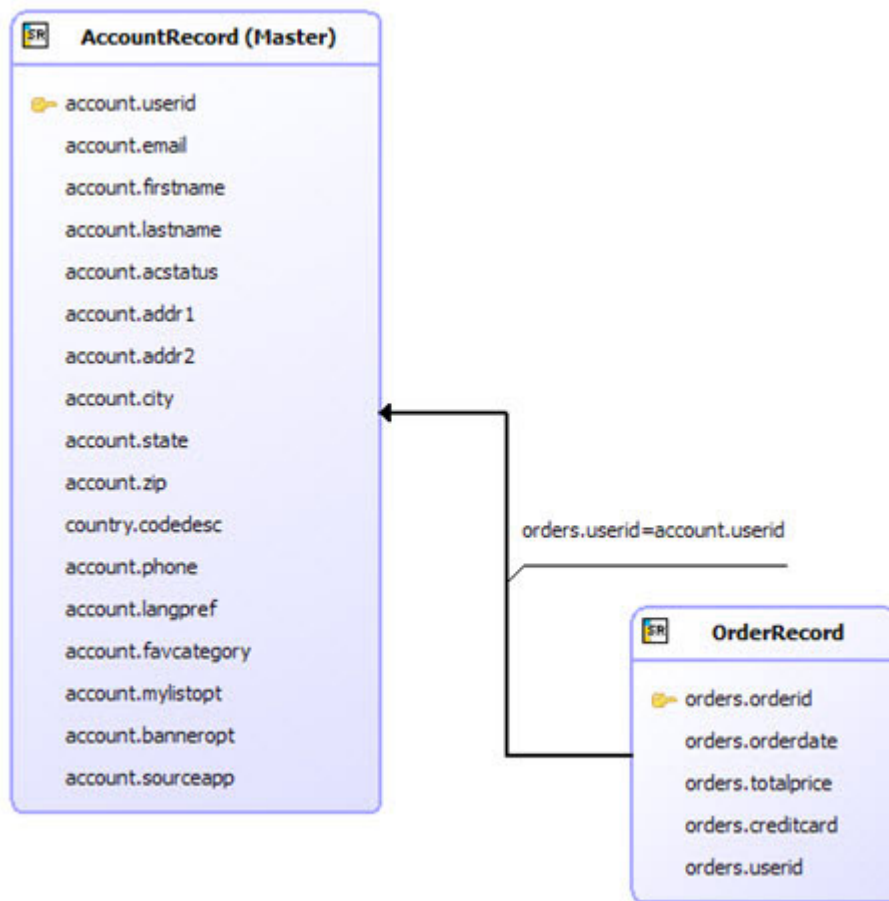


Figure 3-5. Business Record diagram

The Business Record diagram serves multiple purposes:

- It models the queries and constraints associated with the screen records for forms.
- It identifies the tables, columns and queries/constraints that specify the report data required to generate a .rdd (Report Data definition file). The .rdd file is used in conjunction with a Genero report definition (.4rp) file to automatically generate the reports.

Application Generation TCL templates

A standard template set consisting of files written in TCL language, and a configuration file used for the generation of programs is provided:

- **Database Applications** - the new default template **dbApp**, enhances the behavior of applications that interact with a Genero form, and implements the latest 2.4x features. Using the DIALOG instruction, sub-dialog statements that display data, query a database, and edit single records and record lists can execute in parallel, allowing the application to handle different parts of a form simultaneously.
- **2.3x Compatibility Multiple Dialog** - this template provides the same application generation functionality as earlier versions of Studio.

If you are creating a new program, we recommend that you use the Database Applications (dbApp) template. You can select the default template from the Studio main menu, Tools>>Preferences, Application Generator.

You can modify the template sets (an advanced technique) to meet your business requirements. This requires basic knowledge of Genero BDL, the TCL language (or another template language), and the structure of Application Generator programs and modules. See Customization for additional information.

BAM Tools

Genero Studio provides the following tools for application modeling:

Data Consolidation - the components of the models are stored in a single XML file, which consolidates all the inputs into one package. This file is used when processing and generating the application code. The file could also be used to provide input to create the application models.

Code Generation - processing and generating the code is controlled by TCL templates, which you can create or modify using the TCL Editor. A default TCL template is provided. Power users can create their own TCL templates to process and generate the code. The default template is specific to 4GL, but there is nothing in Studio BAM code that is specific to 4GL modeling; you could use other tools if desired.

How Code Generation Works

When you have created the Business Application Model diagram and its associated diagrams, and you select Build from the Application right-click menu, the following occurs:

- Producing an XML File - the input from the various BAM diagrams that you have created is gathered into a single XML file.
- Generating code from the XML file - The XML file and a code template are used to generate the application code (the default TCL template produces Genero .4gl files)
- Restoring custom code - if any custom code was created earlier, it is automatically restored in the newly generated application code

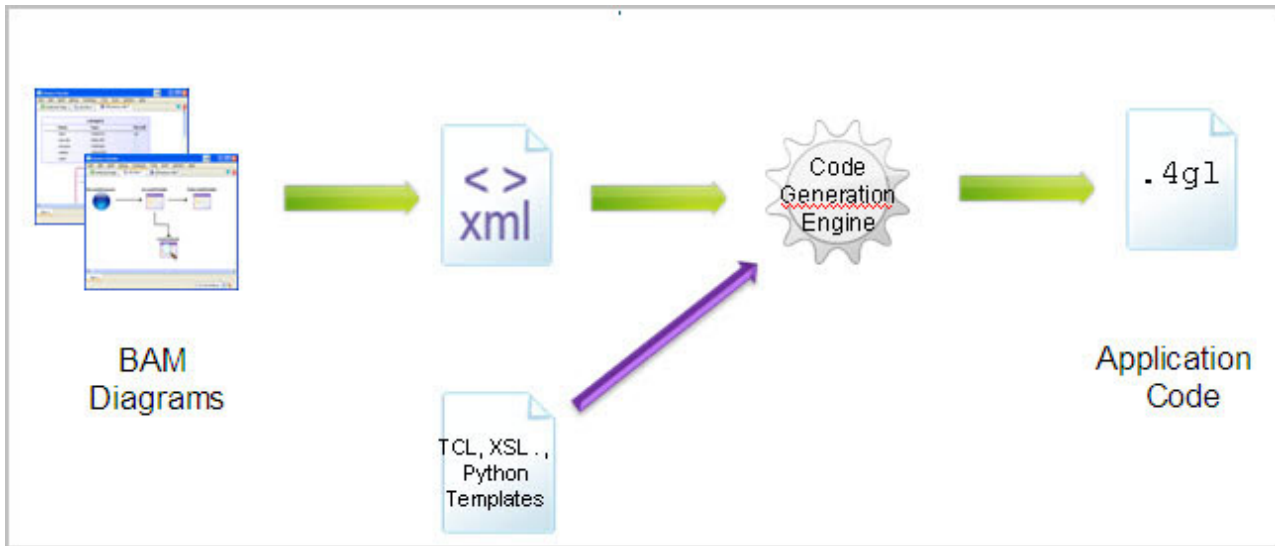


Figure 3-6. Code Generation flow

Application Generation Requirements

A Genero Studio schema file (.4db) must be added to a node in the project (in an Application Generator Project the default project structure has a node named **Databases**), or added to the DBSchema tab for the project.

Note: To execute an application that accesses a database, Genero Studio uses the Genero runtime system and the specific database client software, which must be properly configured. See Connections in the Genero BDL manual.

Sample databases and files

When you install Genero Studio 2.40, a directory of example files named "samples" is created (in the folder Documents and Settings/<username>/My Documents/My Genero Files/samples on Windows). An SQLite 3.7.5 database engine is included, with the *officestore* and *custdemo* databases preloaded.

Studio is already configured for the sample projects and databases. An **fglprofile** file is in the samples/DSConfig directory, and is used by the sample projects.

You may also create projects and applications using these sample files. See Your First Application.

Deploying the Generated Application

The Genero application that was created can be deployed as a stand-alone application, or it can be called from a menu that is part of a larger system. See Executable Application for a list of the files that must be included in a deployment.

Your First Application (the dbApp template)

This tutorial will take you through the steps to generate a basic Genero business application that allows the user to add/update/delete/query the rows in a relational database. The application will access the *officestore* sample database included in the Studio distribution. Code generation will be handled by the default Database Application (dbApp) template.

Topics:

- Step 1: Create a Managed Project
- Step 2: Add the Database Schema (.4db) to your Project
- Step 3: Create the Business Application Diagram (.4ba)
- Step 4: Create the Program and Form Definitions
- Step 5: Define Properties for the Program
- Step 6: Define the Properties for the Form
- Step 7: Build and Execute the New Program

Steps 8-10 guide you in creating a master-detail application.

- Step 8: Add a Detail Table to the Form
- Step 9: Define Properties for the Form
- Step 10: Set the Relationship Between the Tables

See also: Business Application Modeling, Program Flow: Opening Zooms and Additional Forms

Step 1: Create a Managed Project

- After starting Studio, select **Tools>>Preferences** from the main menu to display the choices you can select for Studio. Select **Application Generator** from the list to display its preferences.

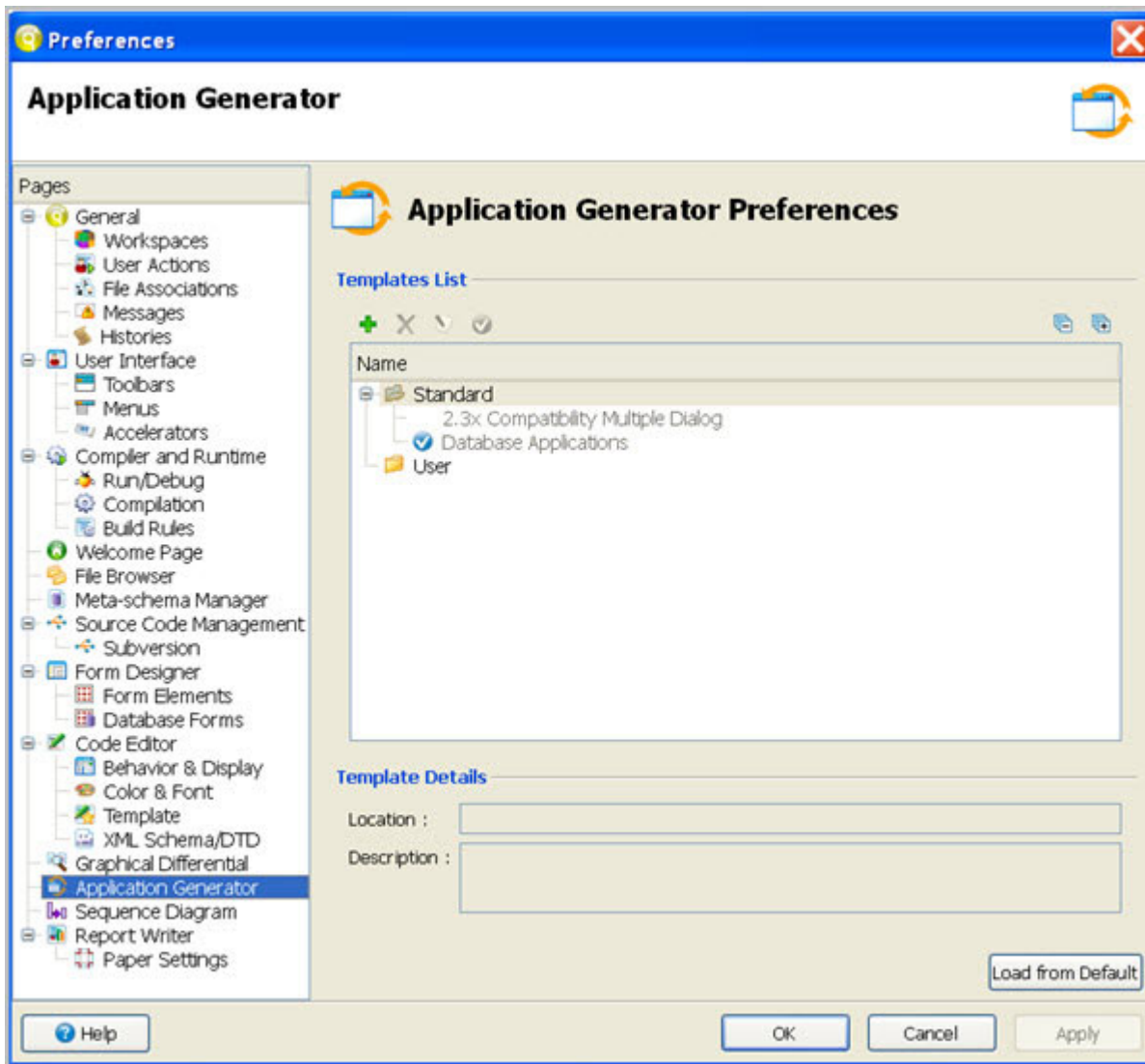


Figure 3-7. Application Generator Preferences

In the Templates list section, the **default Database Applications** choice should be checked. This template contains the latest application generation features (version 2.40).

Click the **OK button** to return to the Studio main menu.

- Use the Studio main menu option **File>>New, Design, Managed Project (.4pw)** to create a project for your application. Managed projects automatically contain the additional build rules for creating the generated programs.

The project structure has the following pre-defined nodes, to contain the files for your project:

- **Application** - for the program and form files, and any additional source code files
- **Databases** - for the Genero database schema files
- **Library** - for any additional resource files
- **Right-click on the Application node** and select **Advanced Properties** from the menu. You will see that the dependencies for the Databases and Library nodes have been set, insuring that any files they contain will be included in your application.

Click **Environment Variables** in the Pages list, and add a User variable FGLPROFILE by clicking on the green cross. The fglprofile file contains the configuration information to access the sample databases. This file is located in the Documents and Settings/<username>/My Documents/My Genero Files/samples/DSConfig directory:

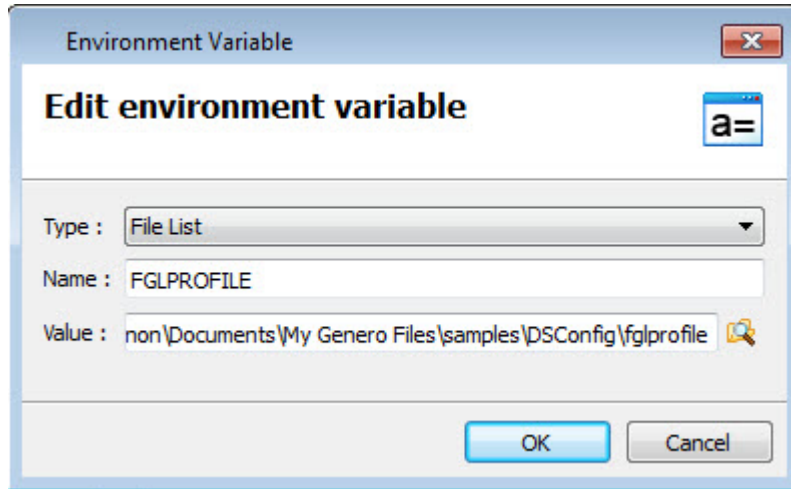


Figure 3-8. Edit environment variable dialog

- Choose **File>>Save All** from the Studio main menu. Save the project (.4pw) to a folder in your file system.

Step 2: Add the Database Schema to your Project

Your application will access the sample officestore database. The Genero Studio schema file for this database, officestore.4db, is located at: Documents and Settings/<username>/My Documents/My Genero Files/samples/databases/officestore/sqlite/officestore.4db.

This .4db file contains the information about the tables, columns, and relations of the relational database that is needed by your application.

- **Right-click the Database node** in your project structure, and select **Add Files**. Provide the location of officestore.4db and click the **Open** button to add the file to your project.
- Use **File>>Save All** to save the changes to your project.

Note: If you want to use a different database to which you have access, you must create a Studio database schema for it, and add that schema to the project. See “Genero Studio Meta-schema Manager” on page 7-1.

Step 3: Create the Business Application Diagram (.4ba)

- Use the Studio main menu option **File>>New, Design, Application Modeling, Business Application Diagram (.4ba)** to create a blank diagram.
- Right-click on the blank diagram and select **New Program**. This creates a Program icon on the diagram.
- Right-click on the blank diagram and select **New CRUD Form**. This create a Form icon on the diagram.
- Right-click the Program icon, and select **New Relation**. Then, click and drag from the Program icon to the Form icon to create an arrow link for the relation.

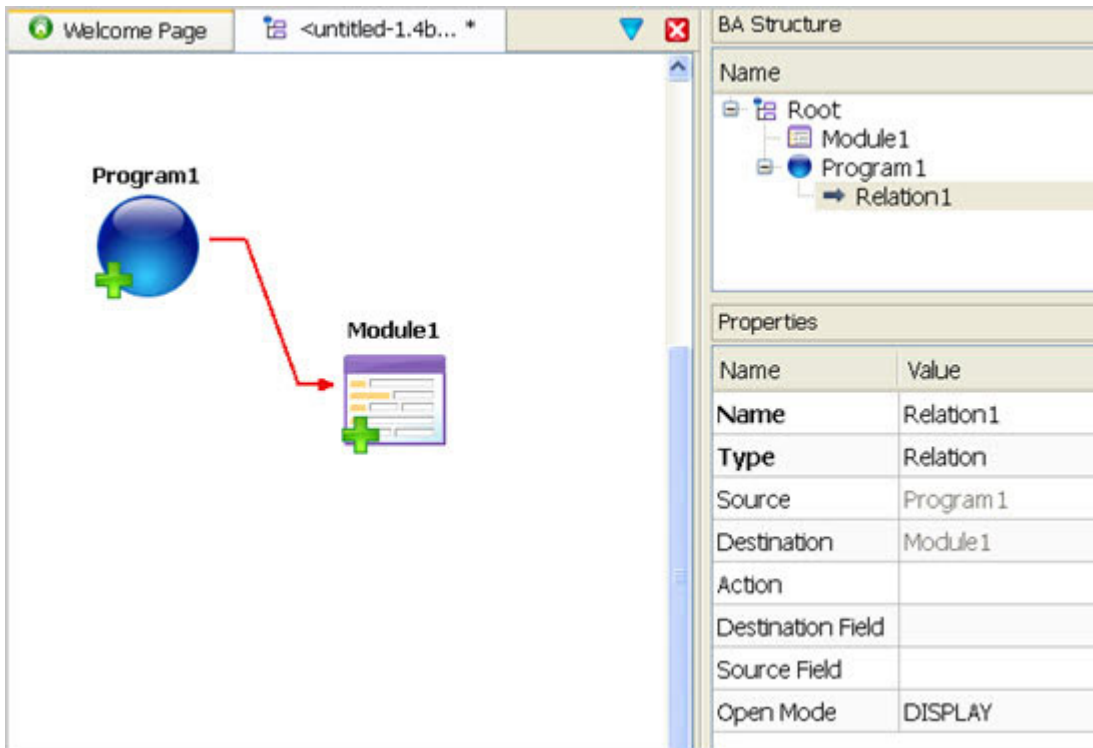


Figure 3-9. Defining a relationship

- Use **File>> Save As** to save the untitled Business Application Diagram to your file structure and to the Application node in the project.

Step 4: Create the Program and Form Definitions

- Right-click on the **Program** icon and select **Implement Program**. When the Save As dialog appears, assign a name for the program file, and save it under the Application node. This file will be used to generate the source code for the main function of the program. The file will be *<program-name>.4prg*.
- Right-click on the **Form** icon and select **Implement Form from Database** to generate a form for the database table that you wish to access. Select the officestore database, the account table and the first ten fields of the table from the Table List. Click the right arrow to transfer this selection to the Selected Fields list, which lists the desired fields for your form:

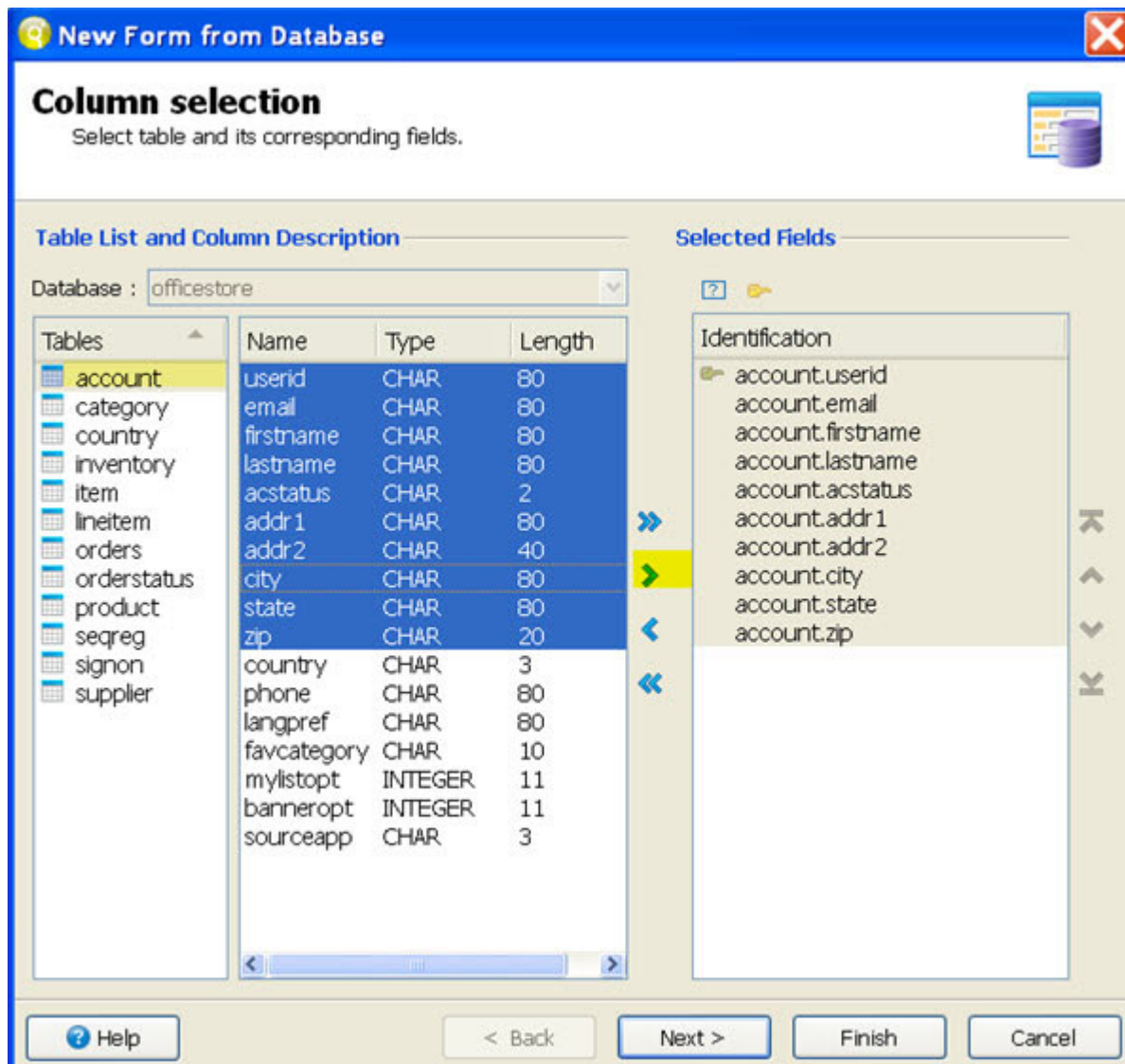


Figure 3-10. New Form from Database wizard Column selection

- Select Finish and save the form (.4f~~dm~~) to the project structure under the Application node.
- Select **File>>Save All** to save the edited project and its contents.

Now you have two documents in your Central work area, the Application Diagram (.4ba) and the Form Definition (.4f~~dm~~). The Projects Structure view has the added files:

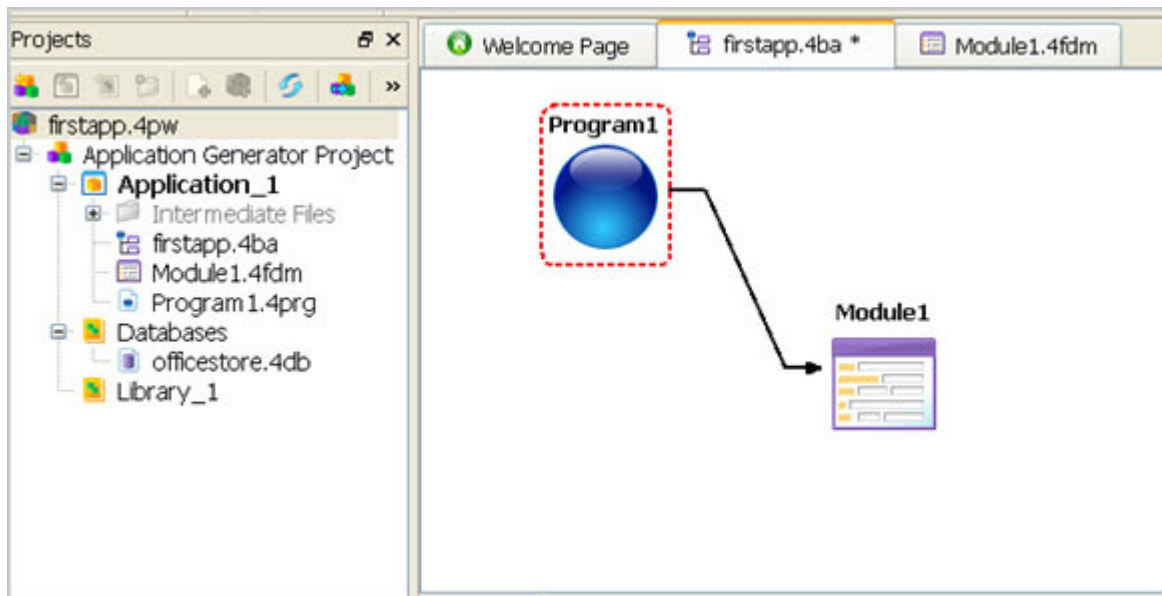


Figure 3-11. Projects Structure view

The Intermediate files are generated by Studio, using the definitions for the Program and Form. For generated applications that use the default Database Application (dbApp) template, BDL source code files (.4gl) are created and are used in deployment.

Step 5: Define Properties for the Program

The default properties are used for this application, so no changes are required. However, the Program properties can be accessed and changed as shown in this step.

Open the Application Diagram (.4ba). Click the relation arrow to display its properties.

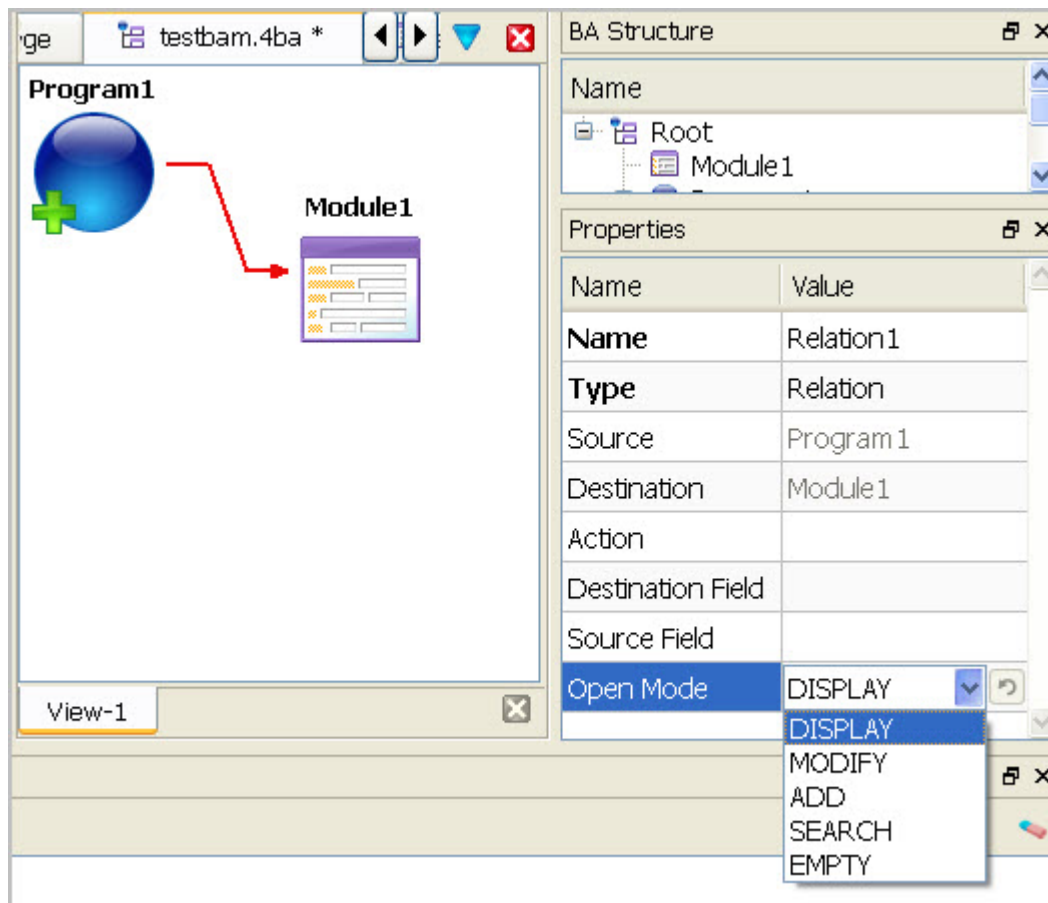


Figure 3-12. Business Application Diagram

The **Open Mode** property specifies the action mode that is currently active when the user first starts the program. Icons on the form's default toolbar allow the user to switch the mode.

- The default mode, **DISPLAY**, retrieves the rows from the database and displays them as a record on the form, one at a time. Icons on the default form toolbar allow the user to scroll through the list of records.
- The **MODIFY** mode allows the user to modify the currently displayed record.
- The **ADD** mode allows the user to enter a new record.
- The **SEARCH** mode allows the user to enter criteria for a search of the database, displaying the records that match that criteria.
- The **EMPTY** mode displays an empty form. The user can click the search icon or the add icon on the form to change the mode.

Step 6: Define Properties for the Form

The default properties are used for this application, so no changes are required. However, the Form properties can be accessed and changed as shown in this step.

Open the Form file (.4fdm).

The screenshot shows a window titled 'Welcome Page' with two tabs: 'testbam.4ba' and 'account.4fdm'. The main area displays a form layout with the following fields:

User Id		E-mail	
First Name		Last Name	
Status		Address	
Address		City	
State		Zip code	

Figure 3-13. Form file

Business Record - Click the Records tab at the bottom of the Form window to display the screen record associated with the form. The Business Record contains the data definition, structure, and table relationships required to generate the application.

This record lists the columns in the account table that have corresponding fields on the form.

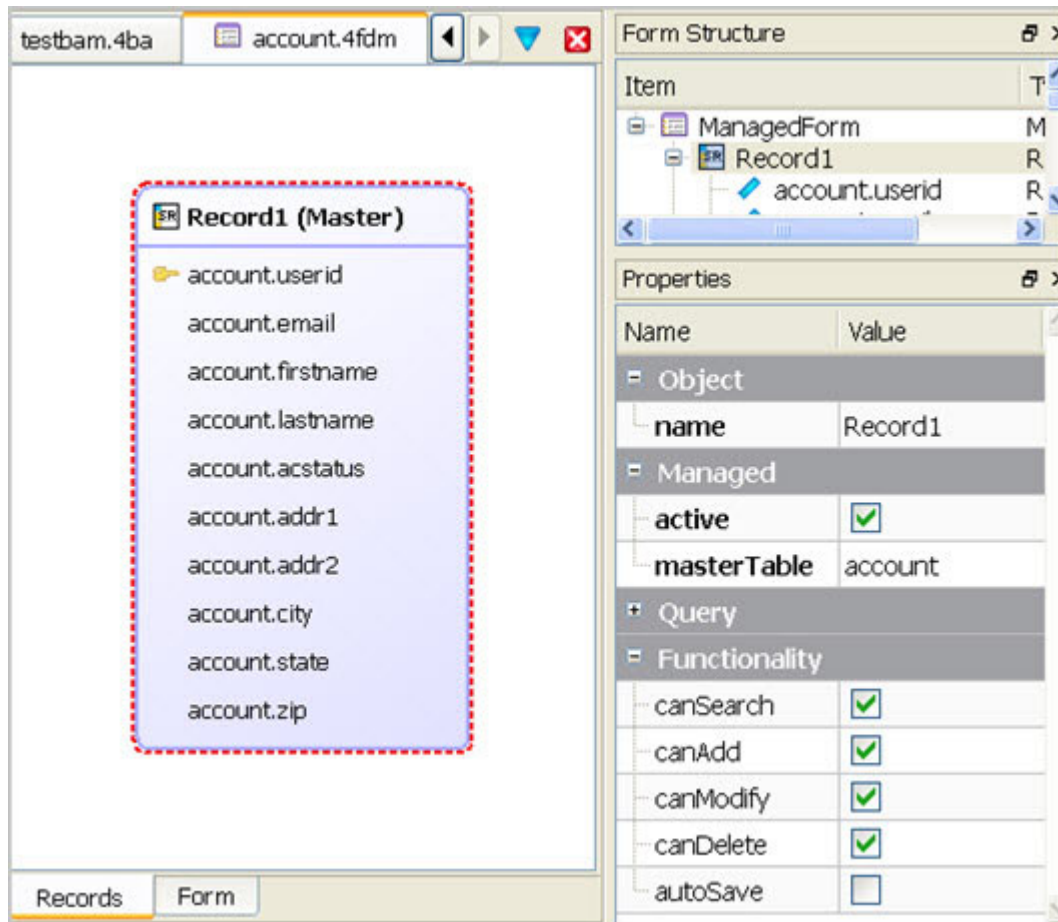


Figure 3-14. Business record

Properties - select the record to display the properties associated with it.

- The **active** property is checked, and the name of the database table is shown in the masterTable property.
- Based on the information in the database schema file, the field account.userid is identified as the **unique key**.
- The **Functionality** section specifies the actions on this table that are available to the users of this application form. Based on the default checked properties of this screen record, the user can search the database table for specific rows, add new rows, modify existing rows, and delete existing rows.

Step 7: Build and Execute the New Program

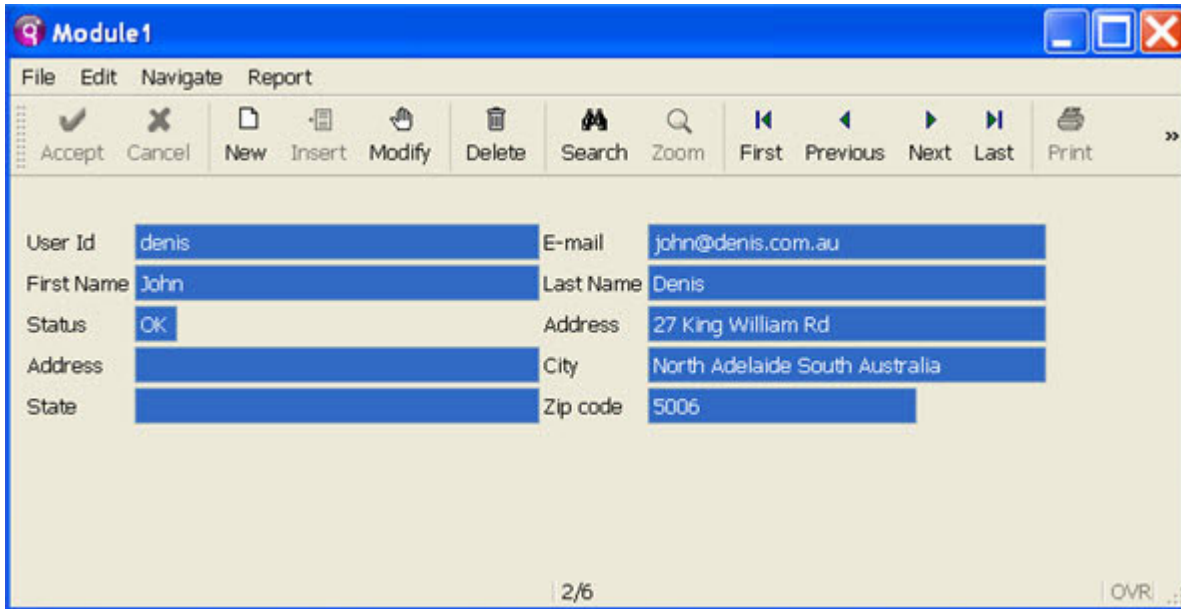
Building the Program

On the Business Application Diagram (.4ba) page, right-click the Program icon and select **Build Program** from the context menu. The results of the Build will be displayed in the Output View.

The generated and compiled files will be stored in the path specified in the Target Directory property of the project node.

Executing a Generated Application

To execute a generated application, right-click the Application icon in the Business Application Diagram (.4ba) page and select **Execute Program** from the context menu, or right-click the Application node in the project and select **Execute**.



The screenshot shows a window titled 'Module1' with a menu bar (File, Edit, Navigate, Report) and a toolbar with icons for Accept, Cancel, New, Insert, Modify, Delete, Search, Zoom, First, Previous, Next, Last, and Print. The main area contains a form with the following fields and values:

User Id	denis	E-mail	john@denis.com.au
First Name	John	Last Name	Denis
Status	OK	Address	27 King William Rd
Address		City	North Adelaide South Australia
State		Zip code	5006

At the bottom of the window, there is a status bar showing '2/6' and 'OVR'.

Figure 3-15. Your First Application running!

Toolbar icons (action views) for a program's default actions will be generated, depending on the properties selected for the module's screen records and the fields on the form.

The necessary business logic was created in the generated BDL files (.4gl) to implement the relevant actions.

See Actions and Action Views.

Step 8: Add a Detail Table to the Form

A Genero application can display a form that contains a master-detail relationship between two tables. The user can search for a row in the master table, and the corresponding rows in the detail table will also be displayed. The values in rows from both tables can be added/deleted/modified.

The form must contain fields from both tables and the table relationships must be set:

- In the form, enlarge the grid around the master table to make room for the fields from the other table.
- Select Data Control from the **Container** menu, and select the columns for the detail table. The container for these fields can be a table (to display multiple rows). Draw the container within the enlarged grid, under the original.
- Save the form (.4fdm).

On the Form tab

This form design contains fields from the master table, **account**, and a detail table, **orders**.

The screenshot shows a form design interface with a tabbed window. The active tab is 'custform.4fdm *'. The form layout includes the following fields:

- Store #
- Address
- City
- Zipcode
- Phone
- Store Name
- Address2
- State
- Contact

Below the fields is a table with the following structure:

Order #	Order Date	Store #	Shipping
Edit	Edit	Edit	Edit
Edit	Edit	Edit	Edit
Edit	Edit	Edit	Edit
Edit	Edit	Edit	Edit
Edit	Edit	Edit	Edit

Figure 3-16. Form tab

Step 9: Define Properties for the Form

On the Records tab

- Select the **master table record**, make sure **active** property is checked.
- Select and **identify a column as the unique key**, if not already identified, as indicated with the key icon.

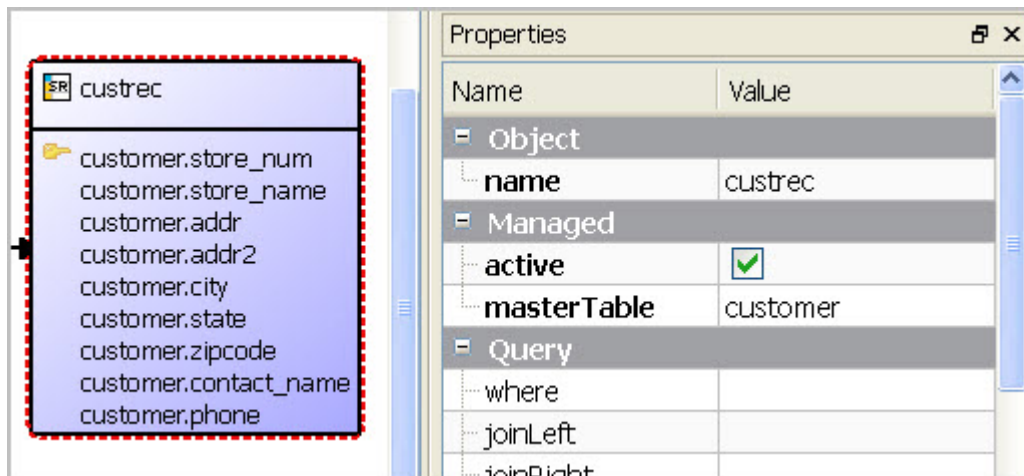


Figure 3-17. Master table record

- Do the same for the **detail table record** :

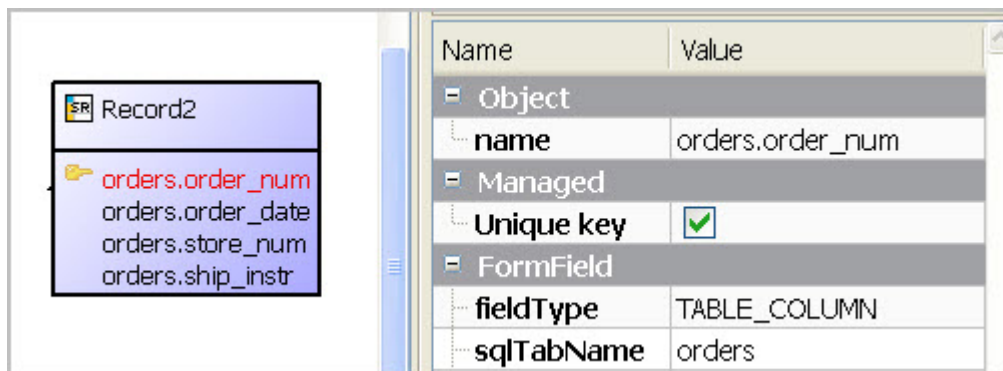


Figure 3-18. Detail table record

Step 10: Set the Relationship Between the Tables On the Records tab

- Right-click the foreign key field in the detail table record (**orders.userid**, in our example), and select **Add Relation To**. Drag the arrow to point to the primary key in the master table record (**account.userid**.)

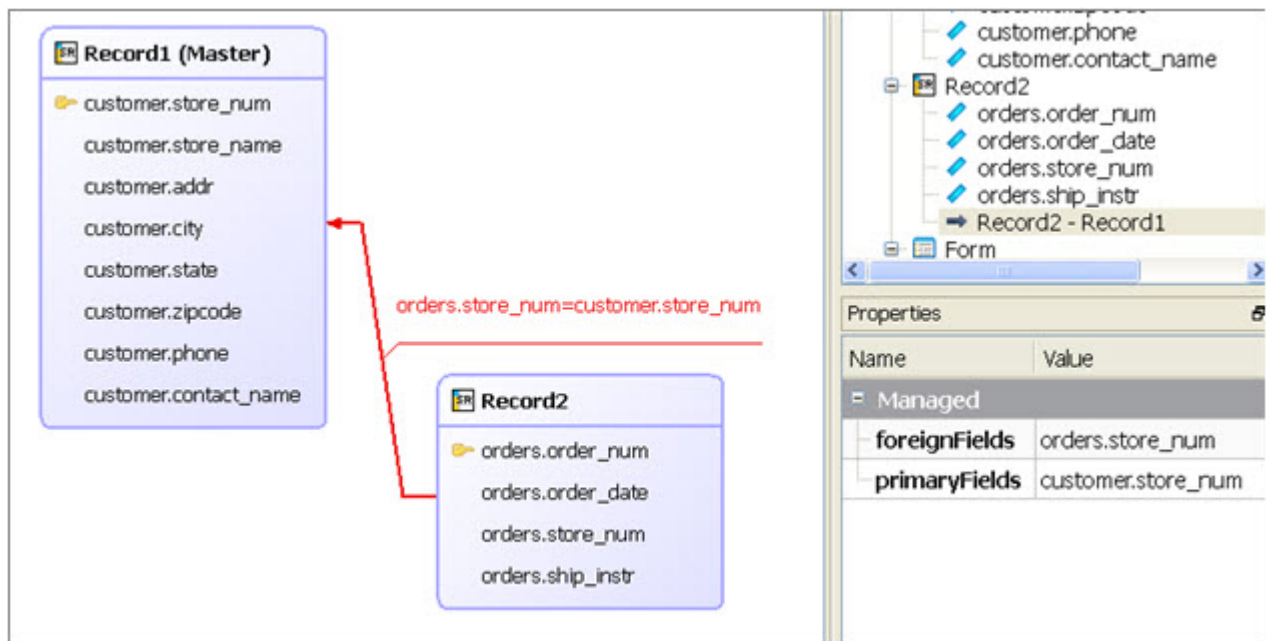


Figure 3-19. Master-detail relationship

Select **File>>Save All** to save the project and the modifications to its contents.

Continue by building and executing the program.

Order #	Order Date	Store #	Ship
5	04/04/2008	101	FEDEX
8	05/05/2008	101	FEDEX
9	05/05/2008	101	FEDEX

Figure 3-20. The application executes

Business Application Model Diagram

Topics:

- The Diagram
 - BAM Components
 - The Project Structure
 - Database Access
- The Program component and Menu
- The Form component and Menu
- Zoom and ReportData Objects
- The Relation component
- Additional Views
- Import Files from Your Project

See Also: Introduction to Business Application Modeling, Your First Application, Program Flow using Additional Forms

The Diagram

The Business Application Model diagram is designed to model the application flow and provide a high-level overview of the application to be generated.

Diagrams are used at all stages in the development. They serve as inputs for code generation, and are considered source code for the application. They are always up to date.

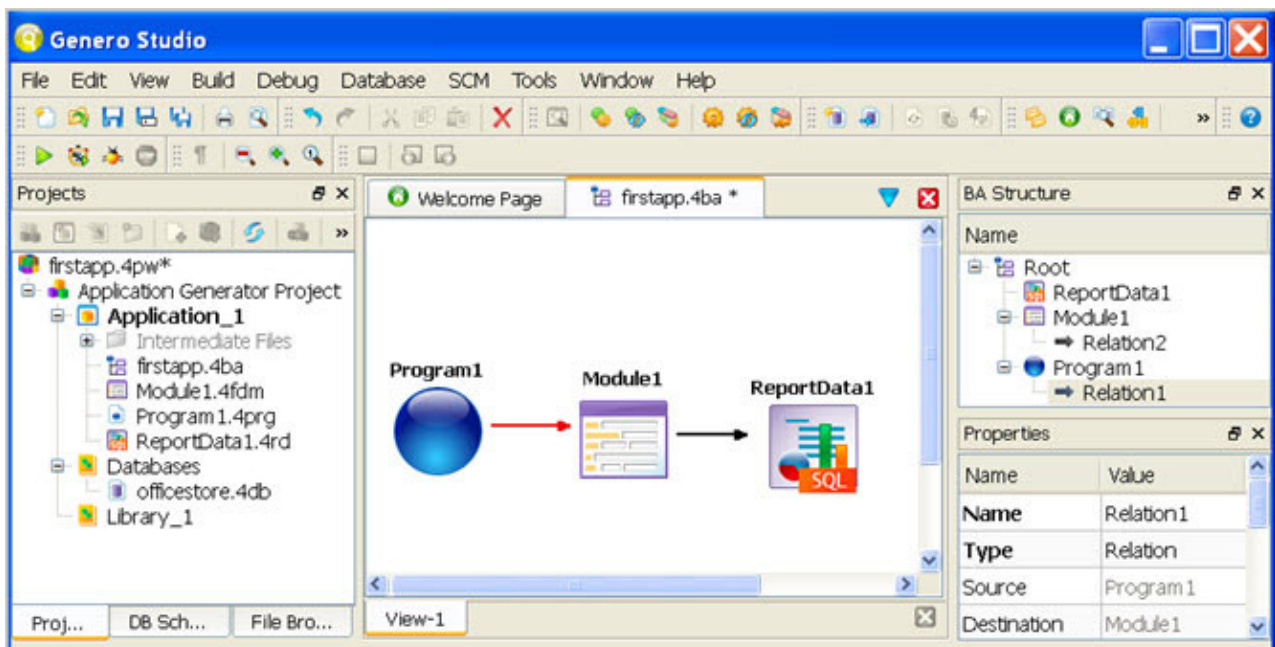


Figure 3-21. Business Application Model diagram

This example program is included with Genero Studio. From the **Welcome Page**, select **Tutorials and Samples** and the **OfficeStore** project.

Create a new BAM diagram by selecting **File>>New, Design, Application Modeling, Business Application Diagram**.

Right-click in the background of the diagram to display a context menu allowing you to add components to the Business Application Model (BAM) diagram.

BAM Components

By default, a Business Application Diagram for generating applications using the default Business Application template (dbApp.tpl). Entries in the diagram are inputs for the application code generation. The diagram can contain the following components:

- Program (.4prg) - generates the main logic to drive the program. This file provides the "glue" combining the generated .4gl files into a Genero application. It generates a Genero .4gl file containing the MAIN function for the application.
- CRUD Form (.4fdm)- generates the user interface for the program (the managed form), and the logic for the Genero program interaction with the database tables referenced in the form. Each **CRUD Form Definition file** (extension **.4fdm**) generates a Genero **.4gl** file containing the program logic to access the tables contained in the form. For example, an action can call BDL statements that permit users to search a database for information or add/delete/modify database data. A Studio database schema file defines the tables and columns that make up the database.
- Relation arrows connect the components on the form. The relation is a node in the project structure, and has its own properties to specify the application actions for the referenced components.

See Your First Application for step-by-step instructions to create an application using Programs, Forms, and Relations.

- Zoom Form (.4fdz) - generates a popup user interface that displays a list of values for a specific field in the managed form, allowing the user to select a value from the list. See Program Flow: Opening Zoom Forms for step-by-step instructions to create and use zooms.
- Report Data (.4rd) - generates the logic to retrieve data from the database and run a report, based on the report layout in a separate report definition file and the Business Record. See Genero Report Writer Reports for step-by-step instructions to include a Genero Report Writer report in the application.

The Project Structure

When you create a Managed Project (**File>>New, Design, Managed Project**), nodes for the basic structure of the project are already defined, and automatically contain the additional build rules for creating the generated programs. As you add components to the diagram, you are prompted to save them and add them to the project. The Projects view displays a tree of the project nodes and files. The default Application Generator project contains the following nodes:

- Group node- parent folder
- Application node - folder for the component files, the form definition files, and the generated source code files.
- **Databases** node - folder for the database schemas for the databases the application will access
- Library node - folder for any additional files used by the application; form files are frequently stored in those folder, to make them available for other applications in the same project

Right-click a node in the project to display a menu of options, including adding and deleting nodes and files. See Genero Studio Project Manager for additional information about the use of projects.

Database Access

You must also have access set up for the database that the application will use. A Studio database schema must be created for the database.

Example databases custdemo and officestore are provided in the installation package, and have already been configured for use in Studio. Database schemas and an fglprofile file are in the **samples** directory.

The Program Component and Menu

The Program contains the instructions to generate the main function of the application.

Select the Program Component on the Application Diagram (.4ba) to see a set of properties in the Properties View:

- **Name** - name of component
- **Type** - type of component
- **File Name** - full path to the file

Right-click the Program icon in the diagram, to display a context menu of options:

- **Implement Program** - create the Program file (.4prg) and add it to the project structure of the application. This file generates the logic for the main function of the program. Until you create the Program file, the Program component on the diagram is simply an icon.
- **New Relation** - add a relationship to another object on the diagram. Click the icon and drag the arrow to the related object (usually a Form icon). The relation is shown as an arrow in the Structure view, and has properties that specify the actions between the program components.
- **Build/Execute Program** - generate/execute the source code for the program based on the related components in the diagram.
- **Hide Selected Items** - select objects that will still be present but hidden from view
- **Show Filter View** - used to filter the display of objects on the diagram
- **Delete** - delete the Program component

See Your First Application. for step-by-step instructions to create a program and generate an application.

The Form Component (.4fdm) and Menu

This component contains the information about the application form for the program, used when generating logic for the generated program's interaction with the database tables referenced in the form. CRUD Forms serve as the graphical interface for the application.

Select a Form Component on the Application Diagram (.4ba) to see a set of properties in the Properties View:

- **Name** - name of component
- **Type** - type of component
- **File Name** - full path to the file

Report Options

If your form has a valid relation to a report definition (.4rd), some actions are provided by default in the ToolBar to launch the report. These form component properties can be used to define if these report actions are available or not.

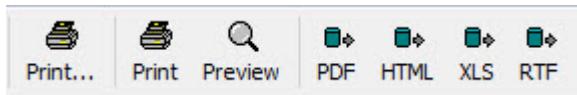


Figure 3-22. Default toolbar

- **quickPrint** - defines if print action is available
- **quickPreview** - defines if preview action is available
- **canExport** - defines if the PDF, HTML, XLS, and/or RTF print actions are available. If canExport is checked and quickPDF, quickHTML, quickXLS, or quickRTF are also checked, the action will be available.
- **quickPDF** - defines if export to PDF action is available
- **quickHTML** - defines if export to HTML action is available
- **quickXLS** - defines if export to XLS action is available
- **quickRTF** - defines if export to RTF action is available

Right-click the Form icon in the diagram to display a context menu of options:

- **Implement Form** - create a new blank managed form definition file (.4fdm) that you can design from scratch.
- **Implement Form from Database** - generate a managed form definition file (.4fdm)

After you create the managed form, you are prompted to save it in the project structure.

- **Open Form** - open an existing form definition file associated with the Module.
- **New Relation** - add a relationship to another object on the diagram.
- **Hide Selected Items** - select objects that will still be present but hidden from view
- **Show Filter View** - used to filter the display of objects on the diagram
- **Delete** - delete the Program component

See Creating a CRUD Form for additional information about the application interface.

See Your FirstApplication. for step-by-step instructions to create a program and generate an application.

Zoom and ReportData Objects

Each of these additional diagram objects has its own dedicated documentation page.

See Program Flow: Opening Zooms and Additional Forms for information about creating and using a pop-up form (Zooms) to assist the users of the generated application during data entry.

See Genero Report Writer Reports for information about creating and using ReportData objects to display Genero Report Writer reports in the generated application.

The Relation Component

Objects on the BAM diagram are linked by arrows that indicate their relationship.

Right-click an object (Program icon for example) and select **New Relation** from the context menu.

Then click the object and drag the arrow to the related object. The relation is shown as an arrow in the diagram and is listed in the BA Structure tree. Relationships have properties that specify the actions between the program components. The properties vary depending on the relationship.

See Your First Application. and Genero Report Writer Reports for additional information about specific relations and their properties.

Additional Views

Information about the diagram and its components is provided in views displayed in the Studio window, or through the “Windows Menu” on page 2-13. For example:

- The **BA Structure View** - this view contains a tree of the structure of the diagram and its components. Right-click an element of a tree to display the context menu, allowing you to delete the element, or to show the Filter View.
- The **Properties View** - select a component in the Structure View or the Diagram, and its properties will be displayed in the Properties View. Changing the values of the properties influences the generation of the application. See Your First Application for examples of the property settings.

See Your First Application for additional information about property settings for the generated application.

- The Output View and the Document Errors View - these views provide feedback during the creation and execution of the generated application.
- The **Filter View** - right-click the diagram or a component and select the **Show Filter View** option of the context menu to display the view. Checkboxes on the Filter View allow you to specify which components are to be displayed:

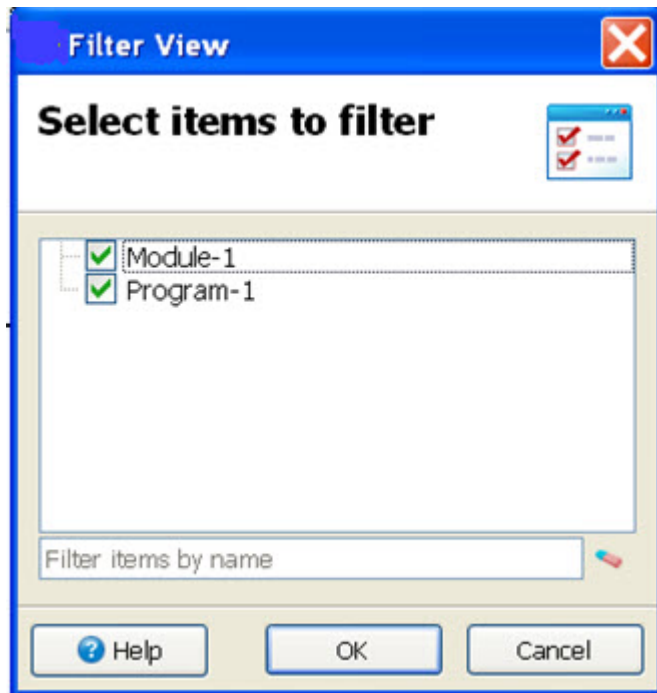


Figure 3-23. Filter View

Import Files into the Diagram (.4ba) from the Project

If there are files in the Project structure that are not in the Business Application Diagram, they can be imported.

Right-click on the background of the Diagram, and select **Import Project Files to BA Diagram**.

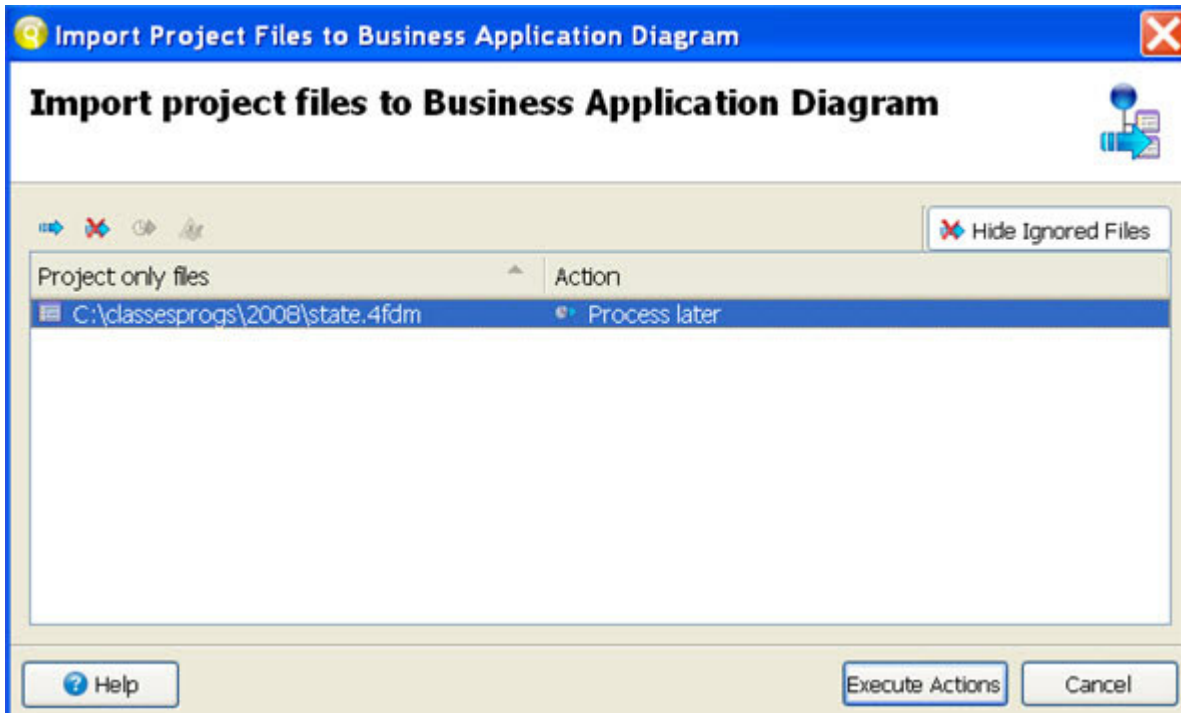


Figure 3-24. Import project files to Business Application Diagram dialog

Each file that can be imported will be listed in the dialog. Select a file, and use the icons at the top of the listing to choose whether to:

- **Ignore:** Do not add selected files to the Business Application Diagram
- **Process Later:** Decide later if the selected files should be added to the Business Application Diagram
- **Add:** Add selected files to Business Application Diagram
- **Resolve conflict:** resolve an existing conflict

Creating CRUD Forms

From the Business Application Model you can add a CRUD (Create, Read, Update, Delete) form. This type of form contains the logic to generate application code that will manage the form interface and its interaction with the referenced database tables.

Right-click the form icon on your Business Application Model diagram to:

- **Implement Form** - create a new blank managed form definition file (.4fdm) that you can design from scratch.
- **Implement Form from Database** - generate a managed form definition file (.4fdm). It is generally much easier to select this option, and make any adjustments that you wish to the form after it is generated.

Topics:

- Database Schema
- Implement Form from Database
- Form Components
- Records Tab

See also: Your First Application, Actions and Action Views

Database Schema

Prior to creating the form the Studio database schema file for the relational database you wish to access must be part of your Application Generator project. See Creating the Meta-schema file for additional information. If you are using one of the example databases provided in the samples directory distributed with Studio, the database schemas are already created and available.

The Studio meta schema file (.4db) describes the structure of a relational database. This database definition is used to provide the table names and column names and data types that will be referenced by fields on the managed form. Additional information about indexes, primary and foreign keys, and relationships between tables is used when the application source code is generated.

Implement Form from Database

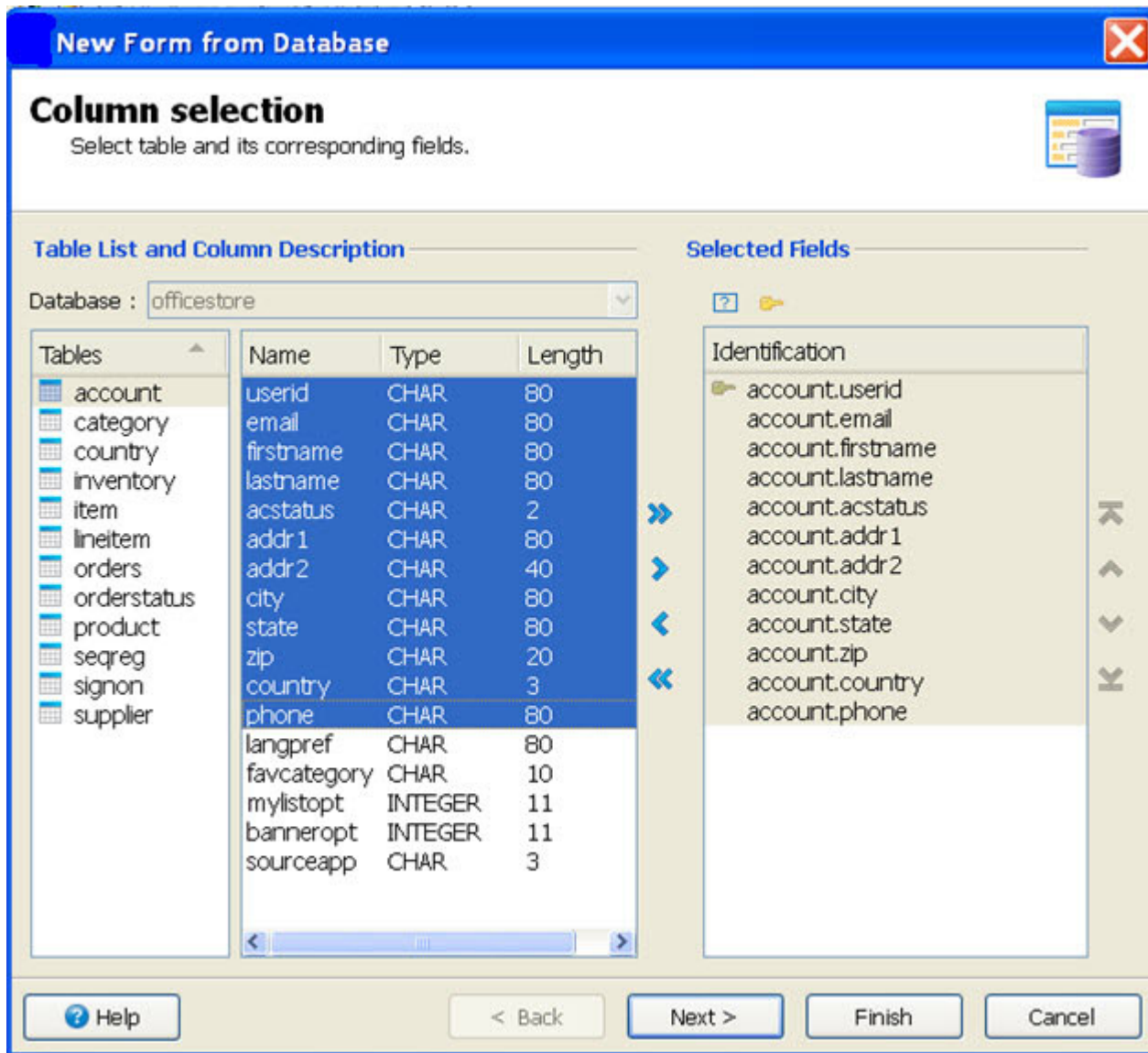


Figure 3-25. New Form from Database dialog

This **New Form From Database** dialog allows you to specify the database table columns that you wish to appear on the form. A form can contain multiple tables (see Master-Detail relationships).

- Click **Next** to continue, with Container Selection. The form must have a container, generally Grid to display a single database table row, and Table to display multiple rows at once.
- Click **Next** to continue, with Query Creation. If your form will contain more than one table, you can specify the type of join, the where clause, and order by clause of the SQL query that will retrieve the data for the form.
- Click **Next** to continue, The Dynamic Properties dialog allows you to change the default label and widget to be used that will be used for the form fields.
- Click **Finish**. The dialog will close, and the cursor will become a +. Click the empty form design window and drag to draw the container and its fields, in accordance with your choices in the previous dialogs.

Database Interface is a more detailed look at the creation of a form from a database table.

Form components

- **form fields** - (data handling widgets) define an area where the user can view and edit data; by default the widget is a simple Edit field, or a different type of widget can represent the underlying field (See Using Lookups and Combo Boxes.)
- **action views** - can be used to trigger an action defined in your program code. Toolbars and TopMenus are automatically generated for your application. See Actions and Actions Views.
- **decoration** - for display only (Images, Labels)

You can modify the appearance of a form item by changing the values for its properties in the Properties view. Select the form item to display its properties.

See Your First Application for a step-by-step guide to creating a form for your generated application.

Records Tab

This tab displays the screen record for the form, consisting of all the database field names that make up the fields on the form itself.

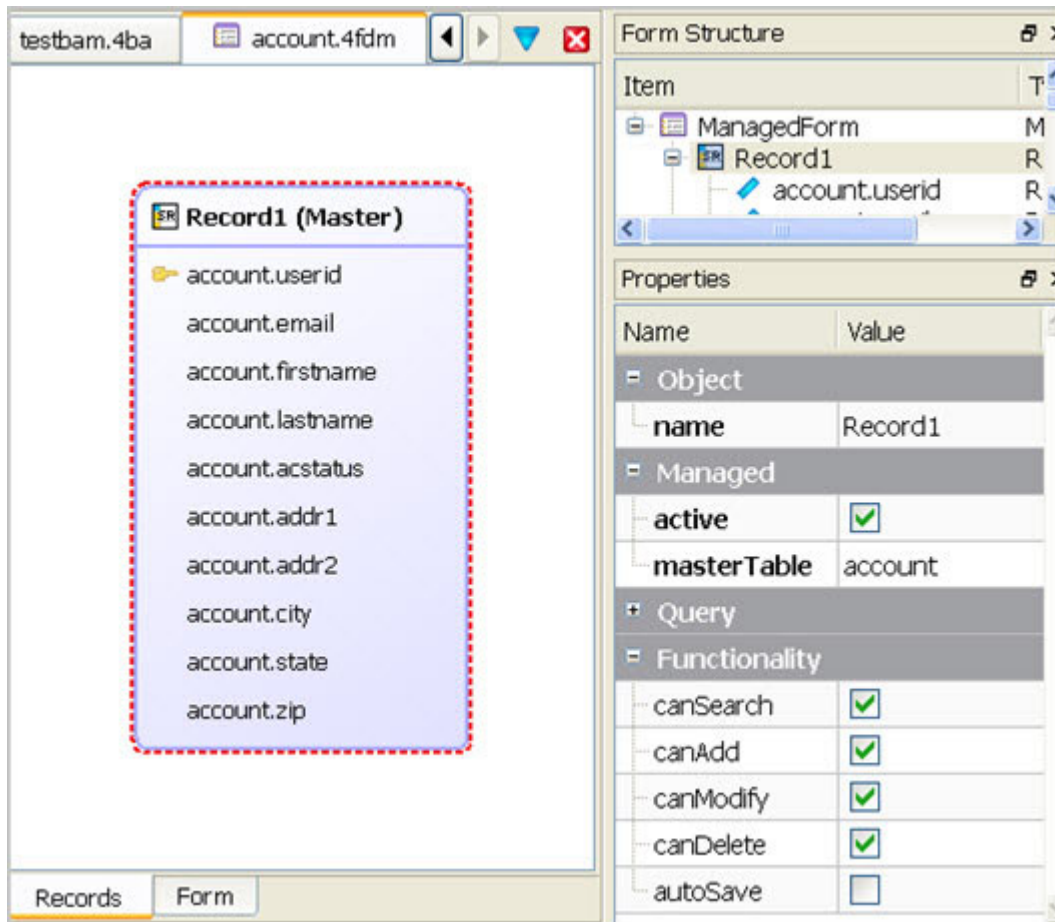


Figure 3-26. Screen record

A right-click context menu allows you to:

- **add field** - add an additional field to the form
- **edit** - edit the name of a new field that you have added, or the name of the record
- **edit query** - open the Edit Query dialog, allowing you to make changes
- **cut/copy/paste/delete** - make changes to the contents of the record.

Select the Record to display its properties in the Properties views. These properties allow you to specify the actions available to the user of the form. This information is used when the application is generated to create the logic that implements these actions.

See Your First Application for step-by step instructions to creating a form for your generated application.

Genero Report Writer reports

Topics:

- Modifying the Business Application Diagram
 - Add the ReportData module
 - Create the Business Record
 - Generate the .rdd File

- Creating the Report Definition file (.4rp)
- Rebuilding the Application

See Also: Your First Application

Modifying the Business Application Diagram

You can add a Genero Report Writer report to your generated application.

Add the ReportData module

- Open the Business Application Model (BAM) diagram for the application that will contain the added report.
- Add a new ReportData module to your BAM diagram, and link it to the Form containing the screen record that lists the fields from the database table to be accessed by the report by right-clicking on the ReportData module and selecting Add Relation from the context menu.
- Select the link to display its properties, and change the Type property to ReportRelation.
- In the property Report file, enter a name for a Genero Report Designer report definition that you will create (in the example, rep1.4rp). This file does not need to exist yet, as the next steps will generate the .rdd (data definition) file that is required by the Report Designer. Save the .4ba file.

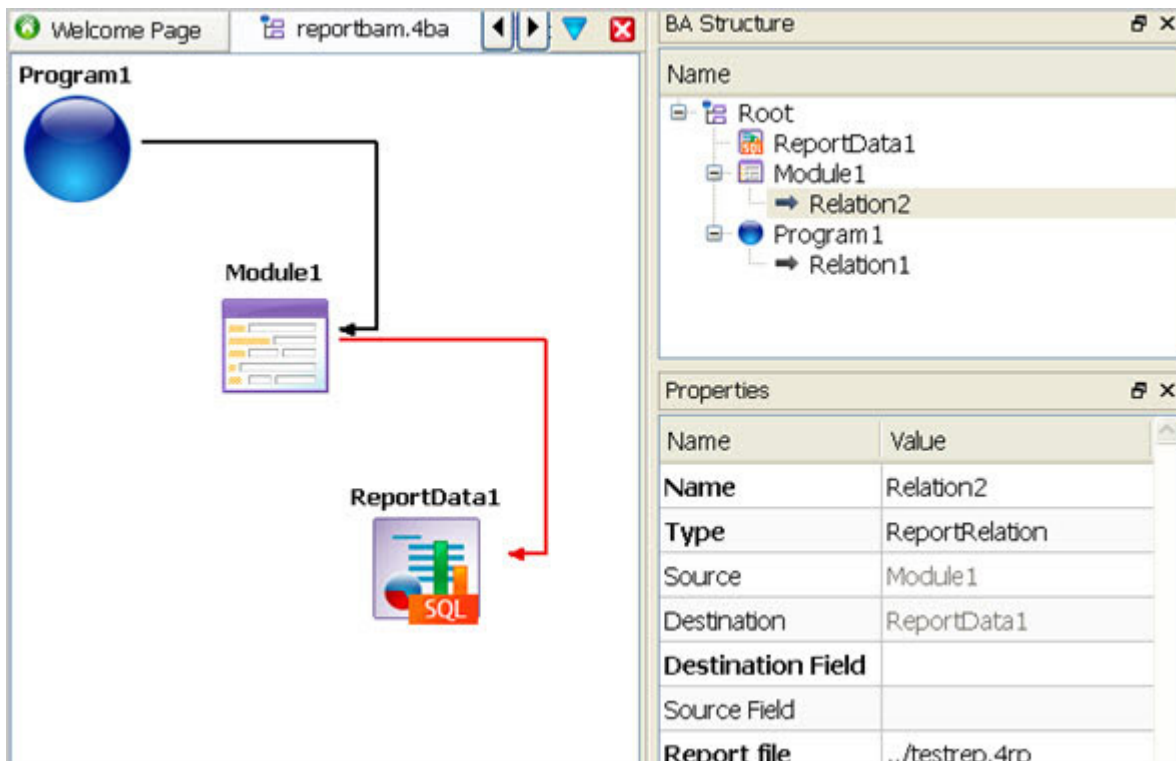


Figure 3-27. Example

Create the Business Record

The Business Record contains the data definition, structure, and table relationships required to generate the application.

- Right-click on the ReportData module and select New Business Record. When the blank ReportData area displays, right-click and select Add Record from the context menu. An empty record displays.
- Click the DB Schema tab to in the Projects view to display the schema for the database accessed by the application. Select the table for the report, and drag it to the central work area and drop it onto the empty Business Record.

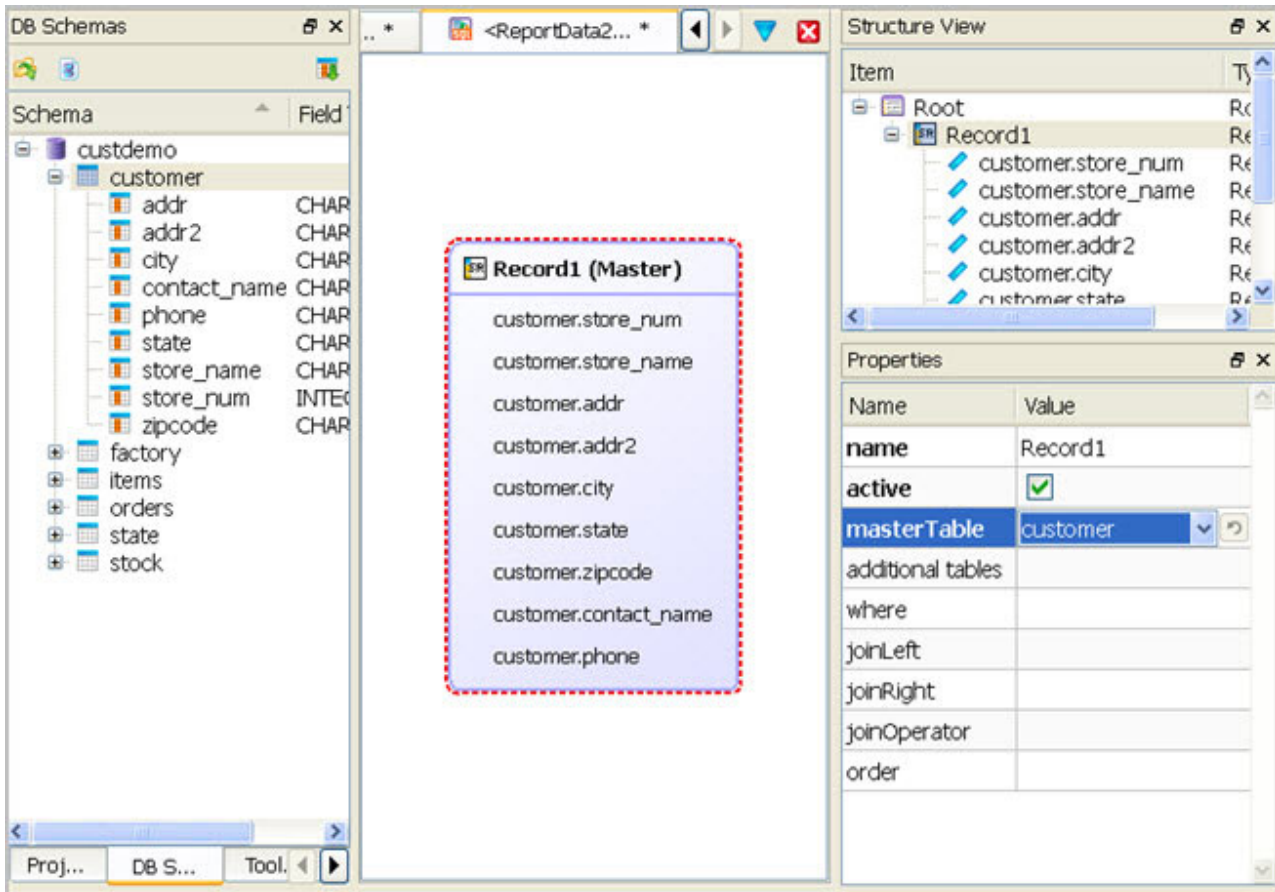


Figure 3-28. Populated Business Record

- Right click on any field in the Business Record that you do not want to be available for the report definition and select Delete from the context menu.
- Select the business record to display its properties. Make sure that the active property is checked and that the masterTable property contains the database table name.
- Select the field in the business record that represents the primary (unique) key in the database table, and make sure that the unique key property is checked.

Select **File>>Save All** to save the changes to the Business Application Model diagram.

Generate the .rdd File

In the Projects view, right-click the Application Node and select **Build** to build the program. An .rdd file will be generated for the database fields listed in the ReportData Business Record. This file will be used when you create the report definition.

Creating the Genero Report Designer report definition (.4rp)

- Return to the Studio main menu, and select **File>>New, Reports** and select the type of report you wish to create.
- When the empty report definition displays in the central work area, **select the DataView tab** from the Project View.

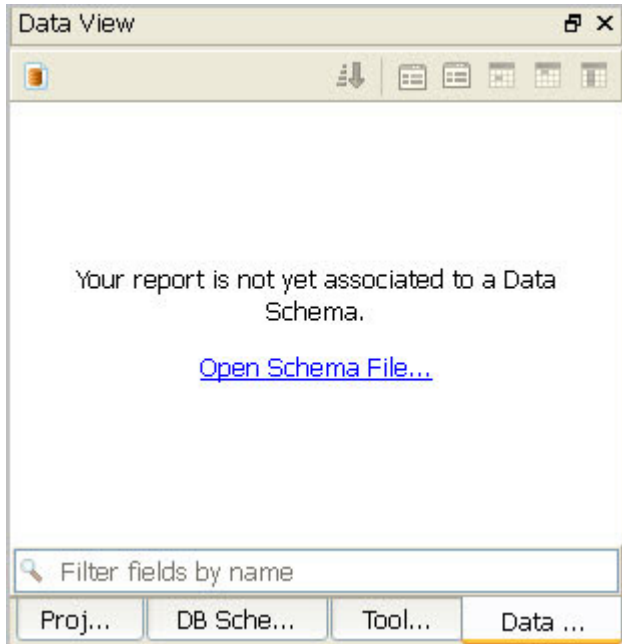


Figure 3-29. Data View

Click **Open Schema File** and add the .rdd file that was generated earlier. At this point, you can continue to define your report. See *Designing a Report* for additional information on the report design file.

- When the Report definition is completed save the file in the Project, using the same name that you listed as the link property (in our example, firstrep.4rp).

Rebuilding the application

Right-click on the application node in the Project view, and select **Build** from the context menu.

When the application is executed, report options will appear on the toolbar and menu, allowing the user to choose to run the report.

Lookup fields and ComboBoxes

Topics:

- Using Lookup Fields
- Populating a ComboBox Dynamically

Using Lookup Fields

A lookup field contains a value that is retrieved from a reference table instead of being input by a user. Lookup fields can be defined in the Form Definition file (.4fdm) used with your generated program. There must be a join relationship between the master table and the lookup table.

In this example, the form is primarily for the **account** table (master table), but it also displays the corresponding country name from the **country** table (lookup/reference table) when a customer's account record is displayed or input:

Figure 3-30. Form showing lookup field

1. Business Application Model

The first step is to set up the Business Application Model and the Program and Form icons as described in Your First Application.

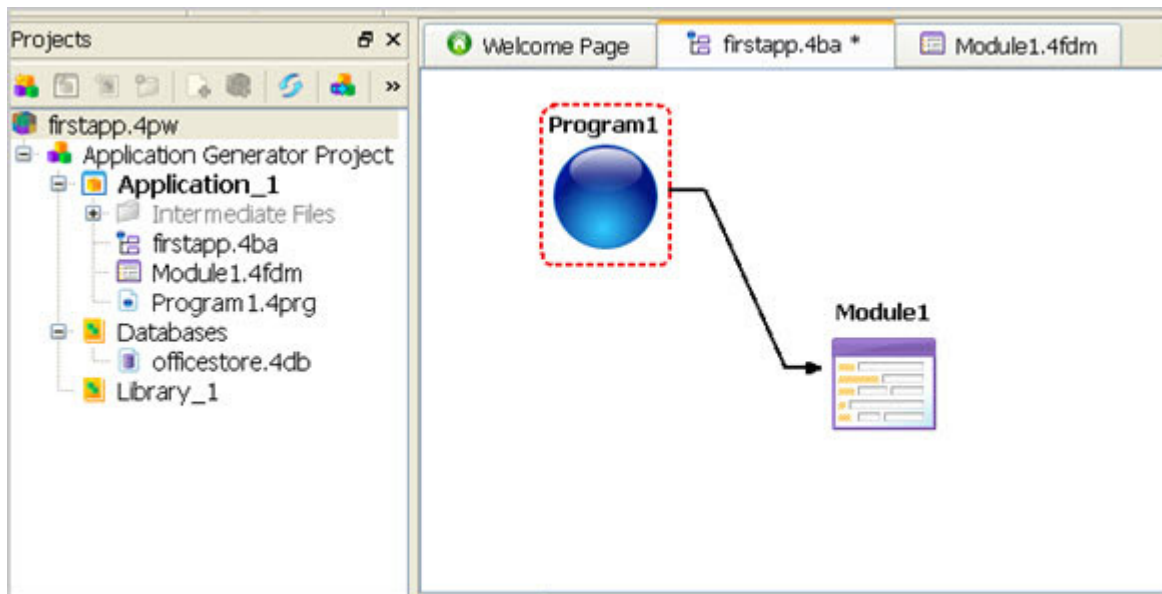


Figure 3-31. Your First Application

2. Modify Form for the Application

Modify the form:

- **Right-click** the Form icon to implement or open a form. Include the fields from the main table, as well as the lookup field from the reference table.
- The widget for the lookup field can be converted to a ffLabel widget if you do not wish the user to have access to that field.

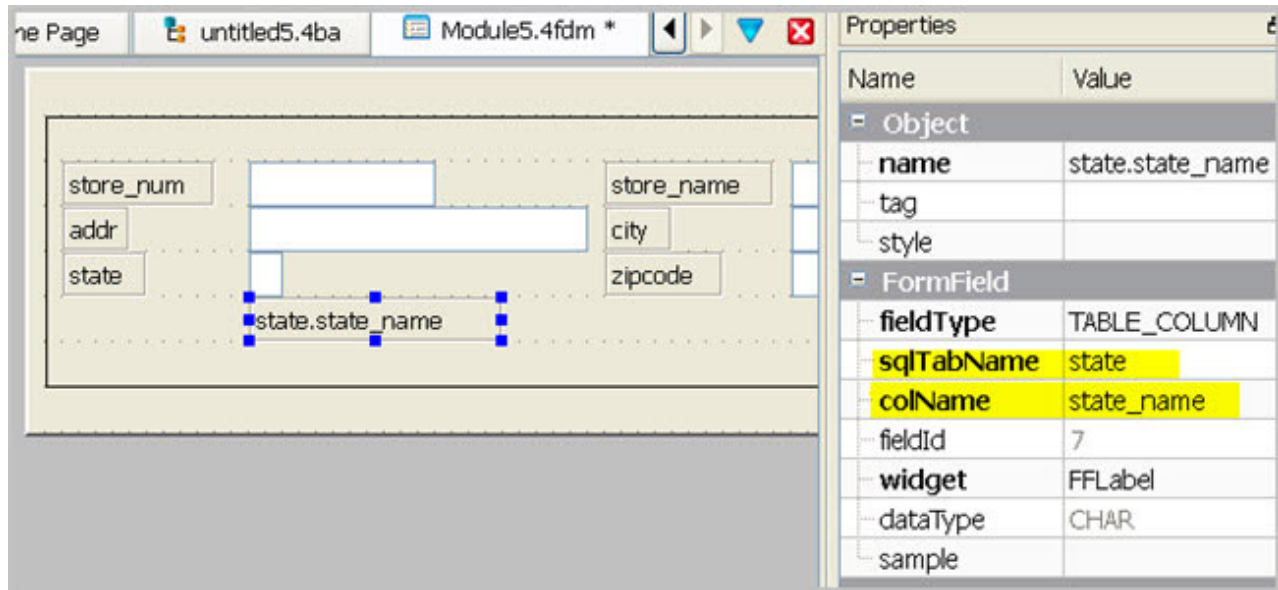


Figure 3-32. FFLabel widget

On the Records tab, the record shows the columns from both the account and country tables.

Select the **country.codedesc** column to see its properties. Set the **lookup** property to a unique name, for example lookupCountry. The generator will use this name in the function call that triggers the lookup when the user modifies the field data. If the value is found in the reference table, it is used to update the master table (account.country). If it is not found in the reference table, the original values on the form remain intact.

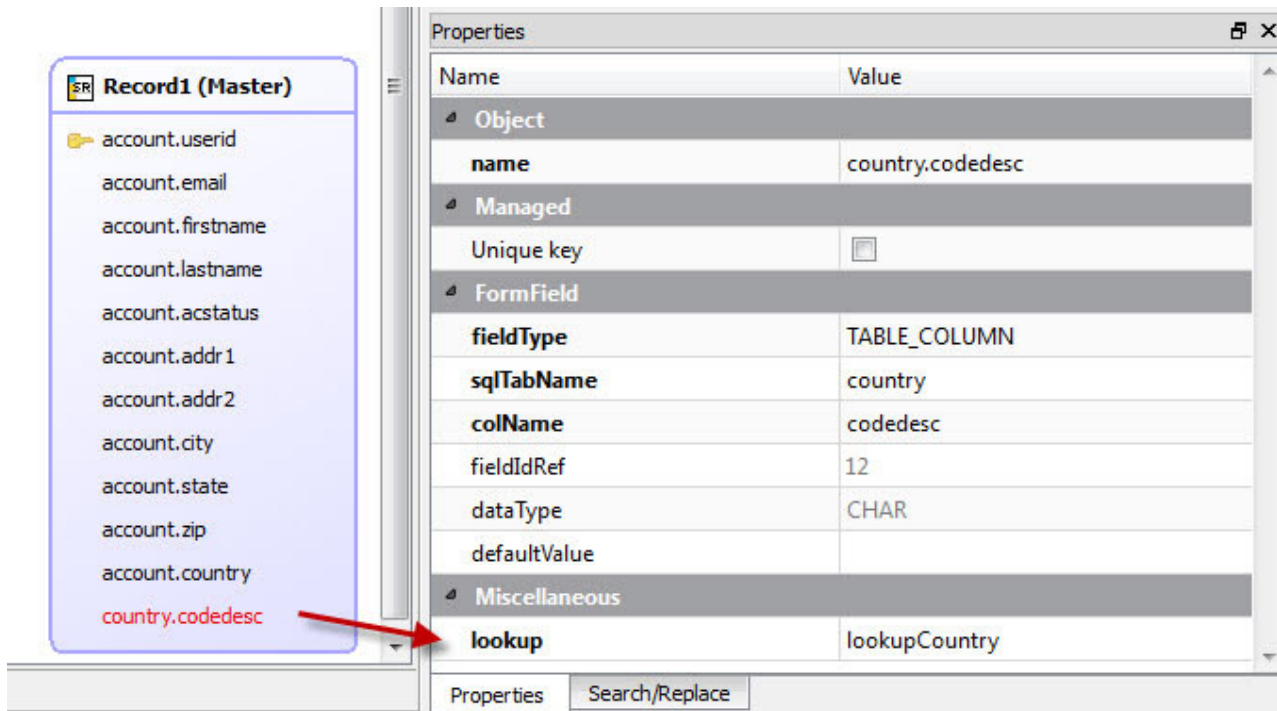


Figure 3-33. lookup Property

Note: it is not necessary to set the lookup property if your lookup field is an FFLabel and the user cannot input into this field because the lookup functionality will automatically be triggered. The join between the tables is enough to display the lookup data.

Composite Lookup

If the lookup requires two fields and both fields are editable, the lookup is resolved when the two values are provided by the user. To implement a composite lookup, set the the lookup property of both fields to the same value.

3. Define the query condition

In the Records tab of the form, select the Record (or select it in the Structure View); **Right-click** and select **Edit Query** to confirm the join between the master and detail tables (country.code = account.country in our example):

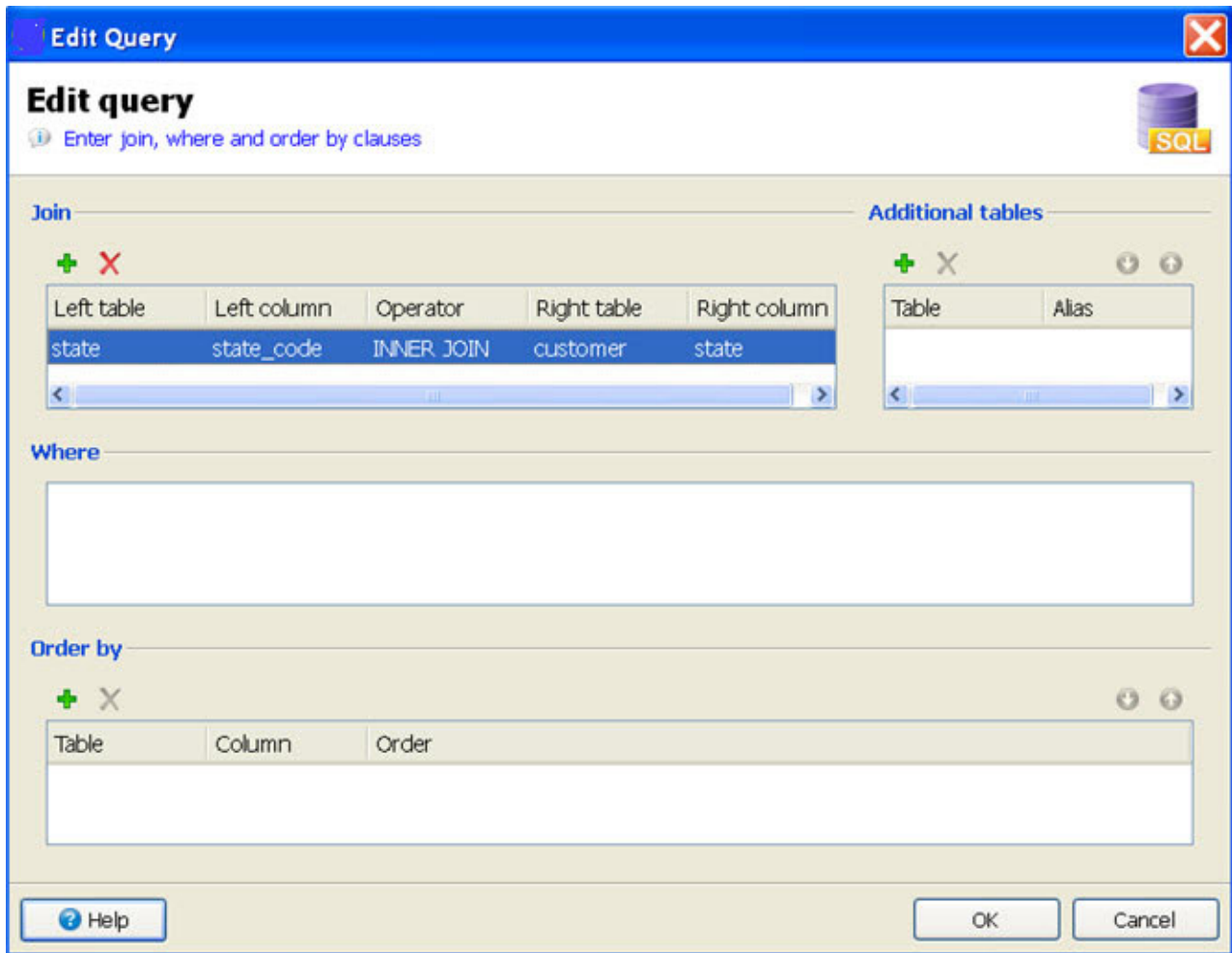


Figure 3-34. Edit Query dialog

- Save the form and the project.

4. Build the Application

You can build your application from the right-click menu for the Program icon in your Business Application Diagram or for the application node in your Project structure.

Application Generator automatically generates the code for the lookup field. This code is integrated into the generated functions that display the account record.

When the application is run (**Execute** from the right-click menu), the join between the master table and the reference table is executed and the lookup value is retrieved and displayed on the form. In our example, the country name from the country table is displayed along with the account information.

Populate a ComboBox Dynamically

When a user is entering data in a form, a ComboBox is useful to display a list of valid values for a field. If your Form Definition file (.4fdm) has a ComboBox, your Application Generator application can set the items for the ComboBox dynamically from the values in a database table.

In the Using Lookup Fields example, the form uses a the country.codedesc column from the country table as a lookup. This field could be changed to a ComboBox and loaded dynamically. The country.codedesc field has been converted to ComboBox and its initializer property has been set. The account.country field has been converted to a Phantom field so that it is not seen on the form, but is used in the code.

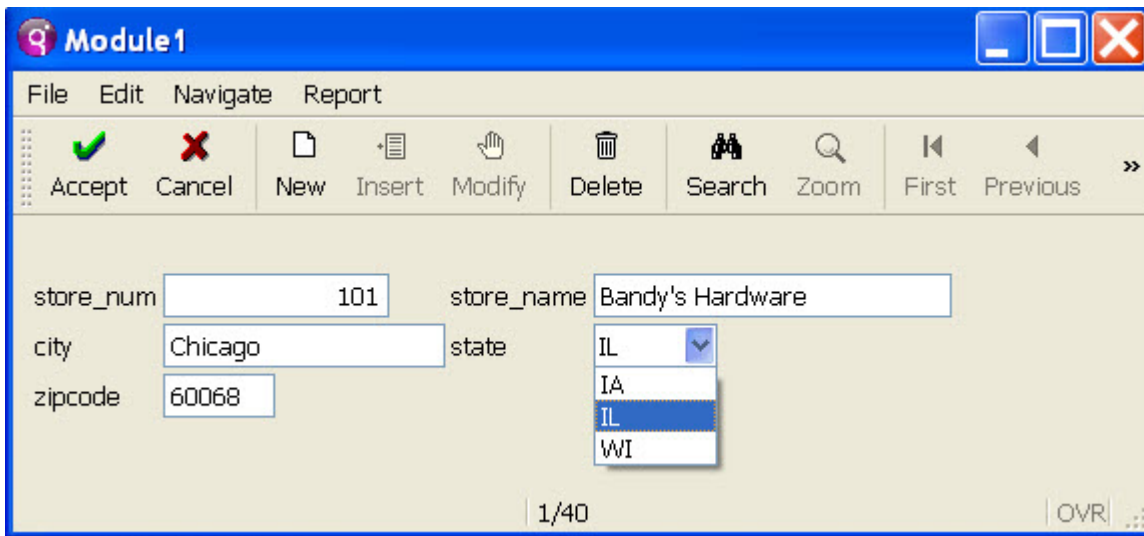


Figure 3-35. Module1 application

When the user selects New or Modify, and clicks on the arrow of the country field ComboBox, a list of the valid values is displayed. When the user selects a country description and accepts the selection, the account table is updated with the correct country information (the country code.)

Application Generator includes the program logic that loads the ComboBox in the .4gl file generated for the .4fdm file.

A reference table for a ComboBox provides two values:

- The value to be **stored** in the form, which is from the primary key column of the reference table and defined in the join between the tables
- The value to be **displayed** on the form, which is the value of the column specified in the properties of the ComboBox form field

These values may be from the same column in the reference table, or the display value can be a different column.

The Form Definition File

Make the following changes in the form:

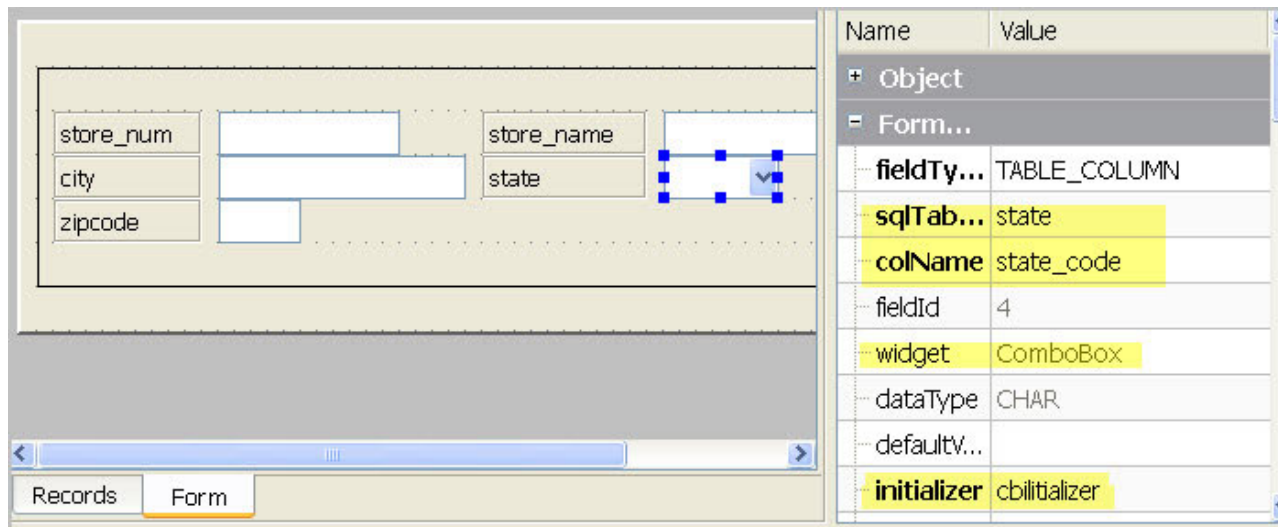


Figure 3-36. Form Definition file

- Convert the **account.country** field to a Phantom widget.
- Convert the **country.codedesc** field to a ComboBox widget and set its properties:
 - **name** - the ComboBox must have a unique name as there may be multiple ComboBoxes in the application.
 - **sqlTabName** - the tableName of the reference table (in the example, the **country** table instead of the account table)
 - **colName** - the column name of the display value (in the example, **codedesc**); this value will display in the ComboBox for the user to select.
 - **initializer** - assign a name to the function that will be called to initialize the ComboBox (in the example, **fillcombo**). Since the initializer function that is generated by Application Generator will use this name, it must also be unique across the application.

Click the **Records** tab. Make sure the **lookup** property is set on the **country.codedesc** column. This will be used as a function name in the generated code that is triggered to perform the lookup.

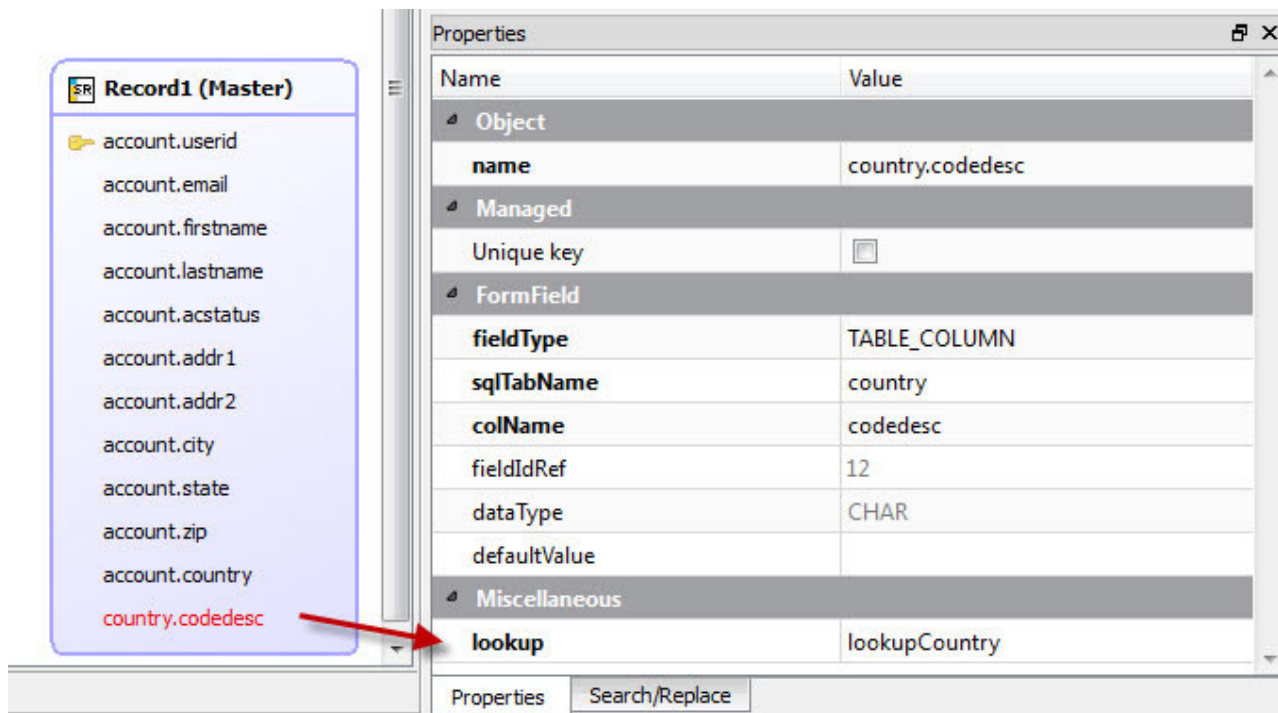


Figure 3-37. Lookup

Right-click and select **Edit Query**, to invoke the Query dialog.

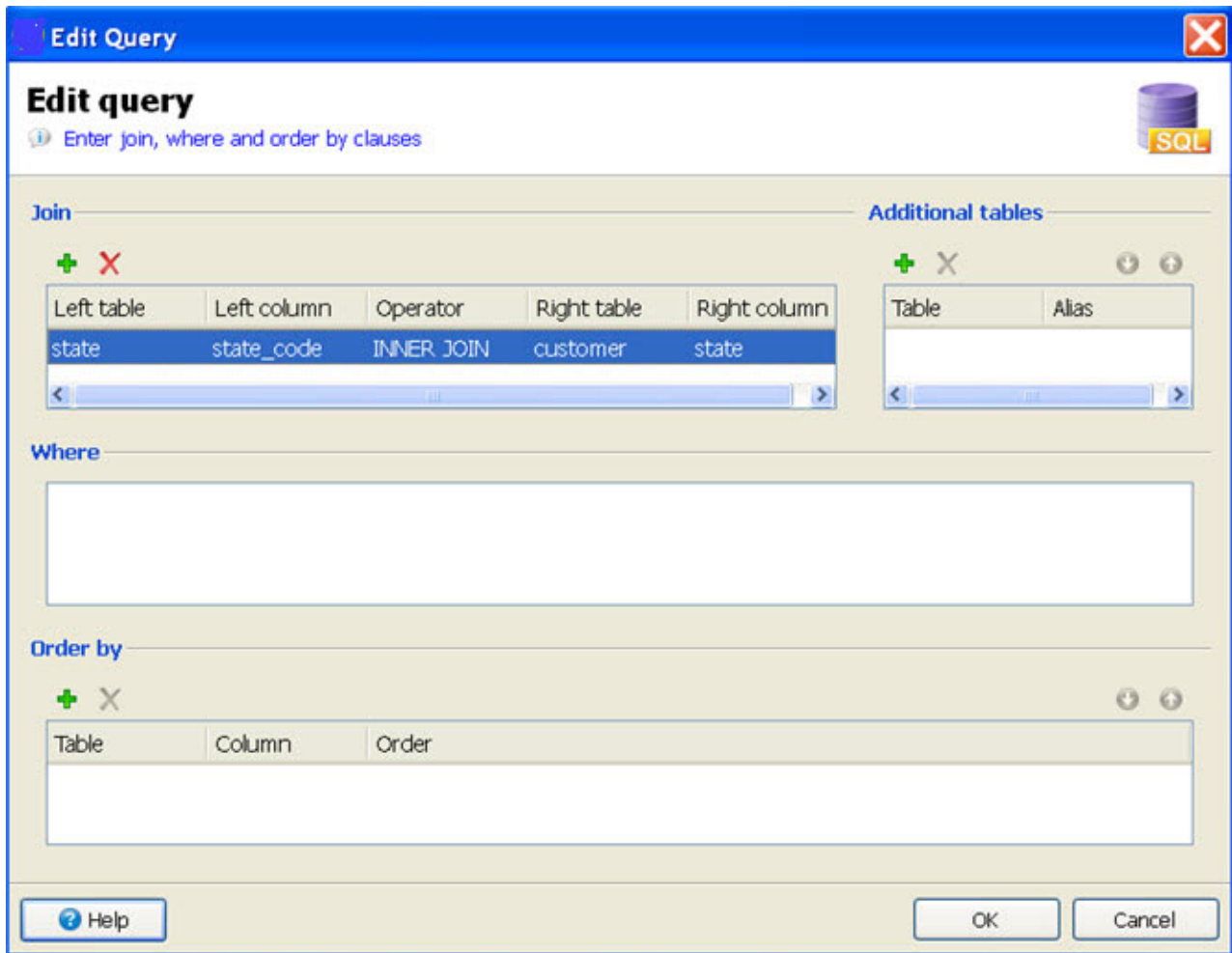


Figure 3-38. Edit Query dialog

Confirm that there is a join condition between the master and reference tables. In the example, the account table and the country table are joined on the columns for the country code. Click **OK**, and save the changes to the Form Definition file.

Generating and Executing the Application

Build the program as usual. Run the program by selecting **Execute** from the right-click menu of the Application Node in the project, or the Program icon in the Business Application Diagram.

Program Flow: Opening Zooms and Additional Forms

Topics:

- Opening a zoom form
 - Create the Zoom form
 - Modify the main form to include an action view
 - Create a relation between the Zoom and Main form
 - Generate the program
- Opening a second form

Opening a Zoom form

Zoom forms contain logic that allow the user to pick a value from a list that is displayed in a form in a popup window. Zoom forms can be added to your generated application through the Business Application Diagram.

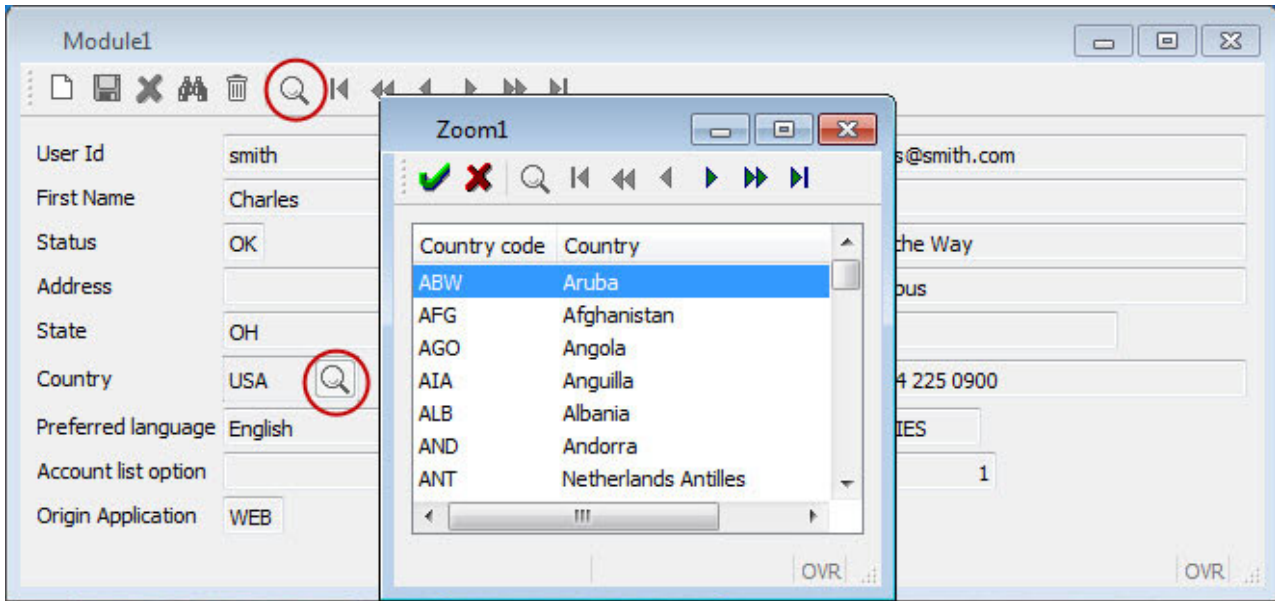


Figure 3-39. Zoom1 dialog

In this example, a Zoom form is created that displays a list of country codes from the **country** database table. When the user triggers the zoom action to occur, either by selecting the zoom icon on the toolbar or from the zoom field, a list of country codes displays in a popup window. When a country code is chosen from the list, the zoom form closes and the value is populated in the corresponding field on the main form.

Create the Zoom Form

1. In the Business Application Diagram, right-click on the page background and select **New Zoom Form** to create the Zoom Form icon.
2. Right-click on the Zoom Form icon and select **Implement Zoom from Database** to generate a form. Select the database table (in our example, the **country** table) that contains the values to be displayed in the zoom form. Select the column that corresponds to the field on the main form. In our example, this is the country code. (You can select the country name also, to display it in the zoom form if you wish; the code field will be identified as the unique key later in this process.)

Country code	Country
Edit	Edit
Edit	Edit
Edit	Edit
Edit	Edit
Edit	Edit

Figure 3-40. Example zoom form definition

From the Records tab:

- Select the **master table record**, make sure **active** property is checked.
- Select and **identify a column as the unique key**, if not already identified, as indicated with the key icon.

Name	Value
Object	
name	Record1
Managed	
active	☒
masterTable	country

Figure 3-41. Records tab.

Save the Zoom form (.4fdz) and add it to the project structure under the Application node.

Modify the main form

The main form has to be modified to trigger the zoom action when the user wishes. In our example, we'll change the country.code field to a ButtonEdit, which can trigger an action when the user clicks it. When the user selects the country code from the zoom form, it is automatically inserted into the country.code field on the main form.

1. From the .4ba, right-click on the main form and choose Open Form.
2. Select the field containing the foreign key (in our example, the **account.country field**.) Right-click to display the context menu and select **Convert Widget** to change the widget to a ButtonEdit:

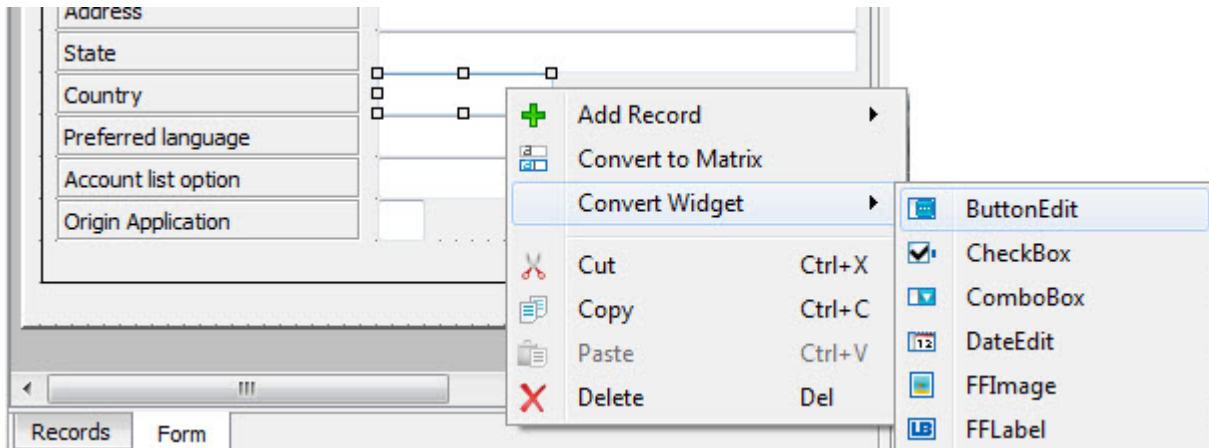


Figure 3-42. Convert Widget

3. Scroll to the bottom of the properties list, and set the value of the **action** property for the ButtonEdit field to a unique action name (such as **zoom**.)
4. Save the form.

Create a relation between the Main form and the Zoom in the BAM diagram

The Zoom form (.4fdz) must be linked to the main form of your application.

On the Business Application diagram :

1. **Right-click the Form icon (.4fdm)** and select **New Relation**.
2. **Click the Form icon and drag** the mouse towards the Zoom icon, creating an arrow link.
3. **Select the relation arrow** to display the relation properties, and enter the action name in the **action property** that is the same as the one assigned to your ButtonEdit field (**zoom**, in our example.) If there is more than one zoom form in an application, the action name must be unique.
4. **Fill in the Source and Destination fields** ; the value for the **Destination Field** property that the user selects from the table in the zoom form (country.code, in our example) will be inserted into the **Source Field** (account.country) on the form.
5. The Open Mode property can be DISPLAY (default) or SEARCH.

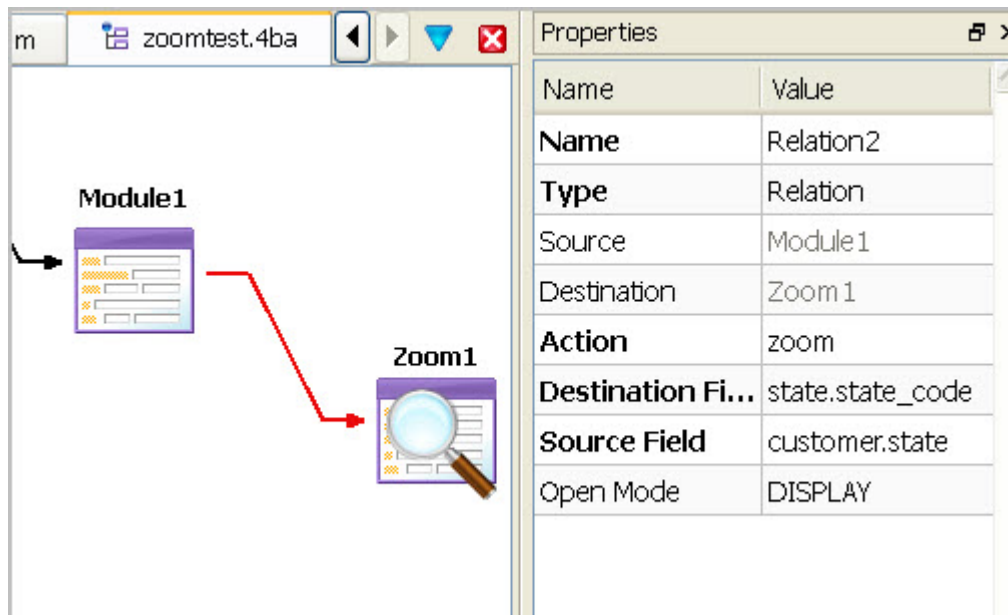


Figure 3-43. Business Application Diagram

Select **File>>Save** or **File>>Save All** to save the modifications.

Generate the Program

To generate the program, right-click the Program icon on the Business Application Diagram or the Application the Project Structure, and select **Build** from the context menu. This will build the application again, incorporating the changes you have made.

To run the program, select **Execute** from the context menu. When entering a new record, place the cursor in the source field of the form (in our example, the account.country field); then, click the button on this field to display the zoom form and select the desired value from the list displayed (values in the country table, in our example).

Opening A Second Form

You can open additional forms in your generated application. The following example illustrates opening a second form in its own window, and adding a toolbar item to trigger that form.

Add a new form to an existing BAM diagram

1. **Open a BAM diagram** (.4ba) that already contains a Program and Form definition.
2. Right-click on the background to add a new CRUD Form.
3. **Right-click the new form** and select **Implement Form from Database** from the context menu.
4. **Select the database table and fields** that you want to have on the new form (the product table, in our example.) Finish and save the form definition file (.4fdm) to the Application node of your project.
5. **Right-click the Program icon** and select **New Relation**. Click the Program icon again and drag the relation arrow to the second form.

6. **Right-click on the relation arrow** in the BAM diagram and set the **action** property in the Properties View to the action name that you wish to trigger (**products** in our example).
7. **Save the form and modified BAM diagram.**

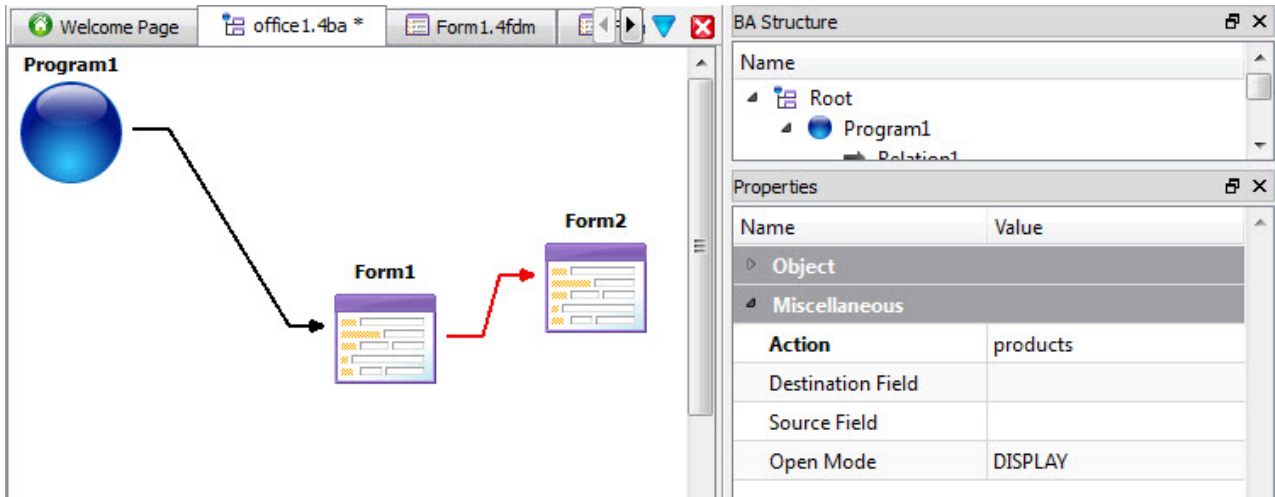


Figure 3-44. The modified BAM diagram

Add an item to the default toolbar for the form

- Open the main form.
- **Right-click the main form** and select **Open Form** from the context menu.
- When the form displays, **right-click the form** and select **Import Toolbar** from the context menu. The default toolbar (.4tb) is stored in the Application Generation templates folder, <studio-install-dir>/gst/bin/src/ag/bin/tpl/dbapp/resource. The default toolbar and its items will display in the Form Structure view.
- **Right-click on the Toolbar item** in the list, and choose **Add Item**.
- **Assign the name to the new toolbar item** (products in our example) in the **name** property of the Properties View; the name must be in lowercase letters. This name must be the same as the name of the action that is to be triggered (the display of the new form):

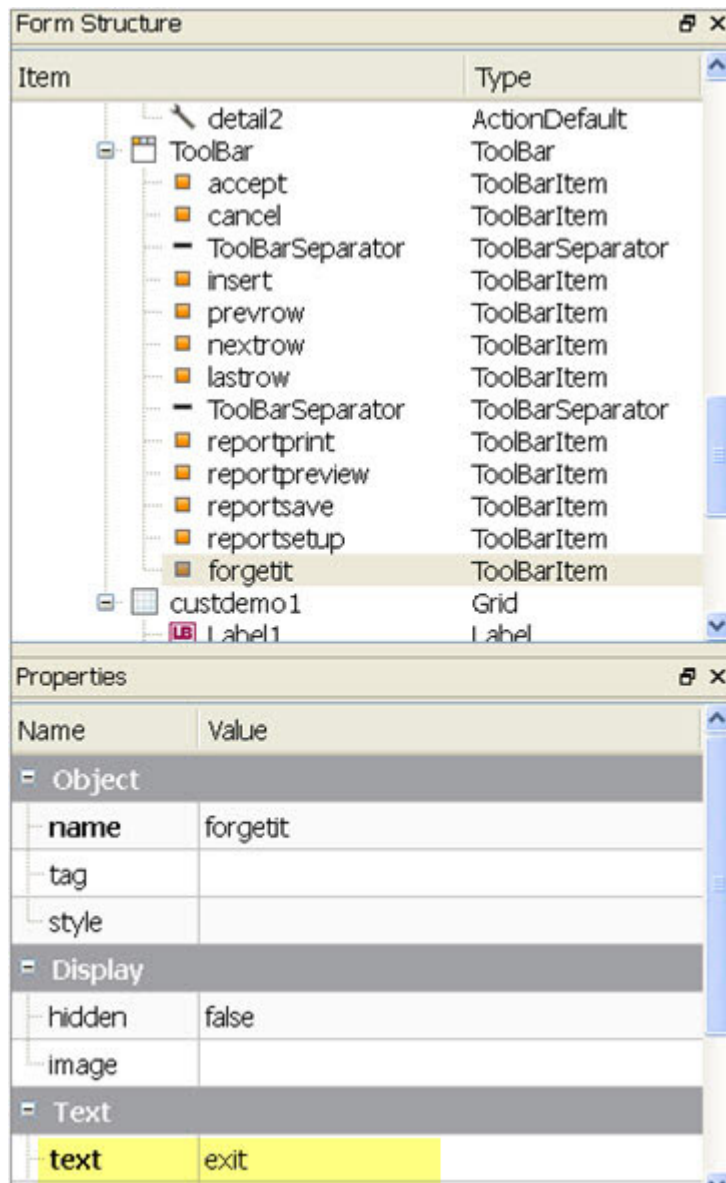


Figure 3-45. New toolbar item

You can also add text and an image for the toolbar item, and a comment to display as a tooltip. If you add an image, it must be included in \$FGLIMAGEPATH.

Note: If you do not add an action view such as a toolbar item, button, or topmenu item to the form, a button to trigger the new action will be automatically generated on the main form when the program is run.

- **Save the modified form** and the changes in the BAM structure using **File>>Save** and **Save>>All**.

Regenerate the program and run it

- Right-click the Application icon on the Business Application Diagram or in the Project Structure, and select **Build** from the context menu. This will build the application again, incorporating the changes you have made.

- To run the program, select **Execute** from the context menu. When the user clicks the **Products** toolbar icon, the second form will display in a separate window:

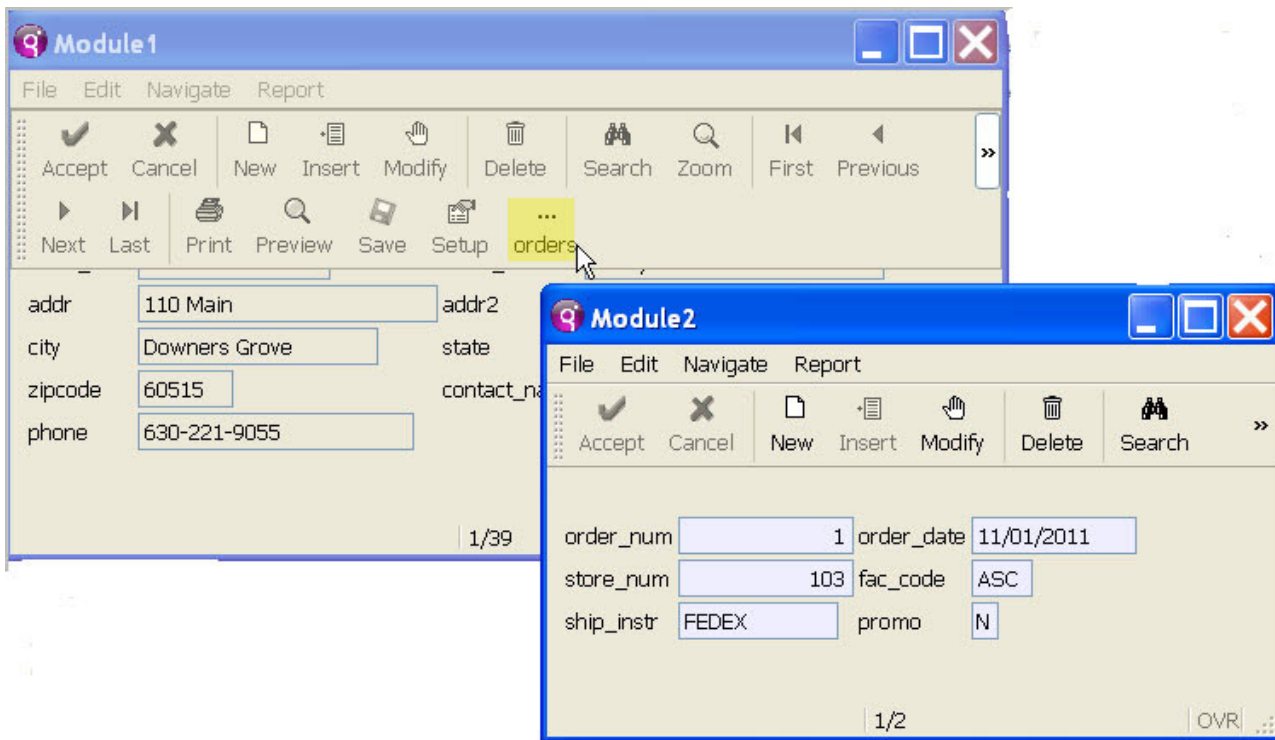


Figure 3-46. Application with Module1 and Module2

Actions and Action Views

Topics:

- Default Actions
- Default Action Views
- Adding to the TopMenu
- Adding to the Toolbar
- Adding Buttons
- Modifying Action Defaults

See also: Your First Application

Default Actions

The user interface for your Application Generator program is based on the Form Definition file created for the program. Generated applications have default actions, which are triggered when the user clicks on an action view (TopMenu Command, Toolbar item, or Button) on the application form. The actions are enabled appropriately as the BDL interactive statements in the generated application are executed.

When you set the Functionality properties for your generated program on the Business Application Model diagram, you are specifying which default actions should be available to the user. For example, if the **can search** property is checked, for example, the default actions needed to enter criteria and search the database would be generated,

Default Action Views

The application form for your generated program contains default Action Views (TopMenu Commands, Toolbar items, or Buttons), allowing the user to trigger an action by clicking the action view.

If a form contains multiple tables, additional actions are automatically defined, and action views are created to allow the user to switch between tables on the form.

Here's an example form for an application generated using the default template, Database Applications (dbApp.tpl).

Figure 3-47. Application generated by the default template

The action views for the default actions are toolbar icons and TopMenu items, which can trigger the following actions:

- **New** - Add a new database row (action append)
- **Insert** - Add a new database row (action append)
- **Modify** - Update a database row (action modify)
- **Search** - Search the database table (action query); enter criteria in the relevant fields and click Accept; or click Accept on an empty form to retrieve all the records in the database table
- **Delete** - Delete a database row (action delete)
- **Zoom** - activate the zoom form (action zoom)
- **First** - move to the first record (action firsttrow)
- **Last** - move to the last record (action lasttrow)
- **Next, Previous** - move between the rows displayed in a list on the form (actions nextrow/prevrow)

If a report was defined for this example application, additional toolbar items would be enabled for the user:

- **Print** - print a defined report (action reportprint)
- **Preview** - preview a defined report (action reportpreview)
- **Print...** - select a report and configure the print settings (action reportsetup)
- **PDF** - Export report to PDF format
- **HTML** - Export report to HTML format
- **XLS** - Export report to XLS (Excel) format
- **RTF** - Export report to RTF (MS-Word) format

You can define additional graphical objects for the form, such as TopMenus, Toolbar items, or Buttons, for the application's actions, and you can modify the Action Defaults, which specify the default decoration and accelerator keys for a default action.

Adding to the TopMenu

Note: Adding a TopMenu item to a form will replace the default Topmenu. You must import the existing TopMenu before adding items.

Right-click the background of the form in Studio Form Designer, and select **Import default TopMenu** from the context menu. The default topMenu file for applications generated using the default dbApp template is: `<studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4tm`.

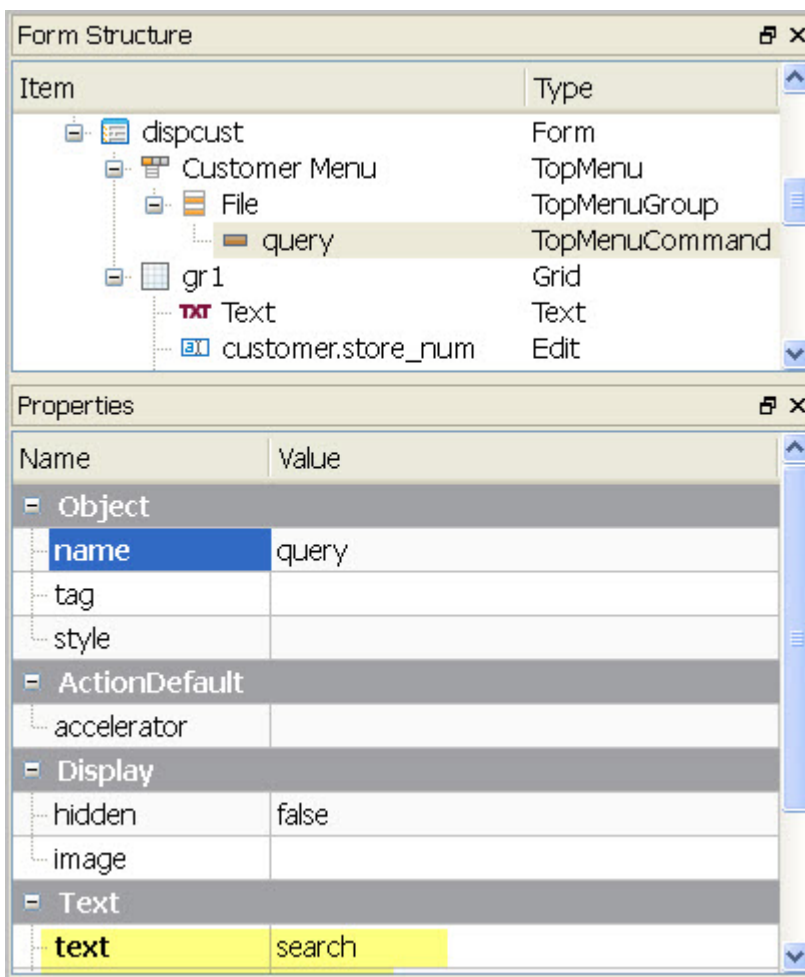


Figure 3-48. TopMenuCommand example

The following objects make up a TopMenu:

- **TopMenuGroup** - All commands that make up the menu are organized into groups.

Right-click the TopMenu in the Structure View, and select **Add Group** from the context menu.

- **TopMenuCommand** - specifies the action the menu option must be bound to.
Right-click a TopMenuGroup in the Structure View and select **Add Command** to add this object to the form.
- **Separator** - an optional vertical line to be displayed on the menu, visually separating some of the commands.
Right-click a TopMenuGroup in the Structure View and select **Add Separator** to add this object to the form.

Select the TopMenu object that you have added to set its properties:

- **TopMenu Group** - set the **title** of the new group using its text property.
- **Top Menu Command:**
 - Set its text property to the name of the default action to be triggered. The value must exactly match the name of the default action being triggered; action names are case-sensitive.
 - Optionally add a value for the accelerator property, to allow the command to be triggered from the keyboard.

Save the modified form.

If you wish to change the default TopMenu for all your generated applications, as an alternative to making the changes on each form, you can modify the default TopMenu.4tm file, `<studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4tm`. We suggest you make a copy of this XML file before making changes to the original.

Adding to the Toolbar

Note: Adding a Toolbar item to a form will replace the default Toolbar. You must import the existing Toolbar before adding items.

Right-click the background of the form in Studio Form Designer, and select **Import default Toolbar** from the context menu. The default Toolbar file for applications generated using the default template dbApp is: `<studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4tb`.

Modify or add new items to the Toolbar.

The following objects make up a Toolbar:

- **ToolbarItem** - specifies the action the toolbar button must be bound to.
Right-click the Toolbar in the Structure view and select **Add Command** to add this object to the form.
- **Separator** - an optional vertical line to be displayed on the Toolbar, visually separating some of the commands.
Right-click a Toolbar object in the Structure View and select **Add Separator** to add this object to the form.

Select the **Toolbar Item** to set its text property to the name of the default action to be triggered. The value must exactly match the name of the action; the names are case-sensitive.

If you wish to change the default Toolbar for all your generated applications, as an alternative to making the changes on each form, you can modify the default

Toolbar.4tb file, <studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4tb. We suggest you make a copy of this XML file before making changes to the original.

Adding Buttons

If you add a Button to your Form Definition file, one of the actions that is automatically generated in the application can be associated with this Button.

Select **Widget>>Button** from the Studio main menu, or use the Button toolbar icon, to add a button to the container of the form.

Assign the name of a default action as the name property of the Button; action names are case-sensitive.

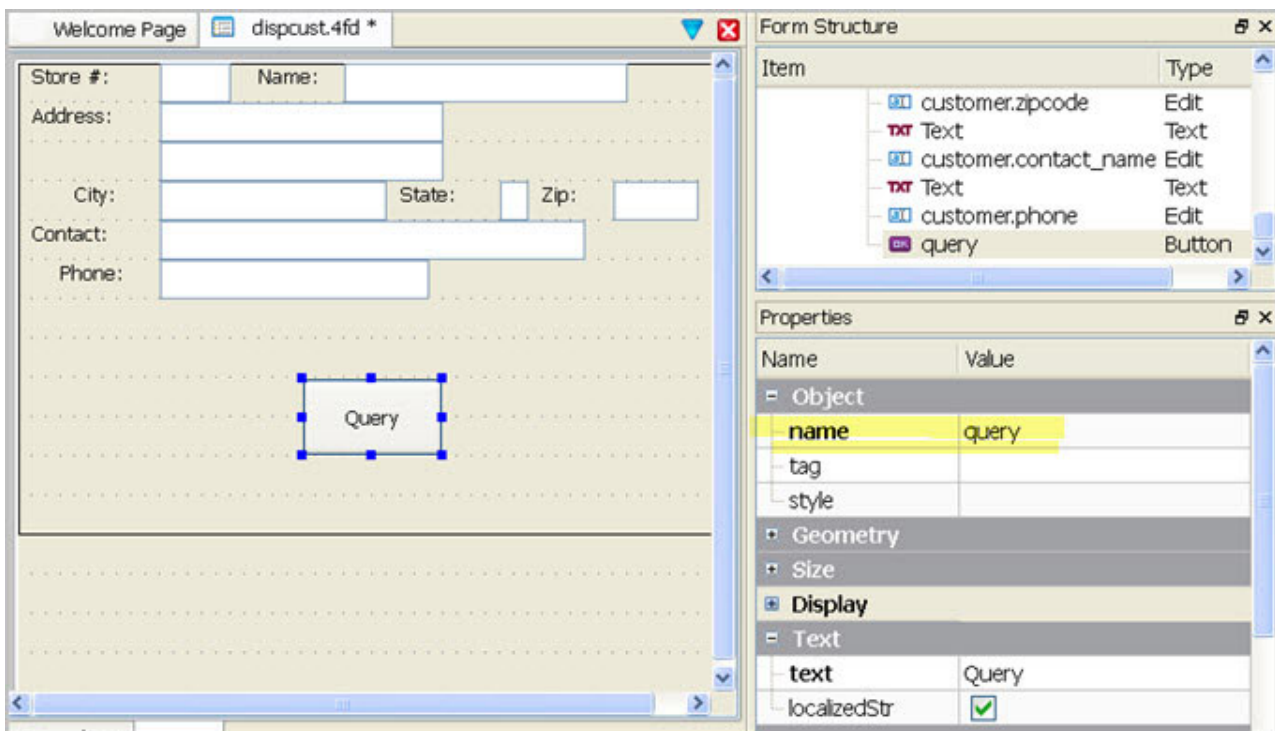


Figure 3-49. Add Button

Modifying Action Defaults (dbapp.4ad)

You can add or modify the attributes (the decoration) for the default actions of an application that was generated using the default dbApp template.

Note: You must import the existing action defaults file for the form before adding or modifying their default properties.

Right-click on the background of the form in the central work area of Studio Form Designer, and select **Import Action Defaults** from the context menu. The default file for applications generated using the default template dbApp is:
<studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4ad.

In the Form Structure View, select the action whose decoration you wish to change.

In the Properties View, make the desired changes for this action; for example, change an accelerator, image, or the text displayed for the action view.

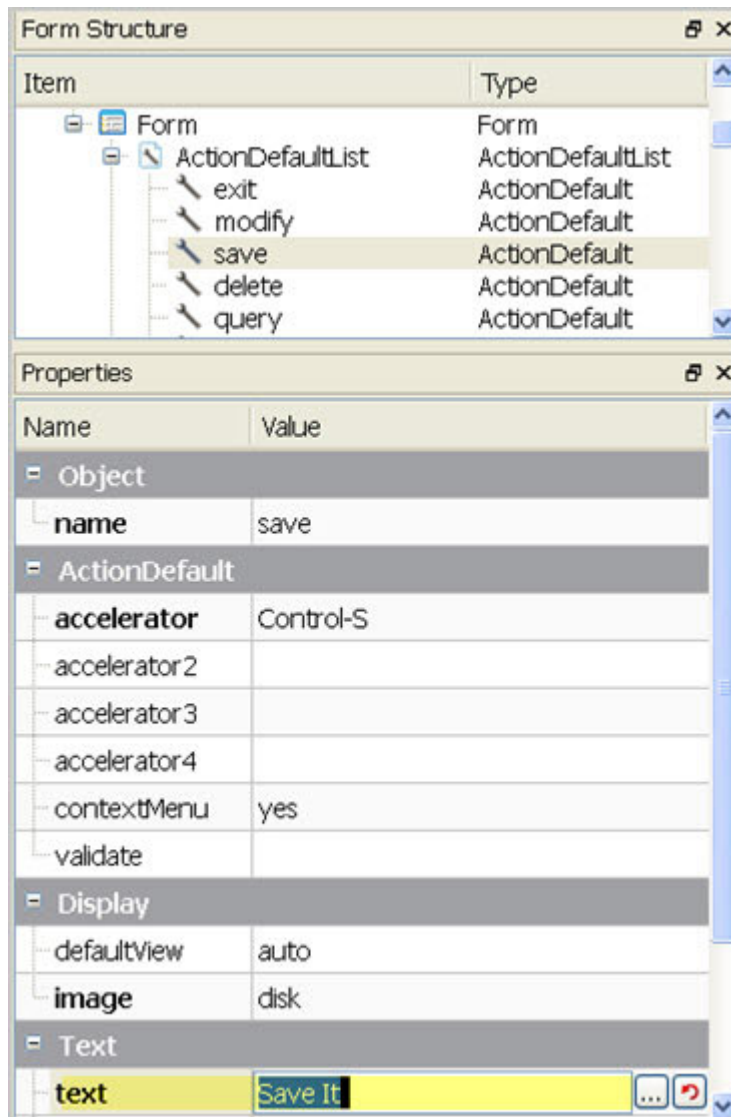


Figure 3-50. Change the text in the Properties View

If you wish to change the action defaults for all your generated applications, as an alternative to specifying the attributes and accelerator keys on each form, you can modify the default file, `<studio-install-dir>/gst/bin/src/ag/tpl/dbapp/resource/dbApp.4ad`. We suggest you make a copy of this XML file before making changes to the original.

The Application Source Code and Files

Topics:

- Application Files
 - Studio Project Files
 - Intermediate BDL Files
 - Resource Files

- Executable Application
- The Program .4gl File
- The Module .4gl Files
- Report .4gl Files
- Zoom .4gl Files
- The Library file (libdbappCommon.4gl)
- Adding Code
 - POINTS
 - BLOCKS

Application Files

- **Project files - Studio source files for the application**
 - . The following files are stored in the project directory:
 - *project.4pw* - project file
 - *project.4pwdb* - internal file containing application symbols/elements.
 - *project.4pwvs* - maintains record of open files in project
 - *bam.4ba* - file for Business Application Model diagram
 - *bam.4bats* - internal file timestamp for compilation of .4ba
 - *program.4prg* - definition file for the Program
 - *module.4fdm* - definition file for a Module
 - *report.4rd* - definition file for a Report
 - *report-name.4rp* - the report definition file created by the developer
 - *zoom.4fdz* - the definition file for a Zoom module
- **Intermediate Genero BDL files - generated from project files**
 - . The following files are generated intermediate files displayed in the project directory folder **Intermediate Files**.
 - *program.4gl* - BDL file containing the MAIN function for the application
 - *program.code* - contains any user-added BDL statements
 - *program.xml* - contains the description and properties of the program.4prg
 - *module_ui.4gl* - contains graphical/interaction functions
 - *module_ui.code* - contains any user-added BDL statements
 - *module_uidata.4gl* - contains database and data manipulation functions
 - *module_uidata.code* - contains any user-added BDL statements
 - *module.xml* - contains the description and properties of the module.4fdm
 - *reportdata_report.4gl* - contains the report driver for the report
 - *reportdata_reportdata.4gl* - contains the SQL and BDL statements to fetch the report data
 - *reportdata.xml* - contains the description and properties of the *reportdata.4gl*
 - *reportdata.rdd* - contains the data file for the report definition
 - *zoom_zoom.4gl* - contains graphical/interaction functions
 - *zoom_zoomdata.4gl* - contains data functions
 - *zoom.xml* - contains the description and properties of the module.4fdm
- **Resource Files for the application**
 - . Contains the files that define the Toolbar, TopMenu, action defaults, presentation styles, and database schema for the generated application. The files are stored in <Genero Studio install path>/gst/bin/src/ag/tpl/dbapp/resource.

- **dbapp.4ad** - Genero action defaults file that specifies default attributes (such as text to be displayed, comments, and images) for graphical objects of the user interface associated with actions defined in the application
 - **dbapp.4st** - Genero presentation styles file that specifies attributes related to the decoration of the graphical elements of the user interface, grouped in styles
 - **dbapp.4tb** - default Genero Toolbar definition file for the application form
 - **dbapp.4tm** - default Genero TopMenu definition file for the application form
 - **libdbapp*.4gl** files - library files for common functionality that are automatically imported, if needed, into the application's generated .4gl files;
 - **Executable Application - compiled binary files to be deployed as the application**
 - The default location for these files is **<project-dir>/bin**.
 - **Application.42r** - the executable program file, the compiled and linked files
 - **custdemo.sch** - the Genero schema file for the database used by the application
 - **Library.42x** - the compiled files in the Library node for the application
 - **Program.42m** - the compiled Program .4gl file
 - **Module.42f** - the compiled form definition file from the Module
 - **Module_ui.42m** - the compiled *module_ui*.4gl files
 - **Module_uidata.42m** - the compiled *module_uidata*.4gl file
 - **Zoom_zoom.42m** - the compiled .4gl file, if included in your program
 - **Zoom_zoomdata.42m** - the compiled .4gl file, if included in your program
 - **reportdata_report.42m** - the compiled *reportdata_report*.4gl file, if reports are included in your BAM diagram
 - **reportdata_reportdata.4gl** - the compiled *reportdata_reportdata*.4gl file, if applicable
- In addition, the default resource files, or the modified files if you've made changes to them, must be deployed.

The Program .4gl File

A BDL .4gl file is generated for the Application Generator Program definition file (**4prg**). This file, **<programname>.4gl**, defines global variables and contains the logic to set up the application. It is compiled and linked with the other program files to create the executable application. The .4gl file is listed in the project as an intermediate file. The compiled version is stored in the **<project-name>/bin** directory.

The MAIN program block:

- Contains the DATABASE BDL statement
- Loads the styles and action defaults files
- Performs some initialization tasks for the application
- Calls the main function in the Genero .4gl file generated for the AG Module containing the master table

POINTS and BLOCKS allow you to add code to the file. See Adding Code.

Open this file in the Intermediate Files folder of the project structure using the right-click menu option **Open**.

The Module .4gl Files

Two BDL .4gl files are generated for each AG Module definition file.

Module_ui.4gl:

- Defines the module variables
- Launches the module called from MAIN, based on the startup mode
- Opens the application form
- Loads the toolbar and TopMenu
- Does some setup for the tables and keys
- Launches the application state manager function
- Launches the functions containing the Menu or Dialog statements for the various states
- Calls functions in the Module_uidata.4gl to retrieve and manage the database data

Module_uidata.4gl:

- Defines the records for the database columns and tables
- Creates and fills a dynamic row of data
- Creates and fills a dynamic row of record keys
- Contains the SQL statements to select, update, and delete rows in the database
- Handles the SQL transactions

POINTS and BLOCKS allow you to add code to the file. See Adding Code.

Open these files in the Intermediate Files folder of the project structure using the right-click menu option **Open**.

Report .4gl Files

If you have included a Report Writer report in your BAM diagram, the following BDL files are generated.

- *reportdata_report.4gl* - contains the report driver for the report
- *reportdata_reportdata.4gl* - contains the SQL and BDL statements to fetch the report data

POINTS and BLOCKS allow you to add code to the file. See Adding Code.

Open these files in the Intermediate Files folder of the project structure using the right-click menu option **Open**.

In addition, a data file (*reportdata_report.rdd*) is generated for use when the developer creates the report definition file (.4rp).

Zoom Module .4gl Files

If you have included a Zoom Module in your BAM diagram, the following BDL files are generated:

- *zoom_zoom.4gl*
- *zoom_zoomdata.4gl*

POINTS and BLOCKS allow you to add code to the file. See Adding Code.

Open these files in the Intermediate Files folder of the project structure using the right-click menu option **Open**.

The Library file (libdbappCommon.4gl)

The functions in this file handle some of the generic tasks associated with the generated application, such as:

- Defines public constants and module variables
- Contains utility functions for the overall application

This file is stored in `<Genero Studio install path>/bin/src/ag/tpl/dbapp/resource>`.

A BDL IMPORT statement in the other *Module.4gl* files, adds the code in this file to the application.

Adding Code

You may add your own code to the BDL code generated by Application Generator, using a text editor such as Genero Studio Code Editor.

After you add code using a POINT or a BLOCK, the application can be regenerated and your additions will not be lost. They will be preserved in a **.code file** that is used each time the application is compiled. If you want to remove all the changes that you've made, you can simply remove the .code files.

POINTS

A POINT is an area where you might wish to add specific code, appropriate to the location of the POINT, such as:

- definition of variables
- conditions on SELECT statements
- control blocks of interactive dialog statements such as CONSTRUCT, INPUT, MENU
- a change of program flow

The developer is responsible for the accuracy of the code in the POINT.

See Modifying the Application.

BLOCKS

A BLOCK delineates a larger area within which you may add the code that you wish. Once code has been added to a BLOCK, however, the developer takes responsibility for the validity of all the code in that BLOCK.

See Modifying the Application.

Modifying the Application

Topics:

- POINTS and BLOCKS
- Adding your Own Functions
- Adding your Own Actions
- Adding Additional Variables

POINTS and BLOCKS

You may add your own code to the BDL code generated by Application Generator. Your code is added in special areas of the generated code called a POINT and a BLOCK. The additional code you add to a POINT or BLOCK is preserved in the application even when it is rebuilt.

The .code File

Your additions are preserved in a **.code file** that is used each time the application is compiled.

- If you want to remove all the changes that you've made, you can simply remove the .code files.
- If you are using source code management, the .code file must be committed with the project. It is not necessary to commit the generated .4gl files.

POINT

A POINT is an area where you might wish to add specific code, appropriate to the location of the POINT, such as:

- definition of variables
- conditions on SELECT statements
- control blocks of interactive dialog statements such as CONSTRUCT, INPUT, MENU
- a change of program flow

The developer is responsible for the accuracy of the code in the POINT.

BLOCK

A BLOCK delineates a larger area within which you may add the code that you wish. Once code has been added to a BLOCK, however, the developer takes responsibility for the validity of all the code in that BLOCK.

Adding your Own Functions

At the bottom of the code for each **.4gl** generated for your application there is a POINT, allowing you to add additional functions to the file:

This example adds a new function **dispmsg**:

```
{<POINT Name="add_functions">}
FUNCTION dispmsg()
    MESSAGE "Program ending"
END FUNCTION
{</POINT>}
```

```

2838 {<POINT Name="add_functions">}
2839 FUNCTION dispmsg()
2840     MESSAGE "Program ending"
2841 END FUNCTION
2842 {</POINT>}
2843
2844 #</MODULE>

```

Figure 3-51. Example of an added function.

Adding Your Own Actions

There is a POINT section for each of the interactive dialog statements such as DISPLAY, CONSTRUCT, and MENU for adding your own actions. To add actions to the MENU, you use the POINT `fct.uiEmpty.menu.addMenuOptions`.

For the INPUT statement, the POINT is named `fct.uiInput.dialog.addControlBlocks`. This example adds an action `myaction` that calls a previously added `dispmsg` function to the INPUT statement:

```

{<POINT Name="Dialog.input.add.ON_ACTION">}
ON ACTION myaction
    CALL dispmsg()
{</POINT>}
AFTER INPUT
{<POINT Name="Dialog.input.top.AFTER_INPUT">}
{</POINT>}

```

```

806
807 {<POINT Name="Dialog.input.add.ON_ACTION">}
808 ON ACTION myaction
809     CALL dispmsg()
810 {</POINT>}
811
812 AFTER INPUT
813     {<POINT Name="Dialog.input.top.AFTER_INPUT">}
814     {</POINT>}

```

Figure 3-52. Add a custom action

Adding Additional Variables

At the end of each DEFINE statement in the .4gl file there is a POINT that allows you to define any additional variables that might be required by the code you have added to the application. The scope of the variable is determined by the location in the .4gl file of the DEFINE statement.

This example is at the beginning of the module (modular scope):

```

189 {<POINT Name="define.module.add_variables">}
190     DEFINE myvar STRING
191 {</POINT>}
192
193 {<BLOCK Name="MAIN">}
194 -----

```

Figure 3-53. Add a custom variable

Local variables are added in the POINT section for the function:

```

61 DEFINE l_detailList STRING
62 DEFINE l_startMode SMALLINT
63 DEFINE l_actionList DYNAMIC ARRAY OF STRING
64 DEFINE w ui.Window
65 {<POINT Name="fct.uiOpenForm.define">} {</POINT>}
66

```

Figure 3-54. Add a local variable

Customization

Topics:

- Creating a Custom Template - Overview
- Use case: Create a new File Type
- XSLT template for Code Generation

See: Modifying Your Application

Creating a Custom Template - Overview

Additional features can be created by adding new properties or node types to the existing templates and associated files. Modifying the default templates requires a basic knowledge of Genero Business Development Language and the TCL language or another template language. Genero Application Generator uses the TCL Generator by default; it can be replaced by a generator of your choice:

- XSLT, perl, python, and so on
- Genero BDL

We do not recommend that you change the default templates. Instead, create your own template directory:

- Copy the entire content of the `<Studio-install-dir>/gst/bin/src/ag/tpl/dbapp` directory to a new folder. The new template directory could be a common one on the Server, or It could be part of the entire versioned project:
- Using Tools>>Preferences, Application Generator, add a new template name, and set the path to the new directory.

Now you can customize the files that you copied into the new directory. Keep the same filenames as the default template set. All your programs will be generated with these files; the default files will be ignored.

See Application Generator Templates for a description of the templates and associated files.

Use Case: Create a New File Type

You can create a new file type by modifying the relevant files associated with the default template. Follow these steps to create the new file type and add it to the Studio **File>>New** dialog.

Note: We recommend copying the entire default template directory to a different location, and making the changes in the files in that copy. See Overview.

1. Edit `<yourtemplateDir>/file-types.xml` (See The file-types.xml File for a complete description of this file):
Add the new .4fdm file type and its elements - copy the existing entry for .4fdm in the file, and change at least the following properties:
 - mimeType: "application/generostudio-4fdm1"
 - extensions: "4fdm1"
 - description: "My new form type"
2. In the `<yourtemplateDir>/settings.agconf` file, copy the File element for the .4fdm file and paste it under the .4fdm File element. Change the type to 4fdm1. Be sure this File element is a child of the Form element like the others. Save the file. See The settings.agconf File for a complete description of this file.
3. Code generation is done during files compilation, using the Build Rules. Add a new Build Rule for the .4fdm1 in the file `<yourtemplateDir>/dbapp.4pw`, using the rule for .4fdm as a model. See Build Rules for Templates for a complete description. Once the Build Rules are modified, your new projects will have them.
4. The configuration file `creatables.conf`, in `<yourtemplateDir>` directory, contains the entries displayed in the File>>New dialog. Add an entry for your new File Type (creatable) to this file, For the example .4fdm1 file type, you can copy the entry for .4fdm, and change the type to .4fdm1. See The creatables.conf File for a complete description of this file.
5. If your modifications have included an image, copy the image file to the folders in the `<your-template-dir>/images` directory. Each image must have a file in the S1 through S5 folders, with the size varying from smaller to larger.

When you select your custom template in Application Generator Preferences, and click **OK**, Studio will load the default custom template and merge the entries from your `creatables.conf` file into the existing categories and sub-categories. Any new categories and sub-categories, with their entries, will be displayed in the File>>New dialog.

XSLT Template for Code Generation

The code for your application can be generated using XSLT instead of TCL. The following example demonstrates how to generate a file with AG using XSL Translator.

Generating a form

Instead of generating 4gl as the dbapp template does, we'll generate a .per file (Genero form definition file). We'll use a 4rd (report data) file and generate a .per form file with formfields corresponding to the 4rd record. We'll add an additional property `frmLabel` on each record field, which will be displayed to the left of of each .per file formfield.

- The new property is added to the .4rd definition in **settings.agconf** (changes are in bold):

```
<File extension=".4rd">
  <DynamicProperties>
    <DynamicProperty name="foreignFields" type="FIELDS" label="foreignFields"
      initialValue="" dynamicContent="srcFieldsContent"
      editorInfo="isDynamic:true" />
    <DynamicProperty name="primaryFields" type="FIELDS" label="primaryFields"
      initialValue="" dynamicContent="dstFieldsContent"
      editorInfo="isDynamic:true" />
    <DynamicProperty name="frmLabel" type="TEXT" label="Label FRM"
      initialValue="" />
  </DynamicProperties>
  <Items>
    <Item nodeName="Relation" srcProperty="foreignFields"
      dstProperty="primaryFields" dynamicProperties="primaryFields;foreignFields"/>
    <Item nodeName="RecordField" dynamicProperties="frmLabel" />
  </Items>
</File>
```

- The file **managedtextform.xml** is created, to describe what is to be generated. This file should be saved to the template directory. The GASDIR environment variable should be defined in the GAS configuration and added to PATH.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" />
  <xsl:template match="/">
    --This is generated by the XSL
```

```
DATABASE <xsl:for-each select="//Module"><xsl:value-of select="@databaseName"/>
</xsl:for-each>
```

```
LAYOUT
GRID
{
  <xsl:for-each select="//Fields/Field">
    <xsl:value-of select="DynamicProperties/DynamicProperty[@name='frmLabel']
      /@value"/>:
    [<xsl:value-of select="@column"/>]
  </xsl:for-each>
}
END -- LAYOUT
```

```
TABLES <xsl:for-each select="//Record"><xsl:value-of select="@table"/>
</xsl:for-each>
```

```
ATTRIBUTES
  <xsl:for-each select="//Fields/Field">
    <xsl:value-of select="@column"/> = <xsl:value-of select="@table"/>.
    <xsl:value-of select="@column"/>;
  </xsl:for-each>
```

```
END -- ATTRIBUTES
</xsl:template>
</xsl:stylesheet>
```

- A new project is created and a new A Business Application Diagram (.4ba) is created and saved to the project.
- A new .4rd file is added to the project and a record created. The properties for **frmLabel** are added to the record fields.
- The .4rd Build Rule is updated to generate a .per using xslt :

```
<BuildRule
  additionalDependencies=""
  commands="$(generate) -depth 0 -ba &quot;$(BAFilePath)&quot;
  &quot;$(InputPath)&quot;;fglxslp -o
  $(InputDir)/$(InputBaseName).per $(TemplateDir)/managedTextForm.xml
```

```

    $(InputDir)/$(InputBaseName).xml;"
description="4RD Report Data compiler"
enabled="true"
fileType="application/generostudio-4rd"
id="1319189142826"
intermediateFiles=""
outputFiles="$(InputDir)/$(InputBaseName).per "/>

```

The same Build Rule displayed in a Studio dialog:

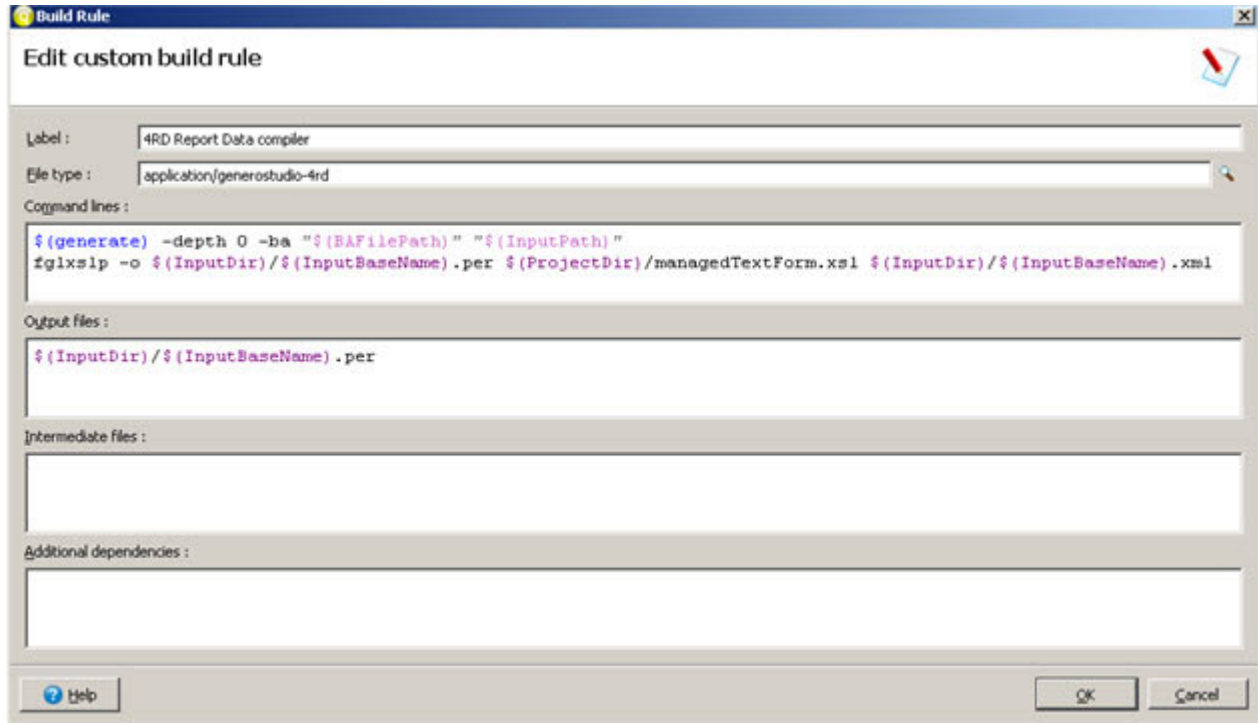


Figure 3-55. Build Rule

- The .4rd is compiled and a .per is generated.

The generated form

When the .per file is opened and previewed, it looks like this.

Figure 3-56. .PER form

Application Generation Templates

Topics:

- Default Application Generation templates
- Generation scripts (Templates)
 - List of TCL Templates
 - Template directory
 - Images directory
- Associated Files
 - File Types - file-types.xml
 - The creatables.conf file
 - The settings.agconf file
- Build Rules for Templates

See: Customization

Default Application Generation Templates

The Application Generator model is an XML file written during AG code generation. It is used by the AG templates to write the .4gl source code. By default

programs are generated with the template selected from the Tools>> Preferences, Application Generator main menu option. The currently selected template will be used to generate all .4pw (project) files.

The default template sets for application generation consist of files written in the TCL language.

- **dbapp** - contains the set of files for the latest application generation template, which enhances the behavior of applications that interact with a Genero form, and implements the latest 2.4x features.
- **2.3x** - contains the set of template files that provide the same application generation functionality as earlier versions of Studio.

All the template files for a set must be in the same directory. These files are located in subdirectories of the following directory: <Genero Studio-install-dir>/gst/bin/src/ag/tpl.

The templates are not per project or per file. Templates are defined for the entire development environment, which is set to point to the default AG templates directory.

See Source Code for a description of the project files and resource files used by the default dbApp template.

Generation scripts (Templates)

Scripts for the default dbApp template are provided in the TCL language; they generate the BDL database application code for the applications defined in the Business Application Model diagram.

Modifying the default templates requires a basic knowledge of Genero Business Development Language and the TCL language or another template language. Genero Application Generator uses the TCL Generator by default; it can be replaced by a generator of your choice:

- XSLT, perl, python, and so on.
- Genero BDL

List of TCL Templates

Table 3-1. List of TCL Templates

TCL template name	Usage	Associated Build Rule
common.tcl	Provide common procedures used by other TCL templates	
defines.tcl	Provide global defines used by other TCL templates	
error.tcl	Provide procedures used by other TCL templates to handle errors and warnings	
lookup.tcl	Generate the lookup DOM	
main.tcl	Generate the .4gl associated to a .4prg	4PRG managed program compiler
report.tcl	Generate the .4gl report section associated to a .4rd	4RD Report Data compiler
reportdata.tcl	Generate the .4gl data management section associated to a .4rd	4RD Report Data compiler

Table 3-1. List of TCL Templates (continued)

TCL template name	Usage	Associated Build Rule
ui.tcl	Generate the .4gl User Interface section associated to a .4fdm	4FDM managed form compiler
uidata.tcl	Generate the .4gl data management section associated to a .4fdm	4FDM managed form compiler
zoom.tcl	Generate the .4gl zoom section associated to a .4fdz	4FDZ managed zoom compiler
zoomdata.tcl	Generate the .4gl data management section associated to a .4fdz	4FDZ managed zoom compiler

Template Directory

Templates are located by default in <Studio-install-dir>gst/bin/src/ag/tpl/<template-name>. User-defined templates can be located anywhere.

In addition to the template files (.tpl), additional files that are customizable are stored in the default template directory:

- file-types.xml - defines the file types available in the default template.
- creatables.conf - an xml file used to create those file types.
- settings.agconf - an xml file defining the properties and node types of those file types
- resource folder - additional Genero files that are used in conjunction with the generated application.

Images Directory

Studio searches the <Studio-install-dir>/images/<size> or <template-dir>/images/<size> folders for Images referenced in the templates and associated files. Five copies of the image, of increasing size, should be stored in the S1 through S5 size folders.

Associated Files

These files are used in conjunction with the default dbapp Template.

File Types - file-types.xml

The file types available in the default dbapp template are defined in **file-types.xml** in the template directory.

Example (Managed form)

```
<FileTypes version="1.0">
  <FileType mimeType="application/generostudio-4fdm"
    contentType="application/genero-form"
    extensions=".4fdm"
    description="Genero managed module form"
    actions="FDOpen BAOpenGeneratedSource"
    defaultAction="FDOpen"
    icon="document_4fdm"
    openAction="FDOpen"/>
</FileTypes>
```

Attributes

- **mimeType**: mime type for this file
- **contentType**: file content type is

- **application/genero-form** for Form designer
- **application/genero-br** for business records
- **application/genero-db** for database schemas
- **extensions:** space-separated list of possible extensions for this file type (for example, 4fdm 4fdz)
- **description:** text description for the file
- **actions:** space-separated list of all actions available on the file item (in Project Manager, File Browser, Business Application Diagram, and so on).
- **defaultAction:** actions to be executed by default (on double click)
- **fileNames:** space-separated list of possible filenames for this file type (for example, makefile)
- **icon:** name of the image file to be used for this file type (Project Manager, File Browser); the extension and full path are optional. Studio searches for the file in the <Studio-install-dir>/images/<size> or <template-dir>/images/<size> folders; see Images Directory.
- **openAction:** action to be executed to open the file (right click open ...). This action is mandatory for application generator, it defines the module that will manage the file. (FDOpen -> form designer BROpen -> Business records ...)

See Modifying File Types.

Creatables (creatables.conf)

Entries in the **creatables.conf** file of the template directory are used in the creation of the file types.

- A **Category** corresponds to a group of creatables.
- Each **Creatable** element corresponds to a file type that can be created using the File>>New menu or context menu.

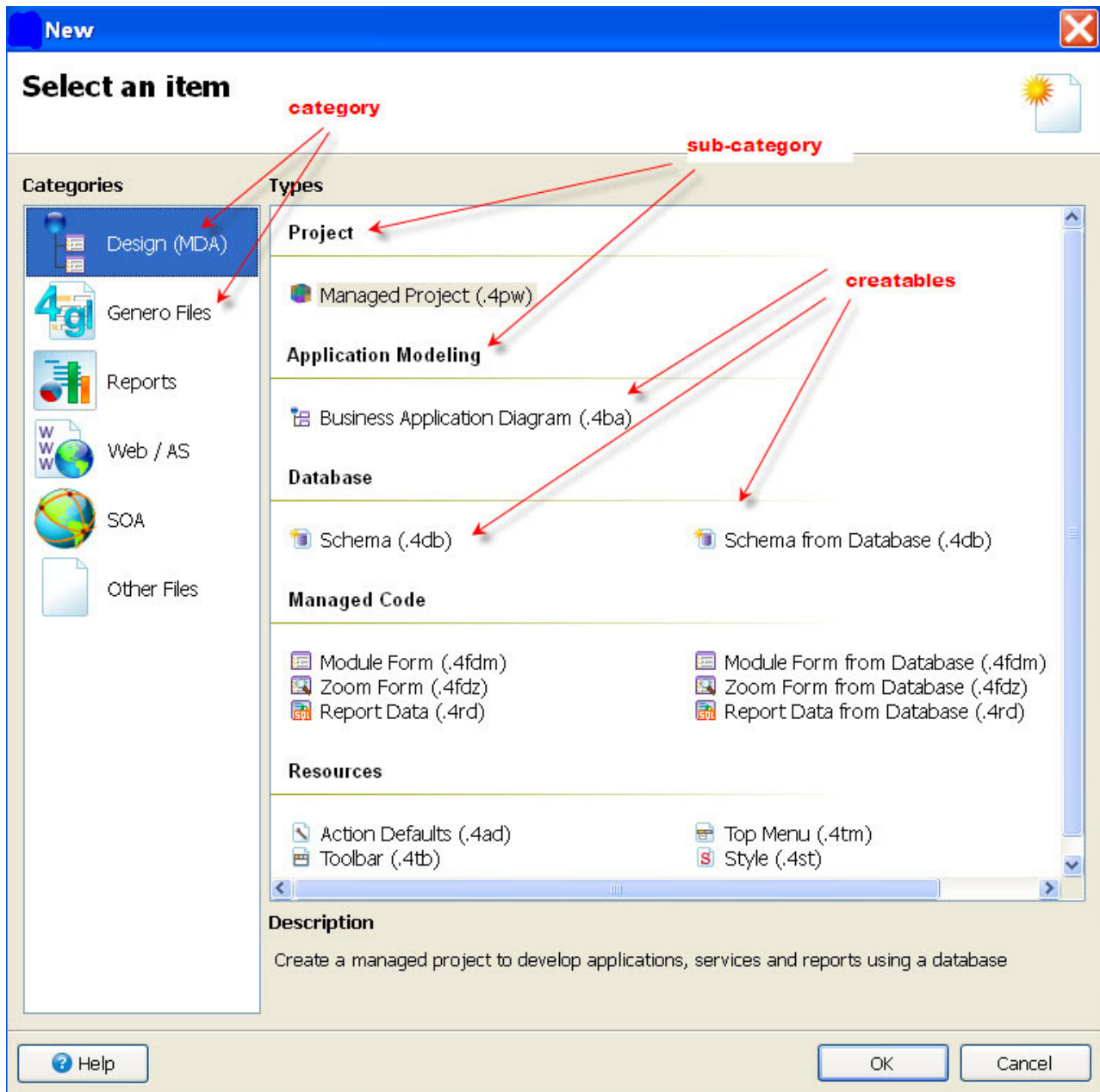


Figure 3-57. New Items: Creatables

Example: Managed Form File Type

```
<Creatables version="1.0">
  <Category index="5" label="Design (MDA)" name="MDA" icon="document_4ba" >
    <Category index="40" label="Managed Code" name="ManagedCode">
      <Creatable index="10"
        name="FDModuleForm"
        action="FDNew"
        label="Module Form from Database (.4fdm)"
        icon="document_4fdm"
        description="Create an empty module in Form Designer"
        extension="4fdm" />
    </Category>
  </Category>
</Creatables>
```

Attributes

- **index:** index order indicates creatable order in the different categories
- **name:** creatable name (internal id)
- **action:** name of the action to be executed when the creatable is clicked
- **label:** creatable label (text)
- **icon:** creatable icon found in the directories `<Studio-install-dir>/images/<size>` or `<template-dir>/images/<size>`; see Images directory
- **description:** text description for the creatable
- **extension:** extension of the file to be created by the creatable (the open action defines the module which will create the file, and defines the file format).

See Modifying File Types

The settings.agconf File

This file defines the file formats for the default File Types referenced in the default dbApp template. For AG files that are modified versions of standard Studio files, only the modifications are listed.

```
<AGSettings version="2">
  <Form/>
  <BusinessRecord/>
  <BusinessApplication/>
  <Database/>
</AGSettings>
```

There are four main sections corresponding to the four file formats that can be customized:

- Form
- Business record
- Business Application Model
- Database

Form

This section describes the default format of the items that have been added to a standard Studio form to create a Module form(.4fdm), and a Zoom form (.4fdz), and their associated Dynamic Properties.

For this format, the items (and corresponding nodeNames) are either Record or Relation.

```
<Form>
  <File extension="4fdm" isManaged="true">
    <DynamicProperties>
      <DynamicProperty name="foreignFields" type="FIELDS" label="foreignFields"
        initialValue="" dynamicContent="srcFieldsContent"
        editorInfo="isDynamic:true" />
      <DynamicProperty name="primaryFields" type="FIELDS" label="primaryFields"
        initialValue="" dynamicContent="dstFieldsContent"
        editorInfo="isDynamic:true" />
      <DynamicPropertyGroup name="relations" label="Relation user properties"
        description="Relation properties group"
        properties="foreignFields;primaryFields" />
    </DynamicProperties>
    <Items>
      <Item nodeName="Relation" srcProperty="foreignFields"
        dstProperty="primaryFields"
        dynamicProperties="primaryFields;foreignFields" />
    </Items>
  </File>
```

```

<File extension="4fdz" isManaged="true">
  <DynamicProperties>
    <DynamicProperty name="foreignFields" type="FIELDS" label="foreignFields"
      initialValue="" dynamicContent="srcFieldsContent"
      editorInfo="isDynamic:true" />
    <DynamicProperty name="primaryFields" type="FIELDS" label="primaryFields"
      initialValue="" dynamicContent="dstFieldsContent"
      editorInfo="isDynamic:true" />
  </DynamicProperties>
  <Items>
    <Item nodeName="Relation" srcProperty="foreignFields"
      dstProperty="primaryFields"
      dynamicProperties="primaryFields;foreignFields" />
  </Items>
</File>
</Form>

```

File element

- Contains DynamicProperties and Items elements.
- isManaged attribute: [optional] values 'true', 'false' (default: false)

DynamicProperty

- **name:** property identifier, name that appears in the xml
- **type:**
 - BOOLEAN: editor is checkbox, values are true, false
 - ENUM: editor is combobox, values are in editorInfo contains: or dynamic
 - INTEGER: editor is spinbox, range is in editorInfo
 - TEXT: plain text property
 - BRQUERY: editor is the record query dialog, fields are dynamic
 - BRFIELDS: edit is the field selection dialog, fields are dynamic
- **label:** text label displayed in the properties view
- **initialValue:** default value used when no user value is set, this is the value set when the restore button is used. The default value is not saved in the file.
- **description:** text description of the property (displayed in property view tooltip)
- **dynamicContent:** name of the dynamic content source if the property is dynamic (see editor info):
 - actionContent: list of actions in the form source of the selected relation
 - dstFieldsContent: list of fields in the destination
 - srcFieldContent: list of fields in the source
- **editorInfo:**
 - alwaysUpdate: if true, the file model is modified immediately on each user action
 - contains: list of values for ENUM type if not dynamic
 - editMode: if true, the property editor is editable (combobo, BRQuery)
 - isDynamic: if true, the property values list is computed internally (ENUM, INTEGER)
 - range: range of integers for the spinbox type
 - readOnly: true or false (sets the property read only)

Items

- **nodeName** - name of the node

- **srcProperty** - name of the property used as the source fields list, if the item is a relation
- **dstProperty** - name of the property used as the destination fields list, if the item is a relation
- **dynamicProperties**- list of dynamic properties that apply to the item node

Business Record

Similar to Form, it defines the settings for a Business Record object, which describes the database columns for a report. The only difference is that the **isManaged** attribute doesn't exist (all Business Record files are managed). In this default format, the only item that differs from the Record associated with a form is a Relation, and the Dynamic Properties apply to that item.

```
<BusinessRecord>
  <File extension="4rd">
    <DynamicProperties>
      <DynamicProperty name="foreignFields" type="FIELDS" label="foreignFields"
        initialValue="" dynamicContent="srcFieldsContent"
        editorInfo="isDynamic:true" />
      <DynamicProperty name="primaryFields" type="FIELDS" label="primaryFields"
        initialValue="" dynamicContent="dstFieldsContent"
        editorInfo="isDynamic:true" />
    </DynamicProperties>
    <Items>
      <Item nodeName="Relation" srcProperty="foreignFields"
        dstProperty="primaryFields"
        dynamicProperties="primaryFields;foreignFields" />
    </Items>
  </File>
</BusinessRecord>
```

Business Application Model

This defines the format of all the objects (items) that a BAM diagram can contain, and their dynamic properties. The items are Program, Module, Zoom, Report Data, Relation, and ReportRelation. This format is similar to the Business Record, except there is no File element.

```
<BusinessApplication>
  <DynamicProperties>
    <DynamicProperty name="action" type="ENUM" initialValue="" label="Action"
      description="Action" editorInfo="editMode:true;isDynamic:true"
      dynamicContent="actionContent"/>
    <DynamicProperty name="dstField" type="FIELDS" initialValue=""
      label="Destination Field" description="Destination Field"
      editorInfo="isDynamic:true;editMode:true" dynamicContent="dstFieldContent"/>
    <DynamicProperty name="srcField" type="FIELDS" initialValue=""
      label="Source Field" description="Source Field"
      editorInfo="isDynamic:true;editMode:true"
      dynamicContent="srcFieldContent"/>
    <DynamicProperty name="openMode" type="ENUM" initialValue="DISPLAY"
      label="Open Mode" description="Open Mode"
      editorInfo="contains:DISPLAY|MODIFY|ADD|SEARCH|EMPTY" />
    <DynamicProperty name="reportPath" type="TEXT" initialValue=""
      label="Report file" description="Report file"
      editorInfo="filters:Report file (*.4rp)" />
    <DynamicProperty name="quickPrint" type="BOOLEAN" label="quickPrint"
      initialValue="1" description="Allow quick printing of the current
      report" />
    <DynamicProperty name="quickPreview" type="BOOLEAN" label="quickPreview"
      initialValue="1" description="Allow quick previewing of the current
      report" />
    <DynamicProperty name="quickPDF" type="BOOLEAN" label="quickPDF"
      initialValue="1"
      description="Allow quick exporting of the current report to PDF format" />
```

```

<DynamicProperty name="quickHTML" type="BOOLEAN" label="quickHTML"
  initialValue="1"
  description="Allow quick exporting of the current report to HTML format" />
<DynamicProperty name="quickXLS" type="BOOLEAN" label="quickXLS"
  initialValue="1"
  description="Allow quick exporting of the current report to XLS (Excel)
  format" />
<DynamicProperty name="quickRTF" type="BOOLEAN" label="quickRTF"
  initialValue="1"
  description="Allow quick exporting of the current report to RTF (MS-Word)
  format" />
<DynamicProperty name="canExport" type="BOOLEAN" label="canExport"
  initialValue="1" description="Allow exporting report" />
<DynamicPropertyGroup name="reportOptions" label="Report Options"
  description="User properties group" properties="quickPrint;quickPreview;
  quickPDF;quickHTML;quickXLS;quickRTF;canExport" />
</DynamicProperties>

<Items>
  <Item name="Program" label="New Program" inherits="Program" extension="4prg"
    icon="bullet_class"/>
  <Item name="Form" label="New CRUD Form" inherits="Module" extension="4fdm"
    icon="document_4fdm" dynamicProperties="quickPrint;quickPreview;quickPDF;
    quickHTML;quickXLS;quickRTF;canExport" />
  <Item name="Zoom" label="New Zoom Form" inherits="Zoom" extension="4fdz"
    icon="document_4fdz" />
  <Item name="ReportData" label="New Report Data" inherits="BusinessRecord"
    extension="4rd" icon="document_4rd_sql" />
    <Relation name="Relation" srcProperty="srcField" dstProperty="dstField"
      dynamicProperties="action;dstField;srcField;openMode"/>
    <Relation name="ReportRelation" srcProperty="srcField"
      dstProperty="dstField"
      dynamicProperties="dstField;srcField;reportPath"/>
  </Items>
</BusinessApplication>

```

Items

Unlike the Form and Business Record formats, the Item elements in this file don't modify the behavior of an existing item, they create new item definitions with new names, inheriting from a default item.

- inherits: item base type
 - File: standard file on disk (file on disk without known format)
 - Program - Program file
 - Module - Form file
 - Zoom - managed Form file
 - BusinessRecord - Records file

Database

This defines the default format for the items that have been added to the Studio meta-schema file. It adds the widget and label properties to the column item.

```

<Database>
  <File extension="4db">
    <DynamicProperties>
      <DynamicProperty name="widget" type="ENUM" label="Widget"
        initialValue="Edit" description="Associated widget"
        editorInfo="contains:ButtonEdit|CheckBox|ComboBox|DateEdit|Edit|FFImage|
        FFLabel|Field|ProgressBar|RadioGroup|Slider|SpinEdit|TextEdit|TimeEdit"/>
      <DynamicProperty name="label" type="TEXT" label="Label" initialValue=""
        description="Associated label when generating form" />
      <DynamicPropertyGroup name="formfieldGroup" label="Formfield"
        description="Formfied properties" properties="label;widget" />
    </DynamicProperties>
  </File>
</Database>

```



```

</DynamicProperties>

<Items>
  <Item nodeName="column" dynamicProperties="widget;label"/>
</Items>
</File>
</Database>

```

Build Rules for Templates

Build rules are defined in the project template of the template directory; the build rules dedicated to a template must not be in the Genero Studio Preferences build rules. The **project template file (4pw)** can be edited with Genero Studio Project Manager.

In the 4pw format, the templates are defined in the <BuildRuleList> element, each one in a <BuildRule> element with the following properties:

- **additionalDependencies** - add the names of other libraries that are to be linked when Project Manager builds a library or application. The library will be recompiled first, if it is not up-to-date.
- **commands** - enter the list of commands to be executed to build the source file. All the available variables can be used in the command.
- **description**- text description
- **enabled** - should be set to "true"
- **fileType**- type of file to which the build rule applies
- **id** - internal id
- **intermediateFiles** - indicates which files are generated during compilation, as intermediate files, before generating the output file. This allows you to view an intermediate .4gl file, for example, and examine or modify the generated code.
- **outputFiles**- enter the list of files that are generated by the Build rule. All the available variables can be used when specifying the path of the file.

Available commands

- \$(fglcomp) [*options*] <*file*>: compiles the given 4gl file locally or remotely depending on the current configuration; see fglcomp
- \$(4fdcomp) <*file*>: compiles the given 4fd file locally or remotely depending on the current configuration
- \$(percomp) <*file*>: compiles the given BDL .per form specification file locally or remotely depending on the current configuration
- \$(agcomp) [*options*] <*file*>: compiles the given AG program, module or zoom file locally or remotely depending on the current configuration
- \$(4dbcomp) <*source.4db*>: builds the .sch schema file from the database meta-schema (.4db)
- \$(agcomp): legacy command for 2.32 templates; compiles AG files.
- \$(blockpoint) [*options*] <*filename*>: extract or inject code between BLOCK/POINT tags. options:
 - -extract extract diff between *filename* and generated part of *filename.codefile*
 - storeGenerated store the generated part of *filename.codefile* with the content of *filename*
 - -inject inject the diff part of *filename.codefile* into *filename*
 - -commentEnd comment end pattern
 - -commentStart comment start pattern

- `$(copy) <sources> <destination>`: copies the source files or directory to a given destination; platform-independent
- `$(delete) <files>`: deletes the given files or directory; platform-independent
- `$(fglmsg) [options] <sourcefile.msg>`: compiles message files into binary version (*outfile.iem*) used by BDL programs; see `fglmsg`
- `$(fglstr) [options] <sourcefile.str>`: compiles localized string files; see `fglstr`
- `$(fglwsdl) [options] <filename | URL>` : calls the `fglwsdl` tool.
- `$(generate) [options] <filename>`: generates an XML file from an AG file.options:
 - `-ba baFilePath` 4ba file path
 - `-depth <depth>` number of visited sources files for XML generation, or "unlimited"
 - `-oldModelFormat` generate XML file with old AG format
 - `-o <outputFilePath>` path of the generated XML file
- `$(move) <sources> <destination>`: moves the source files or directory to a given destination; platform-independent
- `$(tcl) <options> <filename.xml>`: generates the final file using the TCL template and the XML file generated by `$(generate)`

Available variables

- `$(CompilerOptions)` - compilation flags defined in the UI
- `$(InputPath)` - Absolute path of the file represented by this node
- `$(InputDir)` - File Directory
- `$(TargetDir)` - the value of the Target Directory property
- `$(ProjectDir)` - The directory where the 4pw file is located, or the operating system temporary directory if the project has never been saved
- `$(InputName)` - Filename with extension
- `$(InputBaseName)` - Filename without extension
- `$(InputMimeType)` - File mime type
- `$(BAFilePath)` - absolute path of the 4BA file
- `$(TemplateDir)` - AG template directory path
- `$(InstallDir)` - Genero Studio installation directory path

Chapter 4. Project Manager

Topics:

- The Project Manager
- Build Rules and Advanced Properties

Genero Studio Project Manager

Genero Studio Project Manager allows you to define and work with Genero projects.

- Using Project Manager
 - Starting Project Manager
 - The Studio Project
 - The Project Manager Structure View
 - The Properties view
 - The Output view
 - Saving Changes
- Menus of Actions
- gsmake - Command line syntax to Build Projects
- Importing a Project
- Quick Guide to creating and testing a Project

Using Project Manager

From within **Project Manager** you can compile and build projects that contain applications and libraries, and execute and debug the applications.

- A **project** is a *logical view*, a virtual link to the files used to build an **Application** or **Library**. These files can be physically located anywhere on the file system. The projects structure has no connection to the structure of the files on the disk, and you can move and delete files from projects without affecting the actual file on the disk. **Project Manager** does not move or copy your source code files when they are added to a project.
- See the Project Manager view for information about creating projects and project components.
- A Project (file extension **.4pw**) is an internal file used by **Studio** to map the project structure. It stores all the project information used by **Project Manager**. The Project file is automatically updated when you save changes to the project structure in **Project Manager**.

See the Studio Project for information about creating and using Projects.

See Setting Up Studio for information about using local or remote installations of Studio, and sharing projects.

Starting Project Manager

Project Manager automatically starts when you open an existing Project or create a new Project. Use the Studio **File** menu, or choose a Project from the list of existing projects that is presented by **Studio** on startup in the Welcome Page.

A default example project, **HelloWorld.4pw**, is distributed with the **Genero Studio** software.

The Studio Project

A Studio Project stores all the information that you have provided about a project, including:

- the name, type, and location of the source code and executable files
- the database schema, if used
- the name, type and description of each project node
- the parent-child relationships between the nodes

This information is stored in a file (extension **.4pw**) that is used internally by Studio.

Creating a New Project

Use File>>New from the Studio main menu to create a Project (.4pw):

- **Design, Managed Project** creates a project which is already set up for use with automatically generated applications. This project contains additional build rules specifically for the generation of the application.
- **Genero Files, Simple Project** should be used for all other projects.

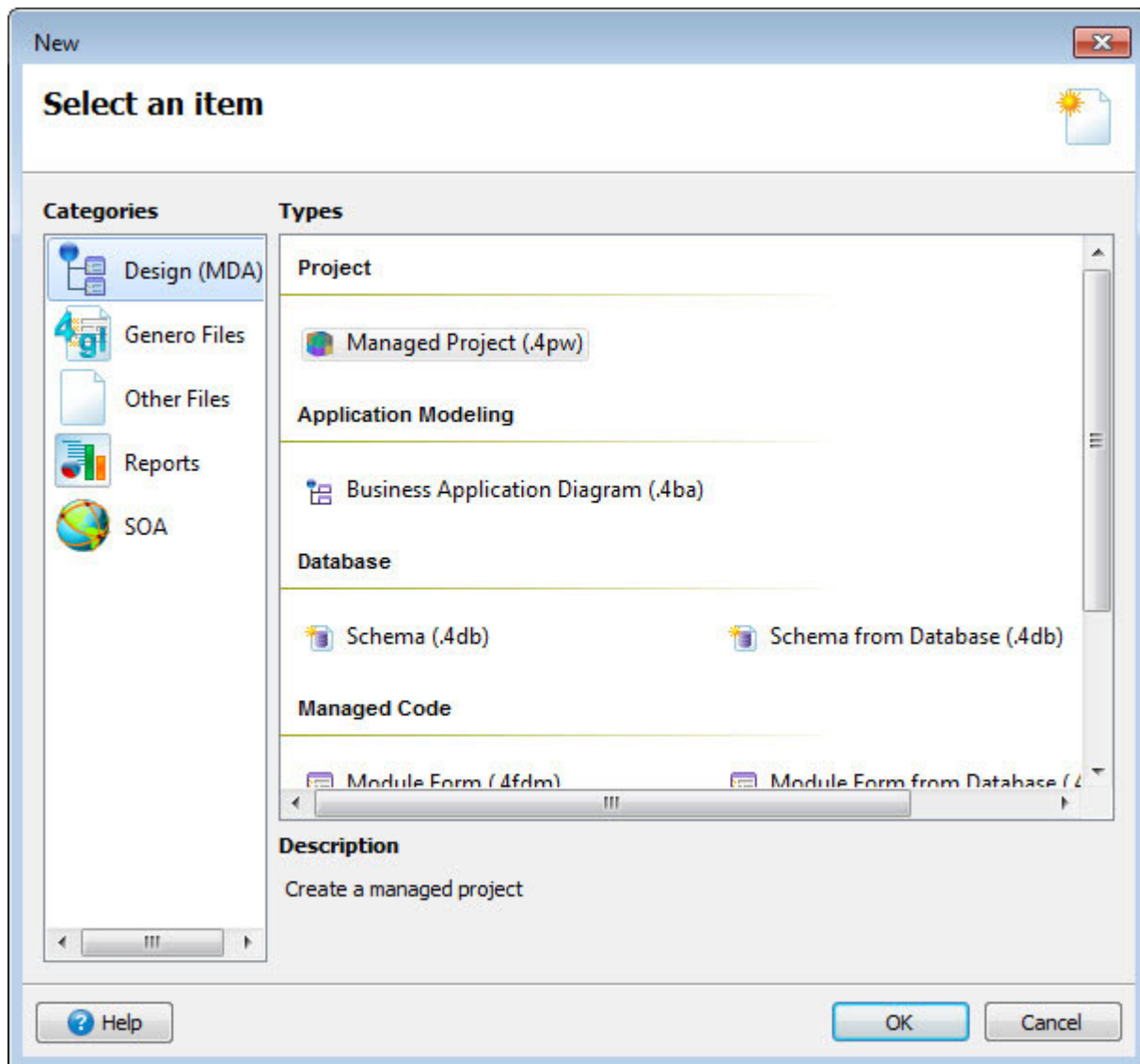


Figure 4-1. New Project dialog

Project Manager is automatically launched.

After adding a Project (.4pw), use File>>Save from the Studio main menu to specify the location of the file in the file system. The Project file can be stored at any convenient place in the file system.

The Project has a default project structure containing nodes for an application, a library, and a databases library.

Using a Project

A Project can contain multiple subprojects.

From within a project you can create new projects, Genero applications, or libraries of Genero files, and compile and build them. Properties can be set for the projects and project components. For example, each file node has the File Path property, and projects, applications, and libraries have the Target Directory property (indicating where the output files from the compile/build process will be stored).

You can compile and build applications and libraries. When an application or library is built, Studio uses the Build rules that are defined for the specific file types. Studio provides default Build rules, and you can create your own global or project build rules. See Defining Build Rules.

You can specify values for Environment Variables to be used by your projects.

You can execute and debug Genero applications, and import a project containing an existing Genero application. You can use the BDL profiler to determine where your Genero program spends its time, to troubleshoot applications that are slow.

See Menu of Actions for a complete list of the actions that are available from the right-click menu, toolbar icons, and Build/Edit main menus.

See Project Manager Structure view for additional information.

Important: Use the **File>>Save** option from the main Studio menu to save any changes that you make in the projects before exiting Project Manager or switching to a different project. See Saving Changes.

The Project Manager Structure View

The structure of your projects is displayed as a tree in the **Project Manager View**.

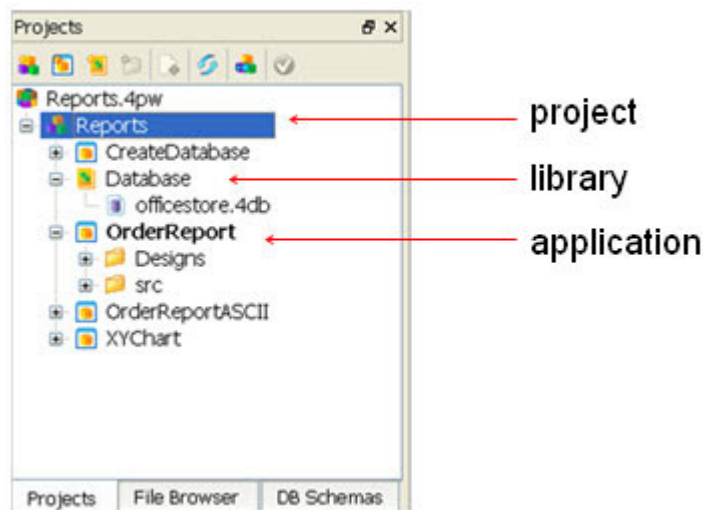


Figure 4-2. Project Manager Structure View

The components of the project are nodes of the tree. Use the toolbar icons to create projects and their components.

- The active project (.4pw) is the root node of the project.
- **Project** - contains the Application and Library nodes that make up the project. Properties defined for the Project node can be inherited by the Application and Library nodes.
- **Application** - will generate an executable program (42r). It can contain both files and virtual folders. Only **ONE** of the files may have a **MAIN** statement: One application node -> one executable. The name of the application node is

used as the name of the **.42r** file, so it must be unique and can only contain characters allowed by the file system. The default application is shown in boldface. Use the toolbar icon to set a different application as the default. The options on the Build menu execute for the default application.

- **Library** - used to group binary files into a single library and generate a library file (**.42x**) It can contain both files and virtual folders. The name of the library node is used as the name of the **.42x** file, so it must be unique and can only contain characters allowed by the file system.
- Libraries should be used when creating a set of features having a common goal, like the logic of an application, a library of mathematical functions, etc. A library can also be used to group other project files together (images, styles or other resources). If a library node contains no 4GL file, no 42x is built.

You can add a library from a different project to your project using the right-click menu option Add external dependencies.

Important: A library must be linked to any application in which it will be used by right-clicking the application node and selecting Advanced Properties, dependencies. In the list that displays, the checkbox for any required library must be checked.

- **Folder** - Folders are virtual folders only, providing a way to group the source files that are under an application or library node. It can contain both files and other folders.
- **File** - is a link to the file in the file system that has the same name as the node. When you add files to an application or library, you are creating a File node. If you rename a File node, you are also renaming the file that is stored on the disk. Project Manager will accept any type of file. When a user tries to open a file, Project Manager will open the corresponding Studio module. If there is no corresponding Studio module, it asks the operating system to open it.

Icon buttons at the top of the view allow you to:

- **Create a new project, library, application, or virtual folder node**
- **Add files** that have already been created to an **application** or **library** node. Or, add the files by dragging and dropping files or folders from the desktop. This adds a link to the given file in the project; it does not physically move the files. All the files must be located on the same drive. Once new files have been added, save the changes using the **File>>Save** main menu option. (To create a new source file use the **File>>New** main menu option instead of Add Files.)
- **Import directory structure as a project**- create a new Project from the contents of a file folder on your disk. Application and/or library nodes will be added, depending on the source files included in the folder. See Importing a Folder as a Project.
- **Set as default application** - only one application can be the default at any time. By default, the first application you create is the default application. The default application node is boldfaced in the project tree. The options on the **Build** menu execute for the default application; use the toolbar icon to set the correct application to the default, if needed, before executing commands from this menu.

Right-click on a project component to display a menu of actions that depend on the context.

Once new project nodes and their components have been added, set their properties in the Properties View.

After adding or making changes to a project, use **File>>Save** from the Studio main menu to save the modifications.

See Quick Guide to Creating and Testing a Project.

Note: The internal format of project files and form definition files has been changed, beginning in version 2.00. The new format is automatically applied when you open an existing project or form in Studio. Make backup copies of the files prior to opening them, if you wish to maintain a copy of the earlier format.

The Properties View

Properties can be set for the projects and project components, to help define the component or specify its behavior. Take particular note of the Target Directory (See Table 4-1), for example, which specifies where compiled modules and linked results will be stored.

The Properties view displays the properties for the project nodes, and their values. Add values in the text box for the property, and use the button on the right to erase existing values:

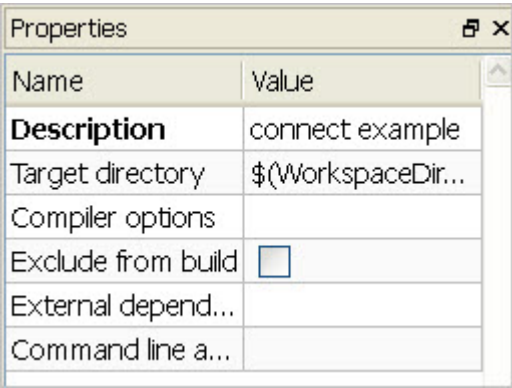


Figure 4-3. Project Manager Properties view

You can specify values for the following properties, depending on the type of project component. Specific Project Manager node variables can be used in the values.

Table 4-1. Project Manager properties

Label	Description	May be inherited	Read-only	Project	Appli-cation	Library	File
Description	A short description of the project node.	NO	NO	YES	YES	YES	

Table 4-1. Project Manager properties (continued)

Label	Description	May be inherited	Read-only	Project	Application	Library	File
Target Directory	Target directory for output files for all applicable nodes. Compiled modules and link results (.42m, .42f, .42r, and .42x files) will be stored in this directory. The default directory is a bin directory created in the current Project directory. Use the Browse button if you wish to change the directory. If you change the directory, Studio tries to maintain a relative path having the project directory as its base. A relative path is not possible, however, if you select a different drive under Windows. For portability, we recommend that Target Directory should always be defined through a relative path, with the Project Directory \$(ProjectDir) as the starting point.	YES	NO	YES	YES	YES	
Compiler options	Options to be passed to the compiler; for example, -S to dump Static SQL messages found in the source.	YES	NO	YES	YES	YES	
Exclude from Build	Excludes the node from the Build process		NO	YES	YES	YES	
External dependencies	External library files that are not a part of the Project but should be included in the linking process, for example, mylib.42x. External libraries are mainly used when you have a library that you use in many projects and that is not intended to be modified, or if you have third party libraries for which you don't have the sources. The directories containing these files must be added to the FGLDPATH environment variable. See Genero variables. To add libraries from the same Project, see Advanced Properties.	NO	NO		YES	YES	

Table 4-1. Project Manager properties (continued)

Label	Description	May be inherited	Read-only	Project	Application	Library	File
Before link command	Displays the Command to execute before linking; provides a hook where you can indicate shell scripts or other programs to be executed. Right-click the Application or Library node and select Advanced Properties to enter values for this property.	NO	NO		YES	YES	
After link command	Displays the Command to execute after linking, provides a hook where you can indicate shell scripts or other programs to be executed. Right-click the Application or Library node and select Advanced Properties to enter values for this property.	NO	NO		YES	YES	
File Path	The complete path of a file, including filename	NO	NO				YES
Before compile command	Displays the Command to execute before the compiler; provides a hook where you can indicate shell scripts or other programs to be executed prior to compiling. Right-click the File node and select Advanced Properties to enter values for this property.	NO	NO				YES
After compile command	Displays the command to execute after the compiler; provides a hook where you can indicate shell scripts or other programs to be executed after compiling. Right-click the File node and select Advanced Properties to enter values for this property.	NO	NO				YES
Command line arguments	Arguments passed to the FGL application when it is launched (Run or Debug). This is useful when your application is written to behave differently depending on arguments that are passed on the command line. For example, the value 123 could be entered as the argument. This value could be retrieved in your application source code using the built-in function ARG_VAL, and the application could be written to respond accordingly.	NO	NO		YES		

The Output view

The Output View contains the following default tabs:

- **Output** - displays messages related to the output and errors specific to the process being performed. The **Filter Messages** checkbox allows you to show only error messages, warning messages, or information messages.
- **Document Errors** - displays errors related to a document.

Check the **Filter messages** box, and click one of the icons at the top of the view to filter the output for specific messages:

- **Show error messages**
- **Show warning messages**
- **Show information messages**

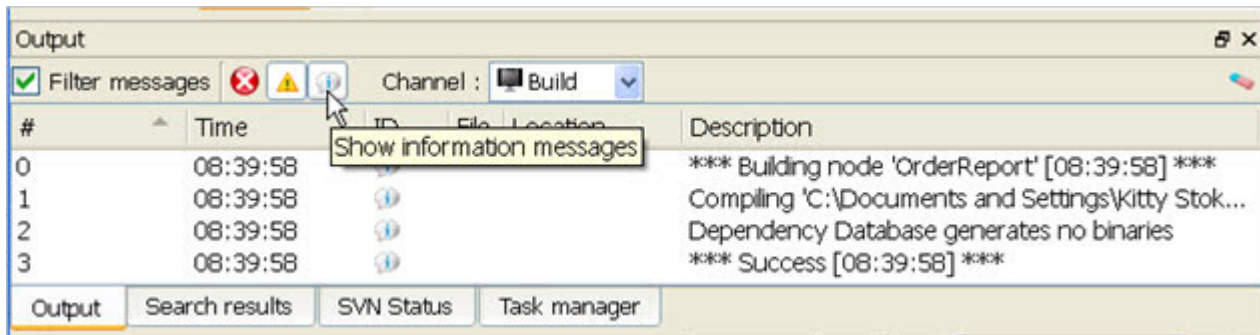


Figure 4-4. Output View

Additional tabs are opened as needed for specific tasks. You can also open specific views using the **Windows>>Views** main menu option. For example:

- **SVN Status** - displays the status of the files in SVN. See Using Version Control.
- **Task Manager** - displays the completion status of current tasks, and allows you to abort a task if desired.
- **Search Results** - displays the matches for Search operations

Saving Changes

Use the **File>>Save**, or **File>>Save All** menu option to save the modified Project Structure after you have added nodes or files.

If a Project is open, when you create a *new* file in Studio the **File Save/Save As** option allows you to save the file to disk and also to specify where a link to the file should be added in the Project Structure:

- **Path:** where the file should be saved in the file system. Click the browse icon to search for the path and enter the filename.
- **Insert the file in the project:** if checked, you can choose the application or library node, or a virtual folder, from the list that is displayed in the dialog. Use the plus icons to expand the list.

Menu of Actions

Select a node in the project and **right-click** to display a menu of actions that depends on the context, or select the corresponding toolbar icon.

You can select multiple nodes using **Ctrl-click**. The desired action will be executed for all the selected nodes; however, the Execute action will be run sequentially. You can not Debug a multiple selection. See options regarding compilation in **Tools>>Preferences>>Compiler and Runtime, Compilation**.

The following actions are also available from the **Studio File** menu:

- **New** - (file, virtual folder, project, application, library nodes) - create a new component depending on context.
- **New file** - create a new file to be added to an application or library node.
- **Open** - open the selected file in Code Editor or Form Designer, depending on the file type.
- **Open with shell** - open the selected file using the open action associated with the file by the operating system.
- **Close** - close a previously opened file
- **Close Project** - close the current Project
- **Delete** - removes the selected node from the project
- **Delete from Disk** - removes the selected node from the project **and** the file system. This can not be undone.
- **Save/Save All** - save the selected file or all files
- **Import form** - import a Genero .per form definition file

Some of the following actions are also available from the **Studio Edit** menu. The actions will be taken for the selected node.

- **Rename** - change the name of a project node. File nodes have the same name as the actual file; if a file node is renamed, the file on the disk is renamed also. Only the name can be changed, not the path. You cannot use the name of an existing node.
- **Cut** - removes the selection from the project. No file system operation is performed.
- **Copy/Paste** - can be used to make a copy of a file in the same file directory, automatically re-naming it and adding it to the project.
- **Delete** - removes the selected node from the project
- **Undo/Redo** - available from the **Edit** menu and toolbar. You can undo any action except a **Delete from Disk**.

The following actions are also available from the **Studio Debug** menu. When run from the right-click menu, the actions will be taken for the selected node. Otherwise, they will be executed for the default application. Unless run from the right-click menu, you must first set an application to default.

- **Execute** - run the application
- **Execute with Profiler** - run the Application using the BDL profiler, a tool in the runtime system that provides a report indicating where your program spends time, and which function calls which function. The report is displayed in the Output view. See Profiler for an explanation of the report sections.
- **Debug** - run the application in debug mode

The following actions are also available from the **Studio Build** menu. From the toolbar icons or the **Build** menu, these actions are executed for the default application, regardless of which application is selected:

- **Build (All)** - build (compile and link) files in the specified node that are not up-to-date, or build all in the Project
- **Clean (All)** - erases all output files (defined in the build rule) created for the specified node, or for the entire project
- **Rebuild (All)** - clean and then build (compile and link) files for the specified node that are not up-to-date, or rebuild all in the project.

- **Compile file** - compile the selected file only
- **Clean** - erase the output file for the selected file only

The command line option `gsmake` is also available to build all in a specified Project file (.4pw)

The following actions are also available from the **Studio Project** menu:

- **New File / Folder / Project / Application / Library nodes** - create a new component in the current project
- **Add files** - add existing source files to the selected Project node.
- **Import project** - import a folder into the current project as a subproject.
- **Edit Build Rules** - customizable build rules that are based on the file extensions. You can add/edit/delete rules for the active project. You can create a rule from scratch, or duplicate a default rule (automatically defined by Studio) or global rule (defined using **Tools>>Preferences.**) Checkboxes for each rule allow you to select the set of rules that apply on each project, application, or library node. Special operands and node variables predefined by Studio can be used in the rules. See Add/Edit Build Rules.

The following actions are available from the **right-click context menu** :

- **Set as default application** - set the selected application as the default. Only one application can be the default at one time. (Also available from a toolbar icon.)
- **Display environment** - displays the values of environment variables in two sections:
 - Values of system environment and current Genero Studio configuration environment variables
 - Values of the local node variables used by Project Manager
- **Advanced Properties** - provides a user-friendly interface to set Application and Library Dependencies, Pre/Post-Link commands, and Environment variables properties. The Dependency property **always** must be set if your project consists of both application and library nodes. See Setting Advanced Properties.
- **Add external dependencies** - this allows you to add a library from a different project to your current project. A new library node (an external dependency library node) is created, which has the name of the 4PW filename. When an external dependency library node is added, all its projects, application and libraries are automatically added as children. The child nodes are in grey because they are in *read-only* mode, they cannot be moved, renamed, etc..., their properties are also *read-only* . It is important to set the dependencies for any application or standard library node that requires the use of this library, using **Advanced Properties**. The new external dependency library node, with all its libraries as children, will be shown in the Dependencies property page; check each library that should be linked with the application or standard library.

If a file is under **Version Control**, additional options from the context menu allow you to commit, update and revert. See Using Version Control.

gsmake - Command line option to Build Projects

The projects in Genero Studio Project (.4pw files) can also be compiled from the operating system command line, using the tool **gsmake**, which is located in the `<Studio-install-dir>/bin` directory.

Syntax

`gsmake [options] <file_list>`

where *file_list* is a list of target Project (.4pw) files, with or without the extension.

Table 4-2. *gsmake arguments*

Argument	Description
-h	Displays help information.
-V	Displays this program name and version.

Table 4-3. *gsmake targets with parameters*

Targets	Parameter	Description
-active		Targets the application set as default in the project.
-all		Targets the complete project (default behavior).
-t	<i>TARGET</i>	Adds <i>TARGET</i> to the list of targets to build. This argument can be used multiple times to build several targets

Table 4-4. *gsmake operations*

Operations	Description
-b	Builds the target (default behavior). The files that are not up-to-date are compiled, the others are not changed.
-r	Rebuilds the target. The output files are deleted, then all files are compiled.
-c	Cleans the target. The output files are deleted.
-force-build	Forces the build/rebuild of the target. The files are compiled, whether or not they are up-to-date.

Table 4-5. *gsmake options with parameters*

Options	Parameter	Description
-j	<i>NB</i>	Sets the number of parallel jobs to <i>NB</i> (default: 1). Set <i>NB</i> to 0 to use the local computer's number of CPUs. When using this option, <i>gsmake</i> will try to start multiple compilations in parallel when possible. This should speed up the global compilation time.
-encoding	<i>ENCODING</i>	Sets the encoding to <i>ENCODING</i> (default: System encoding)

Table 4-5. *gsmake options with parameters (continued)*

Options	Parameter	Description
-disable-dependencies		Disables the computation of the dependencies database (default: false)
-ag-templateDir	<i>DIRECTORY</i>	Sets the Application Generator template directory to <i>DIRECTORY</i> . Enter an absolute path, or a path relative to the <i><gst-installdir>/bin/src/ag/tpl</i> directory. The default value, "default" corresponds to the multiple-dialog template directory.
-max-errors	<i>NB</i>	Sets the maximum number of erroneous files to <i>NB</i> (default: 5) . Set the value of <i>NB</i> to 0 for an unlimited number of error files.
-generate-4pwndb		generate the 4pwndb file only (default is False).
-wcDir	<i>WEBCOMPONENTS_DIR</i>	Web Components Directory path

Importing a Project

The files in a folder on your system can be imported as a new project using the **Project>>Import project** menu option. Lists of files to be included and excluded allow you to specify which files you wish to import. An icon is provide at the top of the Project view to **Import directory structure as a new project**.

Steps

- If the source files to be imported contain reference fields from a database table, create or import a Genero database schema file for the database, if you have not already done so. Add the schema file to the application node.
- To make sure that only sources appear in the project, clean all binaries and generated files from the folder to be imported. Or, set the import filters to exclude them. See Include/exclude files.
- If your sources contain files that are preprocessed to generate other files, set up the appropriate Build Rules. (Example: a file with an extension of .5gl generates a .4gl and then builds it: Add a build rule .5gl → .42m).
- Once the source files are imported into the Project, the import process will automatically fill the required information for any libraries that must be linked in the same project. Right-click the **Application** node and select Advanced Properties, Dependencies.
- Set the other properties for the nodes in the project, as needed. For example:
 - Set Target Directory property for the Project or Application nodes, if you wish to change the default location of the output files when the program is built.
 - If there are external libraries (outside the Project) that should be linked when the application is built, add the library to the value of the external dependencies property for the application node.

- Build the Project to compile and link the files, checking for any dependency errors.

Import Project dialog

- **Project Path:** Enter the path and name of the directory to be imported, or use the browse button to select it from your file system
- **Include files:** A complete list of the files that will be imported (for example, *.4gl indicates all files with the extension .4gl). By default, files having the Genero and Genero Studio file extensions are listed; you can modify this list by adding or removing entries. The entries in the lists are separated by spaces.
- **Exclude files:** Enter the files that are to be excluded from the project. You can use the * symbol to indicate all files; for example, *.abc would exclude all files having the extension .abc. The entries are separated by spaces.
- **Exclude directories:** A list of excluded directories. By default the CVS directory is excluded. The entries are separated by spaces.

In the following example, all the .4gl files in the specified path except for **testform.4gl** will be included. The directories **CVS** and **mydir** will be excluded.

Include files :	*.4gl *.per *.4fd *.4ad *.4sm *.4st *.4tb *.4tm *.att *.msg *.prf *.str *.xcf *.val *.sch
Exclude files :	testform.4gl
Exclude directories :	CVS mydir

Figure 4-5. Import Project dialog

The new Project will be created and listed in the Project Structure view. For each folder, the following process applies:

1. A File node is created for each file.
2. The file nodes are then added to a Library; the name of the folder is used as the name of the Library node.
3. If a file contains a MAIN program block, an Application node is created and the File node is added.
4. A Project node is then created, and all Library and Application nodes, and their nodes, are added to this Project.

Quick Guide to Creating and Testing a Project

1. First, add the project to the Project Manager Structure View using the toolbar icon, and set the properties as needed.
 - We recommend that you set the Target Directory property to a relative path starting from ProjectDir/bin (The bin subdirectory of the Project directory).
 - Periodically during the creation process, use **File>>Save** or **File>>Save All** to save the project and all its changes.
2. Select the **project node**, and right-click to add the new application node. Set the Target Directory and other properties as needed.
3. Select the application node, and right-click to **Add Files** - add the following types of files to this node.

- **Studio database schema file** (.4db) used by the application's source files. Use the **Database** main menu option to create or import Database Schema files for any databases that your project will reference. See Genero Meta-Schema Manager.
- **existing program source code files** (see the Genero BDL Tutorial for example programs),
- **Genero** form definition files

Or, right-click and select **New File** to create program files and form definitions. Save the files in the filesystem; a virtual link for the file to the application node is automatically created. The value of the File Path property is automatically added by Studio.

Note: an application node can have only one BDL file containing a MAIN program block.

Add any values to the Before/After compile commands property as needed.

4. If you have additional Genero or program files that will be used by this project, select the **project node** and right-click to add a library node. Set the properties as needed. Add files to the library node as in step 3. These files will be compiled, any .4gl files will be built into a Genero library having an extension of **.42x**.
5. If you have a library node in this project that should be linked to an application when it is built, right-click the **application node** and select **Advanced Properties**. Choose **Dependencies**, and add a checkmark to the listing for the desired library.
6. If there are external libraries (outside the Project) that should be linked when the application is built, add the library to the value of the External dependencies property for the application node.
7. Right-click the project or application node, and choose **Build** to compile and link the files into an executable program. Check the output of the Build in the Output view.
8. If the Build is successful, right-click the **application node** and select **Execute** to run the program through Studio. If there are runtime errors, right-click and select **Debug** to run the program using the Graphical Debugger.

The compiled and executable files for the program are stored in the Target Directory that you specified. See Deploying Applications in the BDL manual for information about deployment.

Build Rules and Advanced Properties

- Build Rules
 - Categories
 - How Build Rules are Applied
 - Add/Edit Custom Build Rules
- Advanced Properties
 - Dependencies
 - Pre/post link commands
 - Pre/post compile commands
 - Environment variables
- Setting Environment Variables

- The Environment Variable dialog
- How Environment Variables Work
- Predefined Node Variables

Build Rules

Build rules are used to define how Project Manager should compile source files. You can modify or create your own Global or Project Build Rules.

Build Rules are grouped in the following categories:

- **Default** - rules defined by Genero Studio that cannot be modified
- **Global** - rules defined by the user, which will apply to all compilations. To modify the rules list, select the **Tools>>Preferences** menu option, and then select **Compiler and Runtime, Build Rules**.
- **Project** - rules defined by the user, which will apply only to the source files in a particular project. To modify the rules list, right-click in the Project view, and select **Edit Build Rules**.

Only one Build rule can be active for a specific file type. Project rules have priority over Global and Default Rules. Global rules have priority over Default rules.

How Build Rules are applied

Let's look at how Project Manager would compile **myfile.4gl**:

- Project Manager first determines the MIME type of this file; it is "application/genero-4gl".
- Then, we look for a corresponding Build rule in the Project Build rules. If no rule is found that handles the "application/genero-4gl" MIME type, we look in the Global Build rules. If there is still no rule for "application/genero-4gl", we look in the Default rules, and there we find one
- Finally, we execute the commands defined in the Build rules after having replaced the variables.

If no Build rule is found in the Project/Global/Default categories, the file is just skipped and the next one is processed.

Add/Edit Custom Build Rules dialog

Build Rules List

- **Toolbar icons**
 - **Add a Build rule** - select the *Global* category (**Preferences only**) or *Local* category (**Project Manager only**) and click this icon to add a custom rule.
 - **Duplicate a selected Build rule** - select an existing rule and click this icon; the Build rule will be added to the appropriate category. Select the newly added rule and click the **Edit** icon to modify the duplicated rule.
 - **Delete a selected Build rule** - select an existing rule and click this icon to remove it
 - **Edit a selected Build rule** - select a custom Build rule and click this icon to modify the Build Rule fields
- **Name** - enter a label to be assigned to the rule
- **File Type** - defines the MIME-type of the source files that can be compiled with this Build rule; click the icon on the right to display the File Type Selection dialog.

File Type Selection dialog

Select the file type from a list generated from the File Associations in **Genero Studio Preferences**. Use the Search field to limit the display of the list; enter the file extension, or the words Genero or studio, for example, to display only the corresponding file types. The icons can be used to display the list by category, and to expand or contract a category.

Build Rules fields

- **Command line** - enter the list of commands to be executed to build the source file. All the predefined file node variables can be used in the command.

Some *special operands* can be used in the command:

\$(move) - Moves the given file or directory to the given destination in a platform independent way.

`$(move) sourceFilePath destinationFilePath`

\$(copy) - Copies the given file or directory to the given destination in a platform independent way.

`$(copy) sourceFilePath destinationFilePath`

\$(delete) - Removes the given files or directories in a platform independent way.

`$(delete) filePath1 filePath2 ...`

\$(4dbcomp) - Builds the schema file (sch) from the database file (4db).

`$(4dbcomp) sourceFile[.4db]`

\$(4fdcomp) - Builds the compiled form file (42f) from the form file (4fd).

`$(4fdcomp) sourceFile[.4fd]`

\$(percomp) - The fglform tool compiles form specification files into XML formatted files used by the programs.

`$(percomp) [ options] sourceFile[.per]`

\$(fglcomp) - The fglcomp tool compiles BDL program sources files into a p-code version.

`$(fglcomp) [ options] sourceFile[.4gl]`

\$(fglmkmsg) - The fglmkmsg tool compiles message files into a binary version used by the BDL programs.

`$(fglmkmsg) [ options] sourceFile[.msg] [outFile.iem]`

\$(fglmkstr) - The fglmkstr tool compiles localized string files.

`$(fglmkstr) [ options] sourceFile[.str]`

\$(fglwsdl) - Calls the fglwsdl tool for creating a web services program.

`$(fglwsdl) [options] <filename | url>`

\$(generate) - Generate XML file from an Application Generator file.

`$(generate) [options] filename`

Table 4-6. generate build rule options

Option	Description
<code>-ba baFilePath</code>	.4ba file path.
<code>-depth <depth></code>	Number of visited source files for xml generation or "unlimited".
<code>-oldModelFormat</code>	Generate xml file with prior AG format.
<code>-o <outputFilePath></code>	Path of the generated xml file.

\$(blockpoint) - Extract or inject code between BLOCK/POINT tags.

`$(blockpoint) [options] filename`

Table 4-7. blockpoint build rule options

Option	Description
<code>-extract</code>	Extract diff between filename and generated part of "filename.code" file.
<code>-storeGenerated</code>	Store the generated part of "filename.code" with the content of filename.
<code>-inject</code>	Inject the diff part of "filename.code" in the filename.
<code>-commentStart</code>	Comment start pattern.
<code>-commentEnd</code>	Comment end pattern.

\$(tcl) - Generate the final xml using the TCL template and the XML file generated by `$(generate)` command.

`$(tcl) [options] filename.xml`

Table 4-8. tcl build rule options

Option	Description
<code>-tpl <file.tcl></code>	TCL template file path.
<code>-o <outputFilePath></code>	Path of the generated code.

Example

If you have your own preprocessor for your source files (*.my), called **mypp**, the Build Rule for the mime type "**text/my**" would be:

```
mypp $(InputPath) -o $(TargetDir)/$(InputBaseName).4gl $(fglcomp)
$(TargetDir)/$(InputBaseName).4gl
$(delete) $(TargetDir)/$(InputBaseName).4gl
```

- **Output files** - enter the list of files that are generated by the Build rule. All the predefined file node variables can be used when specifying the path of the file.
- **Intermediate files** - indicates which files are generated during compilation, as intermediate files, before generating the output file. This allows you to view an intermediate .4gl file, for example, and examine or modify the generated code.
- **Additional dependencies** - add the names of other libraries that are to be linked when Project Manager builds a library or application. The library will be recompiled first, if it is not up-to-date.

Confirm or cancel your changes.

Advanced Properties

Right-click a Project, Library, Application, or File node and choose **Advanced Properties** from the menu of options to display a dialog for entering the values of specific advanced properties. Once set, the values of some properties are displayed in corresponding fields in the Property view of the Project Manager main window.

Advanced properties dialog

Select a property from the list to set its values:

- **Dependencies (for Application, Library nodes)** - A list of the available libraries from the current Project is displayed. Any library that must be included in the build of the node should be checked.

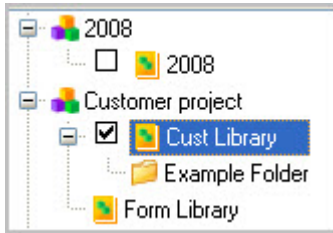


Figure 4-6. List of libraries

- **Pre/Post link command (for Application, Library nodes)** - Shell scripts or other programs to be executed before/after linking. Enter the command in the appropriate text box. Enter the complete path of the script or program, if necessary. For example, if your script **myscript** is in **/home/user/scripts**, and this directory is included in the **PATH** environment variable of your system, enter **myscript**. Otherwise, enter **/home/user/scripts/myscript**.
- **Pre/post compile command (for File nodes)** - Shell scripts or other programs to be executed before/after compiling. Enter the command in the appropriate text box, providing the complete path to the script or program if the path is not part of the **PATH** environment variable of your system.
- **Environment variables (for Project, Application, Library nodes)** - You can define a variable or redefine existing variables. For example, if you have several applications that use different fglprofile files you can set a specific FGLPROFILE environment variable for each one. Predefined node variables can be used in the value.

Genero variables - displays a list of environment variable settings that are automatically computed by Genero Studio, and cannot be altered, although they can be redefined in Project variables.

- **FGLLDPATH**: contains the target directory of the current node and the target directories of all dependencies; path of all the .42x and .42m files used to link/run an application.
- **FGLDBPATH**: path to the database schema file. In Studio it is automatically set to the path of the sources like FGLSOURCEPATH.
- **FGLSOURCEPATH**: points to all directories containing source files for a node + its dependencies. This is computed by Project Manager and is only useful within Studio.
- **FGLRESOURCEPATH**: points to all directories of program resource files. This is computed automatically, and replaces the use of DBPATH by Project Manager for Genero BDL versions later than 2.11.06.

You can add a path to one of these variables in the Project Variables section.

User variables - variables defined by the user. The icons at the top of the list allow you to **add** a new environment variable, and to **edit** or **delete** a selected variable. See Setting Environment Variables.

Setting Environment Variables

Environment Variable dialog

When the New/Edit environment variable dialog appears, enter the following:

- **Type:** valid choices are **value**/**directory**/**directory list**/**file**
- **Name:** the name of the variable
- **Value:** value of the variable. When entering the value, if the type is **directory**, **directory list**, or **file**, select the ... button to browse for the correct value. If the value contains a variable name, that name must be prefaced by \$ and enclosed in parenthesis.

For example, to add **the/added/path** to the existing FGLLDPATH variable, for example, select the **add** icon and enter the following:

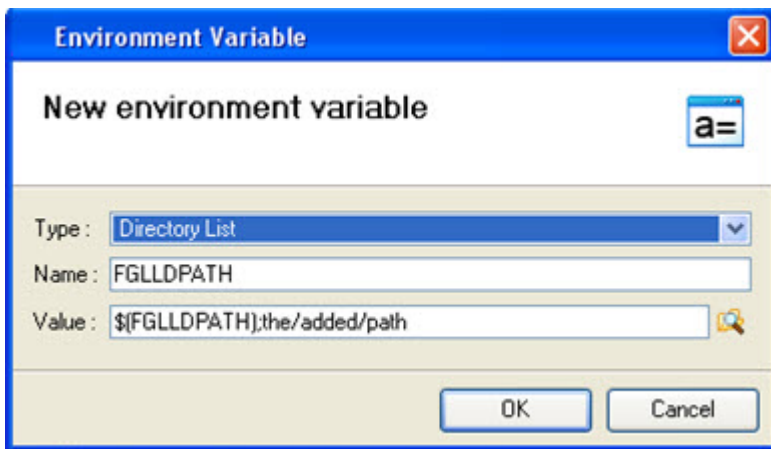


Figure 4-7. New environment variable dialog: Setting the FGLLDPATH

Note:

To define the value for the environment variable FGLPROFILE, for example, which will be a **file**, the entries would be:

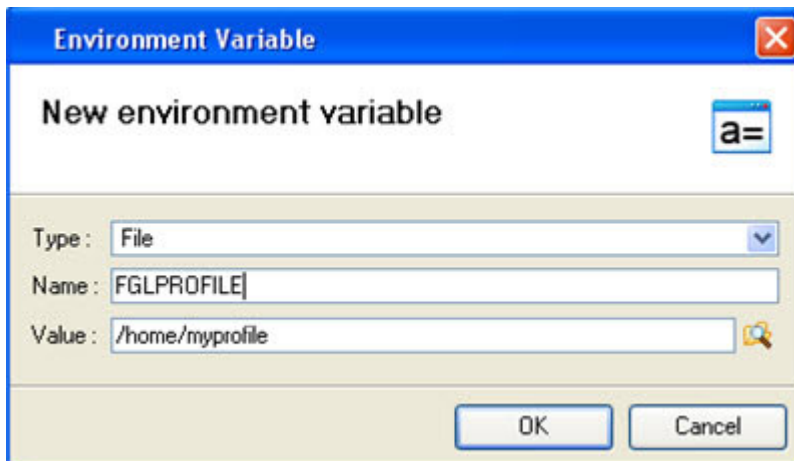


Figure 4-8. New environment variable dialog: Setting the FGLPROFILE

Note: The separator in directory lists is always a semicolon on all systems (Windows and Unix). The directory separator in a path is always a slash (/) on all systems. Respecting these conventions guarantees portability of the project across operating systems.

The value of these variables is the concatenation of the variables defined for this node and the variables defined for its parent or ancestor nodes. A variable defined for a parent or ancestor node can also be reused in definitions for a child node:

Example 1:

The parent node defines **MY_VALUE=hello**.

The child node can reuse it: **MY_COMPLETE_VALUE=\$(MY_VALUE) world**.

The final value of **MY_COMPLETE_VALUE** will be **"hello world"**.

Example 2:

The parent node defines **MY_VALUE=foo**.

The child node can reuse the parent's value and redefine the variable:
MY_VALUE=\$(MY_VALUE) bar.

The final value of **MY_VALUE** will be **"foo bar"**.

Arrow keys allow you to change the order of the variables that you have defined. This affects the way the variables are interpreted when your program is executed. For example:

Table 4-9. Variable interpretation examples

Project level	Variables definition and order	Result when application is executed
Project	V1="hello"	
Application	V2="\$(V1) world"	V1="goodbye"
	V1="goodbye"	V2="hello world"

Table 4-9. Variable interpretation examples (continued)

Project level	Variables definition and order	Result when application is executed
Application	V1="goodbye"	V1="goodbye"
	V2="\$(V1) world"	V2="goodbye world"

Note: If you have selected a foreign language text encoding in Studio preferences, the LANG variable must be set appropriately to correspond to the selected encoding. You can set the variable for a specific environment set in Studio configurations (**Tools>>Configurations**).

How Environment Variables work

The environment used by the compiler and executables launched from Project Manager is made up of the following, in order of priority:

1. The environment defined on the node itself (Genero variables and User variables). This has the highest priority.
2. The environment of the node's parents (ancestors).
3. The environment of the node's dependencies.
4. Environment variables defined in a Genero Studio configuration .
5. System environment variables, which have the lowest priority.

You can *override* System environment variables (5), or environment variables defined in a Genero Studio configuration (4), by redefining them in Project Manager.

For example: PATH=/my/new/path

You can *reuse* the System environment variables or Genero Studio Configuration variables in the Project Manager User Variable definitions.

For example: PATH=\$(PATH);/my/new/path

Use the semicolon to separate directories in a list, and the slash (/) as the separator in a path, for portability of your projects across operating systems.

Calculating the Environment

In this example the program node in the project has a library node as a dependency. The environments have been defined as shown in Table 4-10:

Table 4-10. Environment set for levels used in the example

Level	Environment Set
System environment	PATH=/bin;/usr/bin VAR1=hello VAR2=bonjour VAR3=guten tag
My Library Environment	PATH=\$(PATH);\$(ProjectDir)/scripts VAR1=goodbye
My Program Environment	PATH=\$(PATH);\$(ProjectDir)/scripts VAR1=\$(VAR1) world VAR2=\$(VAR2) \$(VAR1)

If the ProjectDir is /home/joe/project, the environment for the program will be:


```

PATH=/bin:/usr/bin/:/home/joe/project/scripts
VAR1=goodbye world
VAR2=bonjour goodbye world
VAR3=guten tag

```

Predefined Node Variables

Each Project Manager node has a defined set of variables containing values determined at runtime. These variables can be used in Build rules or Environment variables. The variables available on a node are the concatenation of the variables defined for this node plus the ones defined for its ancestors:

Table 4-11. Predefined node variables (*X* indicates available for the node)

Variable	Description	Group	Application	Library	File
\$(ProjectDir)	The directory where the 4pw file is located or the operating system temporary directory if the project has never been saved	X	X	X	X
\$(CompilerOptions)	Compilation flags defined in the user interface	X	X	X	X
\$(TargetDir)	Target directory of the compiled files	X	X	X	X
\$(InputPath)	Absolute path of the input file				X
\$(InputName)	File name with extension				X
\$(InputBaseName)	File name without extension				X
\$(InputExtension)	File extension				X
\$(InputDir)	File directory				X
\$(InputMimeType)	MIME type of the file				X
\$(BAFilePath)	Absolute path of the .4ba file	X	X	X	X
\$(TemplateDir)	Application Generator template directory path	X	X	X	X
\$(InstallDir)	Genero Studio installation directory path	X	X	X	X

Chapter 5. Code Editor

Topics:

- The Code Editor
- Main Window
- Coding Assistance
- Search and Replace
- Search: Wildcards and Regular Expressions
- Print a File
- Keymaps
- vi Commands
- XML Catalogs

Genero Studio Code Editor

Code Editor is a programming-oriented editor. In addition to editing BDL source code, it can handle any kind of text or languages (plain text, .per, HTML, XML, C++, and so on.)

- Introduction
- Edit a File
 - Search for Text
 - Smart Indent
 - Tabs/Smart Tabs
 - Smart Backspace
- Fold Text
- Square Selection
- “Split Windows” on page 5-3
- Encoding (i18n)
- Bookmarks
- BDL Syntax Help
- XML Editing
- Customizing Code Editor

Introduction

Code Editor is automatically launched when you open an existing text file or create a new text file. The file can be opened from the File menu (**New...** or **Open...**); from **File Browser** or **Project Manager** you can right-click the file to open the context menu, or double-click the file. Several files can be opened at the same time; each file will be opened in its own window.

Files are displayed in the Editing window. The **Edit option** on the Studio main menu provides choices for selecting, searching, cursor movement, and other editing features. The available accelerator keys are documented in the KeyMaps page.

The Code Structure displays information about the structure of the file.

The Output view displays messages and errors.

Print a document by using the tool bar icon, the **File->Print** menu option, or **Ctrl+P** shortcut key. The Print dialog opens to show the available print options for your operating system printers.

Genero Studio does not recognize any preprocessor directives and *include* statements in your .4gl files as part of the file structure, unless preprocessor compilation is enabled. From the **Tools>>Preferences** menu under Compiler and Runtime preferences, you can enable preprocessing globally.

Edit a file

The following features are useful when you are entering or modifying the text of a file:

- **Searching for Text** - use the Edit menu options, or the Search/Replace view to customize the search.
- **Smart Indent** - By default, the **Smart Indent** feature is enabled. After you press the Enter key, the new line is indented to align with the immediately preceding non-blank line. For example:

```
01 LET var1 = 55
02 LET var2 = 22
```

If you do not wish your new line to be aligned with the preceding line, use the Left Arrow key to move the cursor towards the left margin of the page.

To disable **Smart Indent** (Behavior and Display Preferences) select **Tools>>Preferences** from the menu or the toolbar icon.

- **Tabs/Smart Tabs** - By default the **Smart Tabs** feature is enabled. When the Tab key is pressed at the beginning of a line, the cursor is aligned with the first character following the next whitespace on the immediately preceding non-blank line. For example:

```
01 DEFINE
02   var1 INTEGER,
03   var2 INTEGER
```

If the preceding line does not have embedded whitespace or Smart Tabs is disabled, the Tab key moves the cursor the number of spaces indicated by the **Tab size** setting in the Behavior and Display Preferences.

To disable **Smart Tabs** (Behavior and Display Preferences) select **Tools>>Preferences** from the menu or the toolbar icon.

- **Smart Backspace** - If enabled, when you press the Backspace key the cursor will move back in the current line to the position of the indent in the previous line.

Fold Text

Folding allows you to condense portions of text based on the scope of statements.

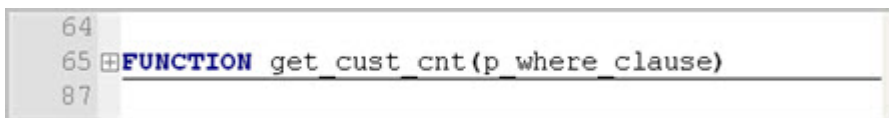


Figure 5-1. Fold text

Click the + or - symbols in the left gutter will fold or unfold all corresponding lines. The **View** main menu option has folding options, which can also be selected from the right-click context menu.

Square Selection

Press the alt key while selecting text with the mouse to select a rectangle of text, instead of entire lines. See Keymaps for additional selection accelerator keys.

Split Windows

Splitting the window allows you to see different parts of a long document at the same time. You can split/unsplit a window using the **View** main menu option, tool bar icons, or shortcut keys. Once a split is requested, the current document window splits into two separate windows which can be scrolled independently. Each window may be split again twice.

Highlight rules are applied to the same text in every window pane. For example, placing the cursor on the IF statement in one window also highlights the same IF statement in other windows in which it is displayed.

Encoding (i18n)

Text Editor supports internationalization (**i18n**), allowing you to enter special characters like Japanese, Chinese, or Indian scripts, and to enter text from right to left as in Arabic. The characters typed at the keyboard are intercepted and changed automatically, based on the Encoding method selected in Genero Studio Preferences. The default Encoding is **Local charset**.

Bookmarks

The **Edit>>Bookmarks** menu option allows you to toggle a bookmark (add/turn on or turn off) a bookmark at the selected location), navigate between existing bookmarks in the document, and remove the selected bookmark or all bookmarks.

BDL Syntax help

Help with BDL syntax is available using the **Edit>>Completion** menu options or the corresponding toolbar icons. Code Templates are available for commonly used bits of code.

XML Editing

Code Editor recognizes XML documents, and provides XML validation. Color coding, Smart Editing, and Code Completion features can be used with an XML file.

Customizing Code Editor

If you want to customize the Code Editor settings for all files, use the **Tools>>Preferences** menu option.

Shortcut keys can also be customized: use **Tools>>Preferences** and expand the listing for **User Interface**; then click Accelerators.

Code Editor Main Window

Topics:

- Document Editing Window
- Code Structure View
- Output View
- Search/Replace View

Document Editing Window

The active document is displayed in the **Editing Window**. Multiple documents can be open, in tabbed windows, with the filename on the tab.

- By default, **Line numbers** are displayed at the left side of the window. To remove the display of line numbers, select **Tools>>Preferences, Code Editor, Behavior and Display** from the menu.
- The **status bar** at the bottom of the window contains the **cursor position** (line, column), the format (Windows, Unix, MAC), and the **mode** (insert/overstrike).
- The Code Structure View displays information about the structure of the active file. Clicking on an element in the Structure view will display and highlight the corresponding lines in the Editing window.

vi Editor

You can execute **VI commands** in the Editing Window. Choose **Edit>>VI Editing mode** from the main menu to open a command line at the top of the window, where you can enter Unix vi editing commands.

You can also set a preference for the vi Editing mode using **Tools>>Preferences, Code Editor**.

Syntax highlighting (.4gl and .per only)

The elements of the program are visually highlighted. Language key words, strings, variables and comments are each colored differently, making the program structure easier to understand. Select **Tools>> Preferences, Code Editor** from the menu or the toolbar icon, to customize the behavior and color.

Syntax errors

An error mark in the gutter flags syntax errors as they occur. Select the error mark to display a message concerning the error. The error message and line number also display in the **Document Errors** tab of the **Output view**.

Menu of options

Use the **Edit menu** from the Studio main menu, or **right-click** in the Editing window, to display some options for selecting, searching, and editing.

The available accelerator keys are documented in the Keymaps page.

Document Format

You can change the document format (Windows, Unix, MAC) using the **Edit>>Convert to** option from the Studio main menu.

Code Structure View

If the file being edited is a **Genero** source code file (**.4gl**), its structure is displayed in a tree:

- **Module listing** - the program blocks and functions, together with their variables
- **SQL** - cursor names
- **Globals** - the global variables, including records
- **Externals** - functions from another **.4gl** file that are called in the active file

Click an object in the tree to display the corresponding lines in the Document window.

The functions listed in the structure are automatically displayed in the order in which they appear in the file. **Right-click** in the **Code Structure view** to display a menu of options:

- **Sort alphabetically/Sort by file order** - switch the order in which the functions, and the variables within the functions, are listed in the view. The default is "file order", the order in which the functions and variables appear within the file.
- **Sort ascending/descending** - specify order

Output View

The **Output view** at the bottom of the page displays all messages.

Errors are listed in the **Document Errors tab**, with the line number and explanation. Right-click the error to highlight the error line in the document window

Search/Replace View

The Search/Replace view allows you to search and/or replace text in one document or a group of documents. Icons at the top of the view allow you to control the search and replace execution. This Studio feature is always available, and the view can be displayed using the **Windows>>Views menu**.

The results of the search are displayed in the Search Results view.

See Search and Replace for additional information.

Coding Assistance

Topics:

- Using Auto-Completion
- Using Code Templates
- Example

Using Auto-Completion (Ctrl space)

Context-sensitive coding assistance is available. Use the **Edit >> Completion** menu option. or the corresponding shortcut key.

Using Code Templates (Ctrl+T)

Code templates are snippets of frequently used code elements that you can insert into your code to avoid repetitive typing. This speeds up the coding process. To use a code template in your code, select one of these methods:

- Type the code template name, such as "case", in your code where you want it to appear; Press **Ctrl+T**, or right-click to display the contextual menu and choose **Expand Template**.
- Position the cursor where you want the code to appear, and press **Ctrl+T** to display a list of code templates. Select from the list using the arrow keys, and press Enter. (Press Esc to cancel)

The selected template is inserted in the document using the exact layout (tabs, linefeeds, and so on.) A corresponding number of lines are shifted.

Templates are user-customizable. Use **Tools>>Preferences**, and select Code Templates to add or modify a template.

Example: Adding a "case" instruction code template

Use the popup window (Ctrl T) to select a keyword, or type "case" and press Ctrl T.

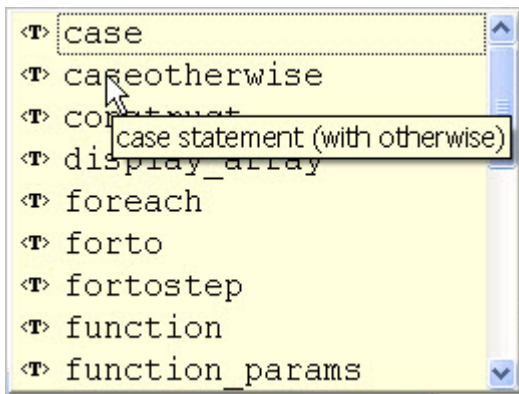


Figure 5-2. Code template pop-up window

The code template is inserted in your code, ready for you to complete.

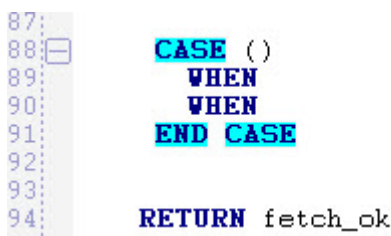


Figure 5-3. CASE code template

Search and Replace

Topics:

- The Search/Replace View
 - Window>>Views menu option
 - Edit menu options
 - Find

- Find in Files
- The Search Results View

The Search/Replace View

A feature that is available through the Studio main menu options, this View allows you to search for and replace text in one or multiple documents.

Using the **Window>>Views>>Search/Replace** menu option opens the Search/Replace View:

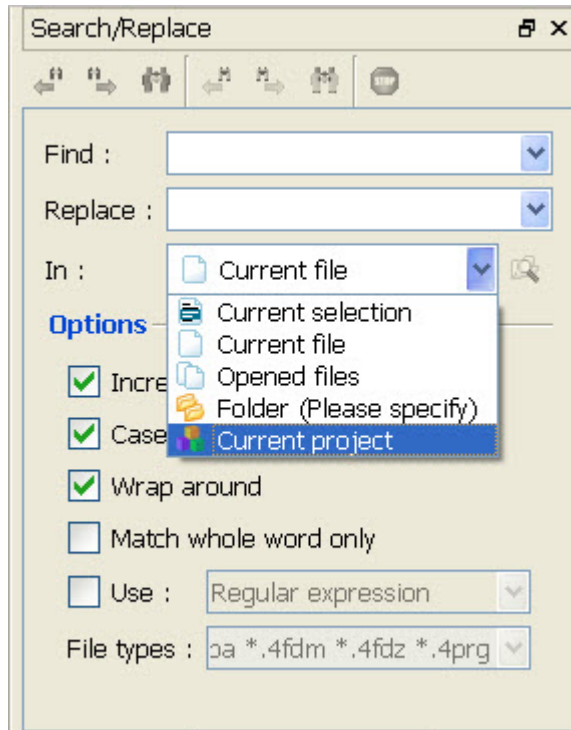


Figure 5-4. Search/Replace View

- **Find:** Enter a search string, or click the down arrow next to the input box to select from a list of previously entered strings.
- **Replace:** Leave this empty, unless you want to replace the text that is found. You can enter the replacement string, or click the down arrow next to the input box to select from a list of previously entered replacement strings. To replace the text with a space, type a space in this box.
- **In:** select the scope of the search/replace. Selecting **Folder** has the same effect as the Find in Files option; you can use the hourglass icon to specify the folder. Selecting **Current project** allows you to search the file content of the project for the search string.

Options:

- **Incremental search** - stops at each occurrence of the string
- **Case Sensitive** - differentiates between uppercase and lowercase when performing the search
- **Wrap around** - when the search reaches the end of the file, wrap around the document to find the next occurrence

- **Match whole word only** - do not find/replace the string if it is contained within another string
- **Text search** - Search file as text, not as formatted Genero file.
- **Use:**
 - Wildcards (* and ?)
 - Regular Expressions

If Use is checked, the Match whole word only option can not be used and is disabled.
- **File types:** the types of files to be searched, specified by **.<extension>*

Icon buttons at the top of the view allow you to control the direction of the search and the implementation of the replace:

- **Find previous**
- **Find next**
- **Find all**
- **Replace previous**
- **Replace next**
- **Replace all**

Use **Esc** to return focus back to the current open file.

The **Edit menu options** allow you to find/replace text in documents.

- **Edit>>Find** (Ctrl + F)
Find/replace text in documents, opens the Search/Replace View.
- **Edit>>Find in Files** (Ctrl+Shift+F)
Find/replace text in multiple documents, opens the Search/Replace View; automatically changes the In combobox value in the Search/Replace View to "Folder". Use the hourglass icon to specify the folder.

The Search Results View

The Search Results are displayed in the **Search Results view**.

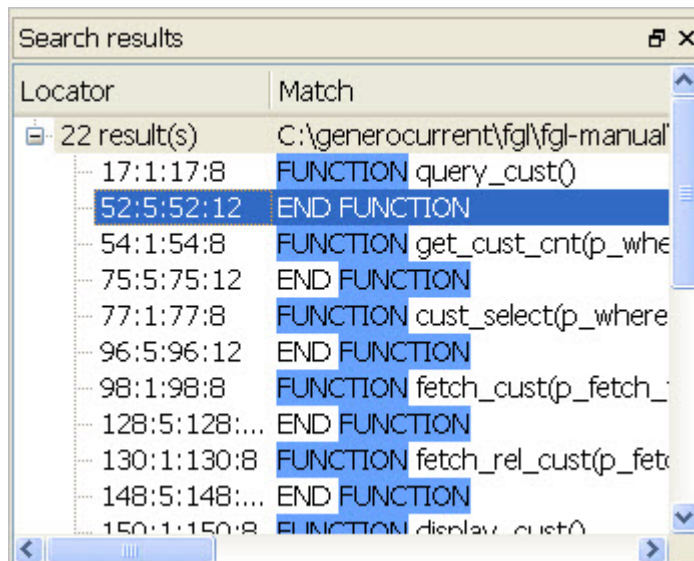


Figure 5-5. Search Results View

The first line in the Locator column shows the number of results found. The first line in the Match column shows the number of matches and the file types where the matches were found. If more than 10,000 results are found, search is stopped and a message is displayed on the summary line.

Double-click the highlighted text to display the string in its corresponding file.

Search: Wildcards and Regular Expressions

Topics:

- Use Wildcards
- Use Regular Expressions
 - Meta Characters
 - Group Capture

Search: Use Wildcards

When you select the **Use Wildcards** option in the Search/Replace view, the search will find the specified combination of characters, including combinations within a word.

Table 5-1. Use Wildcards

Wildcard	Description	Example expressions	Matches any combination of characters
*	Substitutes for any number of characters	1. st*e 2. *rec	1. that begins with st , followed by zero or more characters, followed by e : custrec, store , steady 2. that begins with zero or more characters, followed by rec : gr_custrec, RECORD
?	Substitutes for a single character	• st?re	• that begins with st , followed by a single character, followed by re:store

Search: Use Regular Expressions

You can specify match conditions for a text search using regular expressions, special meta characters, and predefined regular expressions in the string entered in the **Find** box. Otherwise, Code Editor will search for an exact match; for example, if you enter the string *FUNCTION*, Code Editor will find *FUNCTION* or *function*, but not the string *fun*. If the Search option **case sensitive** is checked, the search will distinguish between uppercase and lowercase letters.

Meta Characters

- The meta character `\` matches the character following it, except when followed by a left or right round bracket, a digit 1 to 9, or a left or right angle bracket.
- The special characters "]" and "-" have no special meaning if they appear as the first characters in the set.

Table 5-2. Meta Characters

Wildcard	Description	Example expressions	Matches any string
.	Substitutes for any single character	• fla.	• containing four letters that begin with <i>fla</i> : <i>flag</i> , <i>FLAG</i> , <i>flannel</i>
		• b.g	• containing three letters in the format <i>bxx</i> : <i>big</i> , <i>bog</i> , <i>Bog</i> , <i>bag</i>
*	Substitutes for zero or more occurrences of the preceding expression/character	• a *b (notice the blank before the *)	• "a" followed by zero or more blanks then "b" "a basic" "abasic"
+	Substitutes for one or more occurrences of the preceding expression/character	• a +b (notice the blank before the +)	• "a" followed by one or more blanks then "b" "a basic" "a basic"
\	Searches for the character following; this cancels the special significance of the meta characters including itself, allowing a search for them. When used in a set, it is treated as an ordinary character.	• \+100	• containing +100; treats + as an ordinary character
		• \\user	• containing \user; treats \ as an ordinary character
[set]	Defines a set of characters enclosed in square brackets ([...]) to be used for matching; may define character ranges, as in [a-z] and [0-9]. If the first character in the set is "^", it matches any character NOT in the set.	• [bd]og	• containing <i>bog</i> , <i>dog</i>
		• [Tt]ooltip	• containing <i>Tooltip</i> , <i>tooltip</i>
		• b[^o]g	• containing three characters, <i>b</i> , <any character but <i>o</i> >, <i>g</i> : <i>bag</i> , <i>big</i>
		• [A-Da-d]+	• containing one of the alpha characters <i>a</i> through <i>d</i> inclusive, in uppercase or lowercase: <i>define</i> , <i>Define</i> , <i>age</i>

Table 5-2. Meta Characters (continued)

Wildcard	Description	Example expressions	Matches any string
[set]	Defines a set of characters enclosed in square brackets ([...]) to be used for matching; may define character ranges, as in [a-z] and [0-9]. If the first character in the set is "^", it matches any character NOT in the set.	[A-Da-d]+	containing one of the alpha characters <i>a</i> through <i>d</i> inclusive, in uppercase or lowercase: <i>define</i>, <i>Define</i>, <i>age</i>
x y	Matches either expression x or expression y (composite expression)	bob bog	containing either string: <i>bob</i> or <i>bog</i>
xy	Strings multiple expressions together, finding a single string containing expression x and expression y (composite expression)	def.* iti.*	containing the string <i>def</i>, any characters including none, and then the string <i>iti:definition</i>
^ \$	Restricts the pattern matching to strings at the beginning of the line (^ character) and/or the end of the line (\$ character),	^when - test.\$	- with <i>when</i> at the beginning of the line - with <i>test.</i> at the end of the line.

Group Capture: When the option **Use Regular Expressions** is checked in the Search/Replace view, group capture allows you to isolate groups in the expression to be matched, so they can be captured and substituted during the replacement.

For example, if you have expressions like the following in your document:

```
4+5*6+88
22+4*555
```

You can transform these to the following patterns using group capturing:
(4+5)*(6+88) and (22+4)*555.

In the first expression there are four groups. First you must capture the individual groups in the expression by enclosing them in parenthesis. Specify that the characters are integers using the regular expression [0-9]+ (one or more integers). Use the escape character \ to indicate that the * and + symbols are literal and not meta characters.

Find text: ([0-9]+)\+([0-9]+)*([0-9]+)\+([0-9]+)

Replace expression indicating the desired pattern and the positions of the groups (numbered from left to right): (\1+\2)*(\3+\4)

Result: (4+5)*(6+88)

Print a File

Topics:

- File menu options
 - File>>Print
 - File>>Print Preview
 - File>>Poster Printing Setup
 - File>>Export to Image

File menu options

The Studio File menu offers printing options.

File >> Print

Prints the file using the options available to your operating system printers.

File >> Print Preview

Displays a preview of the printed file in a preview window. For diagrams, scaling and number of pages to use to print can be configured.

File >> Poster Printing Setup

This action is available from Code Analyzer, Report Writer, Business Application Modeling, and Meta Schema Manager to print diagrams. Poster Printing Setup opens a dialog for configuring the scaling and number of pages to use to print.

File >> Export to Image

Exports a diagram to an image file. Export as Image opens a dialog for configuring the image file path, format (such as png and jpg), and size.

Keymaps

Keymap types (and KDE keystrokes) are shown in these topics.

- Cursor movement
- Selection
- Editing Text
- Clipboard
- Search and replace
- Buffers and Files
- Code Completion
- Code Templates
- KDE keystrokes

Note: Genero Studio default accelerator keys may be customized. See Accelerators in **Tools>>Preferences, User Interface**.

Cursor movement keymap

Table 5-3. Cursor movement keymap

Command / Action	Keyboard shortcuts
Left one character	Left arrow
Right one character	Right arrow
Left one word	Ctrl + Left arrow
Right one word	Ctrl + Right arrow
Up one line	Up arrow
Down one line	Down arrow
Beginning of line	Home
End of line	End
Last index of current line	alt + End
First index of current line	alt + Home
Scroll window up one line	Ctrl + Up arrow
Scroll window down one line	Ctrl + Down arrow
Up one screen	Page Up
Down one screen	Page Down
Top of file	Ctrl + Home
Bottom of file	Ctrl + End
Next tab stop	Tab
Previous tab stop	Shift + Tab
Go to line	Ctrl + G
Find matching brace, bracket or parenthesis	Click one brace to highlight matching braces
Smart backspace	Shift + Backspace

Selection keymap

Table 5-4. Selection keymap

Command / Action	Keyboard shortcuts
Select left one character	Shift + Left arrow
Select right one character	Shift + Right arrow
Select current word	Ctrl + L, W
Select to start of current word	Ctrl + Shift + Left arrow
Select to end of current word	Ctrl + Shift + Right arrow
Select current line	Ctrl + L, L
Select to start of line	Shift + Home
Select to end of line	Shift + End
Select up one line	Shift + Up arrow
Select down one line	Shift + Down arrow
Select to top of window	Ctrl + Shift + Page Up
Select to bottom of window	Ctrl + Shift + Page Down
Select up one screen	Shift + Page Up

Table 5-4. Selection keymap (continued)

Command / Action	Keyboard shortcuts
Select down one screen	Shift + Page Down
Select to top of file	Ctrl + Shift + Home
Select to bottom of file	Ctrl + Shift + End
Select all	Ctrl + A
Square selection	alt + mouse (left)
Extend square selection to previous character	alt + Shift + Left arrow
Extend square selection to next character	alt + Shift + Right arrow
Extend square selection to end of current line	alt + Shift + End
Extend square selection to start of current line	alt + Shift + Home

Editing Text keymap

Table 5-5. Editing Text keymap

Command / Action	Keyboard shortcuts
Toggle Insert / Overstrike mode	Insert
Comment (comments all selected lines)	Ctrl + K, Ctrl + K
Block comment	Ctrl + K, Ctrl + C
Unblock comment	Ctrl + K, Ctrl + U
Delete character/selection	Delete
Delete previous character/selection	Backspace
Delete to end of word (kill-word)	Ctrl + D
Delete to start of word (backward-kill-word)	Ctrl + Backspace
Delete line (kill-line)	Ctrl + Del
Delete to end of line	Ctrl + Shift + y
Indent block	Tab
Un-indent block	Shift + Tab
Lowercase the currently selected text	alt + Shift + L
Uppercase the currently selected text	alt + Shift + U
Toggle case of each character in the selected text	Ctrl + U
Lowercase word	Ctrl + alt + W
Uppercase word	Ctrl + alt + Shift + W
Undo	Ctrl + Z alt + Backspace
Redo	Ctrl + Y alt + Shift + Backspace
Insert return	Enter Shift + Enter

Clipboard keymap

Table 5-6. Clipboard keymap

Command / Action	Keyboard shortcuts
Cut selection	Ctrl + X, Shift + Delete, Cut
Copy selection	Ctrl + C, Ctrl + Insert, Copy
Paste from clipboard	Ctrl + V, Shift + Insert, Paste

Search and Replace keymap

Table 5-7. Search and Replace keymap

Command / Action	Keyboard shortcuts
Find	Ctrl + F
Find next	F3
Find previous	Shift + F3
Find in files	Ctrl + Shift + F
Focus back to file	Esc

Buffers and Files keymap

Table 5-8. Buffers and Files keymap

Command / Action	Keyboard shortcuts
File New	Ctrl + N
File Open	Ctrl + O
File Print	Ctrl + P
File Save All	Ctrl + Shift + S
File Save	Ctrl + S
File Save As	Ctrl + alt + S
File Close	Ctrl + F4
Close Genero Studio	Ctrl + Shift + Q
Split window vertically	Ctrl + alt + V
Split window horizontally	Ctrl + alt + H
Unsplit window	Ctrl + alt + Shift + N
Unsplit all	Ctrl + alt + Shift + A
Preview view	alt + Shift + P

Code Completion keymap

Table 5-9. Code Completion keymap

Command / Action	Keyboard shortcuts
Auto Complete	Ctrl + Space
Macro	Ctrl + M

Note: Code Completion is for .4gl files only.

Code Templates keymap

Table 5-10. Code Templates keymap

Command / Action	Keyboard shortcuts
Code Templates	Ctrl + T

KDE keystrokes

KDE uses keystrokes that conflict with some of the keystrokes available in the editor emulations. Please note that the following keystrokes will activate KDE functions, not Code Editor functions, regardless of the Code Editor emulation you use:

- Ctrl + c
- Ctrl + f
- Ctrl + n
- Ctrl + o
- Ctrl + p
- Ctrl + q *
- Ctrl + r
- Ctrl + s
- Ctrl + v
- Ctrl + w
- Ctrl + x
- Ctrl + z
- Ctrl + F1 *
- Ctrl + F2
- Ctrl + F3
- Ctrl + F4
- Ctrl + F5
- Ctrl + F6
- Ctrl + F7
- Ctrl + F8
- Ctrl + Insert
- Ctrl + End
- Ctrl + Home
- Ctrl + Escape *
- Ctrl + alt + Escape *
- alt + F1 *
- alt + F2 *
- alt + F3 *
- alt + F4 *

vi Commands List

These vi commands are currently implemented for Studio Code Editor.

Table 5-11. Cursor movement

Keystrokes	Description
h l k j	character left, right; line up, down
b w	word/token left, right
ge e	end of word/token left, right
0 ^ \$	beginning, first, last character of line
nG/:n ngg	line n, default the last, first
B W	space-separated word left, right
gE E	end of space-separated word left, right
H M L	Top, middle, bottom of screen
g0 gm	beginning, middle of screen line
g^ g\$	first, last character of screen line
fc Fc	next, previous occurrence of character c
tc Tc	before next, previous occurrence of c

Table 5-12. Insertion and Replace (insert mode)

Keystrokes	Description
i a	insert before, after cursor
I A	insert at beginning of first character, end of line
gI (g + Capital I)	insert text in first column/go to first column
o O	open a new line below, above the current line
rc	replace character under cursor with c
R	replace characters starting at the cursor
cm	change text of movement command m
cc or S	change current line
C	change to the end of line

Table 5-13. Deletion

Keystrokes	Description
x X	delete character under (right), before (left) cursor
dm	delete text of movement command m
dd D	delete current line, to the end of line
J gJ	join current line with next, without space
:rd	delete range r lines

Table 5-14. Search and Substitution

Keystrokes	Description
/s← ?s←	search forward, backward for s
n or /←	repeat forward last search
N or ?←	repeat backward last search
# *	search backward, forward for complete word under cursor
g# g*	same, but also find partial matches
gd gD	local, global definition of symbol under cursor
:rs/f /t/x	substitute f by t in range r; x: g -- all occurrences, c -- confirm changes

Table 5-15. Copying

Keystrokes	Description
ym	yank the text of movement command m
yy/:y or Y	yank current line into register
:ry	yank r range of lines
p P	put register after, before cursor position

Table 5-16. Undoing, Repeating and Registers

Keystrokes	Description
u U	undo last command, restore last changed line
.	repeat last changes
.n	repeat last changes with count replaced by n

Table 5-17. Standard Mode Formatting / Filtering

Keystrokes	Description
~	switch case and advance cursor
g~m gum gUm	switch case, lc, uc on movement m

Table 5-18. Insert Mode

Keystrokes	Description
esc or ^]	abandon editing → command mode

Table 5-19. Marks, Motions, and Tags

Keystrokes	Description
mc	mark current position with mark c ∈ [a Z]
'c 'C	go to mark c in current, C in any file

Table 5-20. Misc IO related commands

Keystrokes	Description
:e f	edit file f, reload current file if no f

Table 5-20. Misc IO related commands (continued)

Keystrokes	Description
:rw f	write range r to file f (current file if no f) (:w, :w f)
:q :q!	quit and confirm, quit and discard changes
:wq or :x or ZZ	write to current file and exit
:r f	insert content of file f below cursor
:n	next file
:p	previous file
:n,kw >> file1	append lines n-k into another file file1.

Table 5-21. Ranges

Keystrokes	Description
, ;	separates two line numbers, set to first line
:n,m	lines n to m
n	an absolute line number n
. \$	the current line, the last line in file
% *	entire file, visual area (exclude)
't	position of mark t
/p/ ?p?	the next, previous line where p matches
+n -n	+n, -n to the preceding line number

Table 5-22. Folding

Keystrokes	Description
zo zc zO zC	open, close one fold; recursively
[z]z	move to start, end of current open fold
zj zk	move down/up to start/end of next/previous fold
zm zM	fold more, close all folds
zr zR	fold less, open all folds

Studio XML Catalog Files

Topics:

- Overview
- A Studio XML Catalog File
- Add, Edit, Delete XML Catalog entries in Preferences

Overview

Many XML documents contain external links to stylesheets, schemas, DTDs, and so on, which may be stored on remote systems. If the links are absolute URLs, they only work when your network can reach them. However, the entity resolver of the XML SAX parser can be used to determine whether there is a local equivalent in

your system's **Catalog** application level cache. XML Catalog files are used to provide an alternative path when the external entities aren't accessible, or are not where the XML document specifies.

When you use an XML Catalog file to map these external references to local equivalents, the task of locating the reference is shifted from the XML documents to the XML Catalog files, which can be varied for different audiences. Since local copies of the references are accessed instead of remote network resources, and the local copy can be a subset of the complete schema or DTD in the external reference, XML processing may be faster.

The XML catalog files to be used by Studio are specified through the Genero Studio Preferences.

The XML Catalog file

A subset of the elements specified in the OASIS XML Catalog specification are used in Studio XML Catalog files:

- The **catalog** element to assign a name to the catalog
- **System** and **public** elements to resolve DTDs
- The **uri** element to resolve schema location for XSDs
- The **nextCatalog** element to refer to another catalog file

Example File

The following example illustrates the features of a Genero Studio XML Catalog file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
FOURJS_START_COPYRIGHT(D,2006)
Property of Four Js*
(c) Copyright Four Js 2006, 2011. All Rights Reserved.
* Trademark of Four Js Development Tools Europe Ltd
  in the United States and elsewhere

This file can be modified by licensees according to the
product manual.
FOURJS_END_COPYRIGHT
-->
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  1<public publicId "-//W3C//DTD XHTML 1.1//EN"
    uri="D:/xml catalog/xml files/xhtml11-flat.dtd"/>
  2<system systemId="docbook4.5.dtd"
    uri="D:/xml catalog/docbook/docbook-V4.5/docbook4.5.dtd"/>
  3<system systemId="http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"
    uri="D:/xml catalog/docbook/docbook-V4.5/docbook4.5.dtd"/>
  4<uri name="http://www.4js.com/ns/gas/2.32/cfas.xsd"
    uri="D:/xml catalog/xml files/cfas_1.xsd"/>
  5<nextCatalog catalog="C:/Program Files/FourJs/Genero Studio 102327/gst/confcatalog-gst-defaults.xml"/>
  6<nextCatalog catalog="C:/Program Files/FourJs/Genero Studio 102327/gas/etc/catalog"/>
</catalog>
```

Figure 5-6. Genero Studio XML Catalog file

The **catalog** element is the root of an XML Catalog. The **xmlns** attribute assigns a name to the catalog file.

Additional Elements in this file are:

1. **public** - resolves a Public ID for an external DTD; specifies Public external DTDs intended for broad use (public distribution of a DTD file for a wider audience).
2. **system** - resolves a relative system id for an external DTD; used to find the public DTD if it cannot be located by the public ID.
3. **system** - resolves a web-based system id for an external DTD.
4. **uri** - used to locate XSL and other files. It can be used for everything that is not a declared PUBLIC or SYSTEM identifier for a DTD or system entity file.
5. **nextCatalog** - resolves entries related to Additional Catalogs.

Add, Edit, Delete XML Catalog Entries

Use **Tools>>Preferences** and select **XML Schema/DTD** from the **Code Editor** page in the Pages tree; a list of the XML catalog files to be used with Studio is displayed,

You can add/edit/delete/edit entries for these catalog files through XML Schema/DTD Configuration.

Chapter 6. Form Designer

Topics:

- “Genero Studio Form Designer”
- “Design the Form” on page 6-4
- “Studio Form Containers and Widgets” on page 6-13
- “Database Interface” on page 6-27
- “Properties List” on page 6-30
- “Using Web Components” on page 6-35

Genero Studio Form Designer

Topics:

- Overview
- FormFields
- Action views

See also: Design the Form, Containers and Widgets List, Properties list

Overview

Genero Studio Form Designer is a visual editor that supports the creation, editing, and layout of Genero forms.

Genero applications

A Genero application uses forms to interact with the application's users. Built-in functionality automatically detects when a user has entered data into a form, or selected a button or other form item. Frequently the data displayed on the form has been retrieved from a database table, or will be used to update the table.

A Genero form is described in a **Form definition file**, compiled into an XML file having an extension of **.42f**. The XML file is used by the Genero client to display the application form on the user's desktop.

Genero **Program files** (**.4gl**) use high-level Genero BDL instructions to open and display the form, and to allow the application user to query a database or make changes in the rows of a database table, for example.

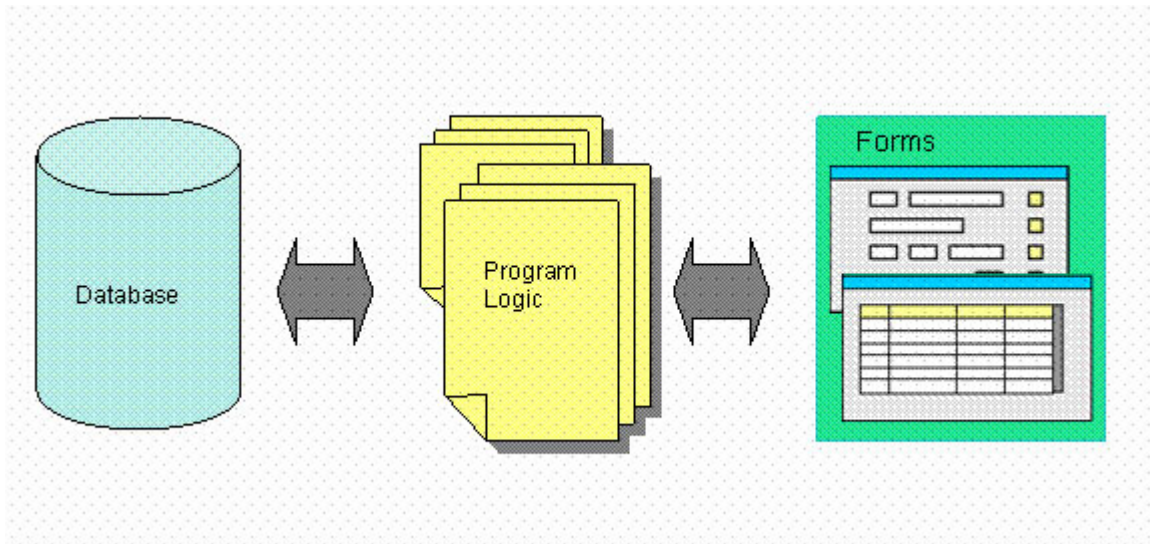


Figure 6-1. Application flow

See the BDL tutorial in the *Genero Business Development Language User Guide* for complete examples of programs.

The Form Designer

You can define a form in the following ways:

- Create a new form by dragging and dropping form items onto the Form Design window. Use **File>>New, Genero Files, Form (.4fd)**; to create a Form that will become part of a generated application, Use **File>>New, Design, CRUD Form (.4fdm)**
- Generate a form based on the columns in an existing database table. Use **File>>New, Genero Files, Form from database (.4fd)**; to create this form as part of a generated application, use **File>>New, Design, CRUD Form from Database (.4fdm)**. See "Database Interface" on page 6-27.
- Import an existing Genero form definition file (.per), converting it to the Genero Studio form definition file format (.4fd). Use **File>>Import Form** to import a Genero .per file that you had previously created. Once the file is imported into Studio Form Designer, you can modify the definition as desired.

Form is a "Managed Form"

When you define a form, a record is created listing the fields for the form, and containing the information to generate the application screen (the form itself).

Some templates are provided for the decoration of the UI, or you can define this by loading a style file, for example, or specifying the preferred widget for database fields in the Database Schema file.

Forms can contain data from one database table, or more complex forms can retrieve/display/update data in multiple tables.

Forms can contain toolbars and menus as well as data field widgets, and the action views on the form are documented.

Form items

Every item on a form must be in a Container. A Grid is the basic container that allows you to group other form items such as widgets and objects, to display a single record of program data, for example. Table containers provide record-list presentation in columns and rows.

Within the containers you can drag-and-drop various Widgets, which can be:

- FormFields that can store data, and facilitate the interaction of the user with the Genero program (Edits, Checkboxes, Radio Buttons or DateEdits, for example)
- action views that trigger actions that are defined in a Genero .agl file (Buttons , Toolbar icons, or Menu items, for example)
- Other items (decoration) that enhance the display of the form or provide information (Labels, for example)

See Containers and Widgets for additional examples.

Properties

Each form and form item has properties that control its appearance and behavior. You can specify values for the properties in the Properties View of the Studio Form Designer window. The available properties vary, depending on the type of form item.

Styles from a Presentation Styles file can be applied to the form and form items.

Structure View

The Form Structure View displays the structure of a form definition file. Right-click the form object in the Structure to display a context menu that allows you to add Toolbar icons, Menu items and Action Defaults to the form.

Once a form definition is created, you can save it to the file system or a project (File>>Save).

The Build Menu

From the Build option on the Studio main menu, you can preview a file, validate it to make sure it conforms to Genero rules, and compile it. You can also compile your form from Project Manager as part of a Build of an entire application.

Formfields

A formfield is a type of form object that can be used to display data or take input. It is presented to the user through a widget; the most commonly used widget, EDIT , defines a simple line edit box that allows the user to enter a value directly into the formfield. Other widgets, such as the COMBOBOX and CHECKBOX, present the data contained in the formfields in a user-friendly way. For example, a ComboBox defines a dropdown box of values, allowing the user to select from a list of valid values for the underlying formfield.

In the Properties View, the fieldtype property of a formfield may be set to specify that its data type is derived from the data types in a database table (TABLE_COLUMN, TABLE_ALIAS, FORM_LIKE), or that it is FORM_ONLY, indicating that the data type is not derived from a database column.

The `sqlType` property indicates the BDL data type. If the data type is to be derived from a database table, the `databaseName` property of the form must be set to a database for which there is a database schema file listed in the **Db Schemas** tab. The `sqlTabName` and `colName` properties of the formfield must be selected from the tables and columns in the schema file. The `dataType` property will be automatically filled based on the data type for the table column in the schema file.

A form has built-in validation to insure that the value entered into a formfield is compatible with the declared `dataType` in the form definition file. Additional validation routines can be specified in your BDL program.

The values entered into formfields are stored in variables in the Genero program, which may be used in the program in any way. A common use is to provide values for SQL statements that update a database.

Action views

Action view items on the form, such as Toolbar icons, Topmenu items, and Buttons, can trigger actions defined in your program code. The value of the `name` property (Toolbars, Topmenus, and Buttons), or the `action` property (ButtonEdits), of the form item in the Properties View must exactly match the name of an action in an `ON ACTION` clause of an interactive statement, such as `MENU`.

For example, a `ButtonEdit` on the form with the `action` property value set to "search" would trigger the action "**search**" in the following BDL program code, when the `MENU` statement is active and the user clicks the button:

```
MENU
  ON ACTION search
    CALL find_customer()
  ...
```

Design the Form

Topics:

- Form Design Window
 - Form tab
 - Records tab
- Handling Errors
- Align/Layout
- Tab Index
- Form Structure View
 - Adding Topmenus, Toolbars, ActionDefault Lists
 - Change the Container for a widget (Convert Container)
- Properties View
- Build (Compile) the Form
 - Build menu options
 - `gsform` - command line syntax
- Using Styles
 - Style File Property
 - Style Property
 - Style Dialog
- Using Localized Strings

See also: “Database Interface” on page 6-27

Form Design Window

You can select items to be placed on the form from the appropriate Widget and Container menus, or from the toolbar icons. All widgets must be within a **container**, usually a Grid, ScrollGrid, Table, or Folder. The Data Control container, or the File>>New, **Genero, Form from Database** menu option, allows you generate a form by specifying the database columns that will make up the form fields.

Note: File>>New, **Design, CRUD Form from Database** allows you to create a CRUD Form (Create, Read, Update, Delete) that will be part of an Application Generator application.

Form tab

This tab displays the form items that you selected in the main window.

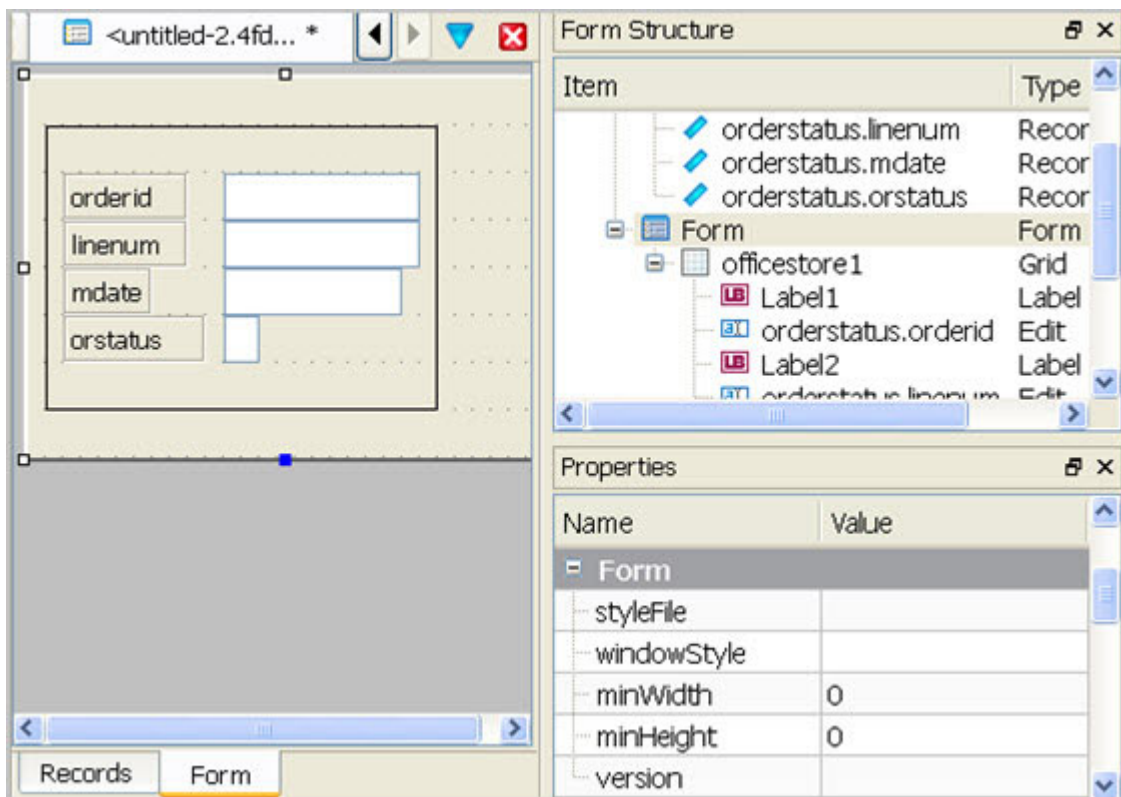


Figure 6-2. Form tab

To position a form item on the container:

- *Single-click* the icon: the item is selected, and one occurrence can be placed on the form.
- *Double-click* the icon: the item remains selected until explicitly deselected by clicking in the main window, another icon, or the select arrow. This allows you to draw several occurrences of the same item without re-selecting it.

Widgets can be:

- **formfields** - (data handling widgets) define an area where the user can view and edit data; the widget can be a simple Edit field, or the widget can represent the underlying field (ComboBox, DateEdit, Checkbox, etc.)
- **action views** - can be used to trigger an action defined in your program code (Button , ButtonEdit, Toolbar icons, TopMenu items)
- **decoration** - for display only (StaticLabel, StaticImage, HLine)

You can specify the appearance or behavior of a form item by setting or modifying the values for its properties in the Properties view.

Records tab

The Records tab displays the screen record for the form, consisting of all the database field names that make up the fields on the form itself.

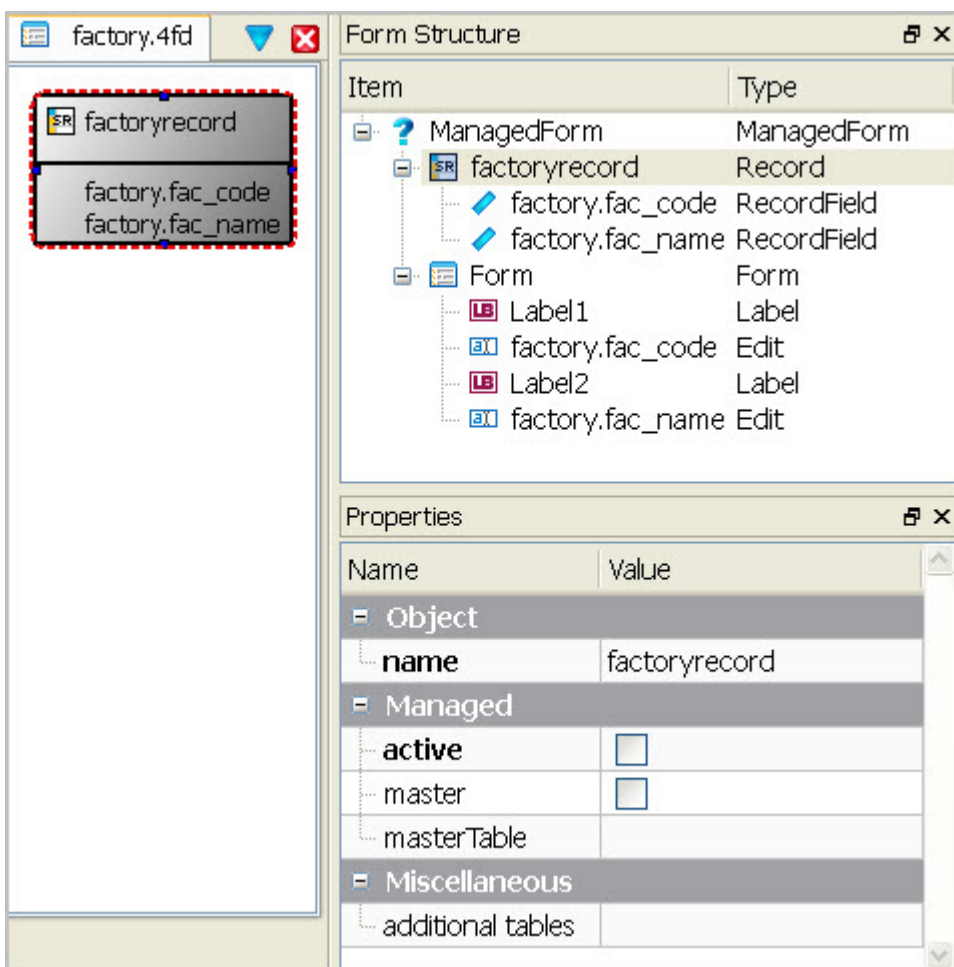


Figure 6-3. Records tab

The structure of the screen record, as well as the structure for the form itself, is displayed in the Central Work area. Properties for the screen record and its items are primarily used in an Application Generator application, see Your First Application .

Handling Errors

Form Designer detects errors in the form design, and displays an error message in the Errors tab in the Output view.

Undo / Redo on the **Edit** menu and the toolbar allow you to reverse all actions taken since the form was opened.

Align/Layout

Parent containers will automatically enlarge when there is not enough space for a child container. The parent container size is limited to the form's maximum width and height.

To align items on the form, use the toolbar icons or the Alignment menu. Only widgets **inside the same container** may be aligned.

- **Left alignment:** selected items are aligned with the leftmost widget selected
- **Right alignment:** selected items are aligned with the rightmost widget selected.
- **Center alignment:** selected items are centered vertically based on the average centering of all selected items
- **Top alignment:** selected items are aligned with the topmost widget selected .
- **Bottom alignment:** selected items are aligned with the bottommost widget selected .
- **Middle alignment:** selected items are centered horizontally to the average centering of all selected items.
- **Advanced:** a dialog box opens to display alignment and distribution options
- **Distribute horizontally:** selected items are aligned with the topmost widget selected (= top alignment) AND items are evenly distributed
- **Distribute vertically:** selected items are aligned with the leftmost widget selected (= left alignment) AND items are evenly distributed

Advanced Alignment dialogs

- **Alignment Type** - varies for Horizontal and Vertical alignment
- **Align with Parent Container** - alignment is based on the parent container's border
- **With/Without Space Reduction** - if enabled, blank lines (vertically) or empty columns (horizontally) are removed
- **User Defined Space Reduction** - user selects the number of lines (vertically) or columns (horizontally) to put between selected items

Layout

If a parent container has more than one child container, the child containers must be grouped and aligned using layouts. Use the toolbar icons or the **Layout** menu to group them:

- **Horizontal Layout** - aligns containers horizontally and draws a red box around the perimeter in the form design window.
- **Vertical Layout** - aligns containers vertically and draws a red box around the perimeter in the form design window.
- **Break Layout** - removes the selected red perimeter box, ungrouping the containers; if you have grouped containers in a layout, you must break the layout in order to move the containers.

The form must be large enough to contain both the child containers in their desired position, as well as the layout container.

First, move the child containers into the desired position on the form. Then select Horizontal or Vertical layout, and draw the layout so that it contains the child containers.

Tab Index

The order of the form items for data entry (when using the Tab key to move from field to field) may be specified using the **Widget>>Tab Index** menu option or clicking the Tabindex icon. All items that allow entries are displayed in the Form Design window with an index number specifying the order:

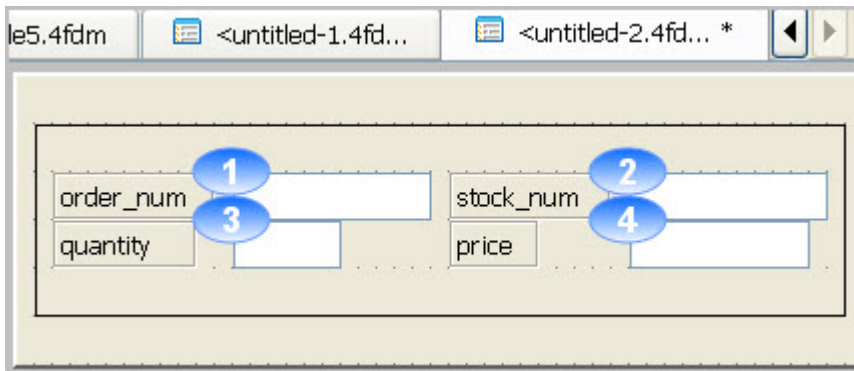


Figure 6-4. Tab Index

To alter the order:

1. **Double-click** a number to change it to 1
2. **Click** any number to change it to the next number after the last used. And so on.
3. To finish the current sequence, Click **ESC**.
4. Click **ESC** again to exit tab order.

Form Structure View

The structure of the form is displayed in the Form Structure view. As you add form items to the Form Design window, the elements are automatically added to a tree structure in the Form Structure tabbed page. Expand the nodes in the tree to display or hide the different form items. For example:

- **Managed Form**
 - **Screen Record** (title is "undefined" until changed when file is saved)
 - list of fields in screen record
 - **Form**
 - list of fields, labels, and other widgets in form

Selecting a form item also selects the corresponding object in the Form Design window, and displays the item's properties in the Properties view.

Right-click on the Form item in the Structure view to display a menu allowing you to add Topmenus, ActionDefault Lists, or Toolbars to the form. You can also apply Styles to the form objects - see Using Styles.

Change a Container for a Widget (Convert Container)

Right-click the container object in the Form Design window, or the container element in the Structure view, to select the menu option **Convert Container**. A dialog window allows you to select the new container.

Properties View

Selecting one item from the design window makes this item current, and its properties are displayed in the **Properties view** and available for updating. Each form item, as well as the form itself, has its own specific properties that can be set.

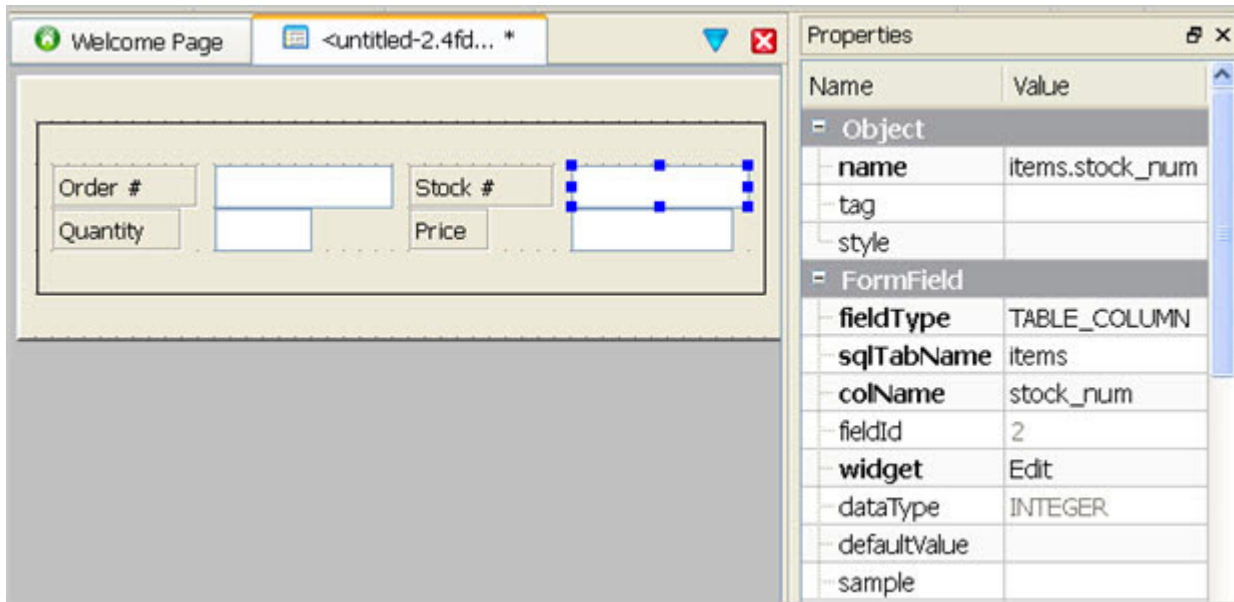


Figure 6-5. Properties view

Multiple properties can be selected using Ctrl-click. If multiple properties have been selected, only the common properties will display in the Properties view.

Build (Compile) the Form

Compiling your form definition translates it into an XML file with an extension of **.42f**. This XML file is used by the Genero client to display the application form on the user's desktop. Compiling a form can be specifically requested from the **Build** menu, or it can be executed from Project Manager as part of a Build of an entire application. A form can also be compiled from the command line using **gsform**.

You can use the **Build** menu or toolbar icons to:

- **Compile form** - create an executable form usable by Genero programs; the file extension will be **.42f**. Errors will be displayed in the Output view.
- **Preview the form** - view the form displayed on your desktop using Genero Desktop Client. The form is automatically validated before the preview process. Only valid forms can be previewed

All **.42f** files must be included with your application code in distributions.

Command-line syntax: gsform

The Studio form definition file (**.4fd**) can also be compiled from the operating system command line, using the tool **gsform**, which is located in the **<Studio-install-dir>/bin** directory. The gsform tool executable file should not be moved to another directory.

Use gsform if you want to:

- compile a .4fd file from the command line
- import a .per file into Studio Form Designer format
- convert .4fd files in the old Studio format to the new .4fd format
- extract localized strings
- keep a copy of the .per file created during compilation

gsform Syntax

```
gsform [options] file1[.4fd] file2[.4fd] ...
```

where *filen* is a Genero Studio form file with or without the extension being specified.

Table 6-1. gsform options

Option	Description
-h	Display help instead of the standard behavior
-m	Extract localized messages, See the <i>Genero Business Development Language User Guide</i> topic Localized Strings for additional information.
-V	Version information
-c	Convert old format .4fd files to new format. Old format <filename>.4fd files are backed up in <filename>.bak, New format files are saved in <filename>.4fd. If the backup fails, the conversion is aborted.
-import	Import .per file as Studio .4fd file
-keep	Keep intermediate .per file that is created; do not delete. The compilation process uses temporary .per files. In case there are unexpected (internal) errors, this option avoids the temporary files deletion so that the file can be read. Important: Form Designer and gsform don't allow compilation when .per files with the same filename as .4fd files are present in the same directory.
-dbname <i>database</i>	Provide the database name <i>database</i> to use for form compilation. If the 4fd files use a database, it is strongly recommended that you provide the database name using this option to reduce computation time.
-i	Ignore existing .per file; overwrites it.

Table 6-1. *gsform options (continued)*

Option	Description
-M	Display all compilation messages. See the <i>Genero Business Development Language User Guide</i> topic Compiling Programs for additional information.
-W	Display warning messages. Option <i>all</i> turns on all messages.

Using Styles

Genero Presentation Styles files define a set of properties that can be used to centralize the appearance of forms and form items.

Style file property

You can use any existing Genero Presentation Styles file for your form. Select the form object in the Form Structure view to display the form's properties:

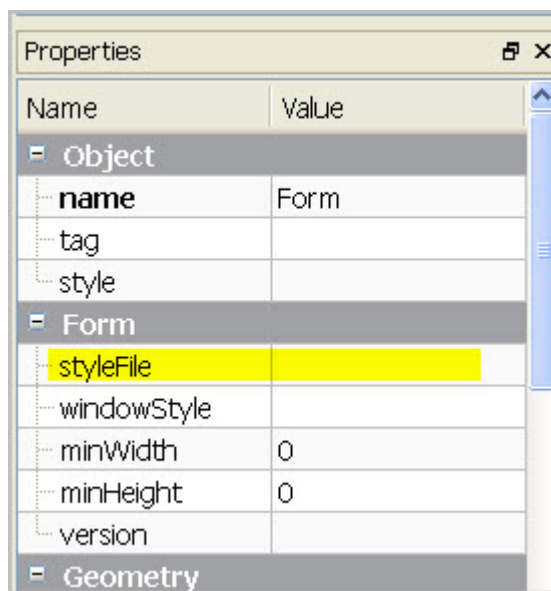


Figure 6-6. *styleFile* property in Properties view

Select the style file property, and click the ... button to browse on your system for the Presentation Styles file that you want to use for your form.

If no Genero style file is specified in the **Style file property**, and a **default.4st** file is found in the path specified by the FGLRESOURCEPATH environment variable, the **default.4st** file is used for form previews and style dialogs. If no **default.4st** file is found in FGLRESOURCEPATH, the **default.4st** file from <FGLDIR>/lib is used.

This provides flexibility in the use of style files. You can:

- Specify the style file in the form file.
- Add a default.4st style file to a project node, and set a dependency to the form application node.

- Put a default.4st in a directory, and add the path to the FGLRESOURCEPATH environment variable of an active environment set.
- Automatically use the <FGLDIR>/lib/default.4st file.

Style property

Once a **Style File** property has been set for the form, the style property is enabled for form items. Select a form item to display its properties. Select the style property, and click the ... button to display the **Style Dialog**, allowing you to select a style for the selected form item.

Style Dialog

A tree of the available styles from the form's Style File is displayed:

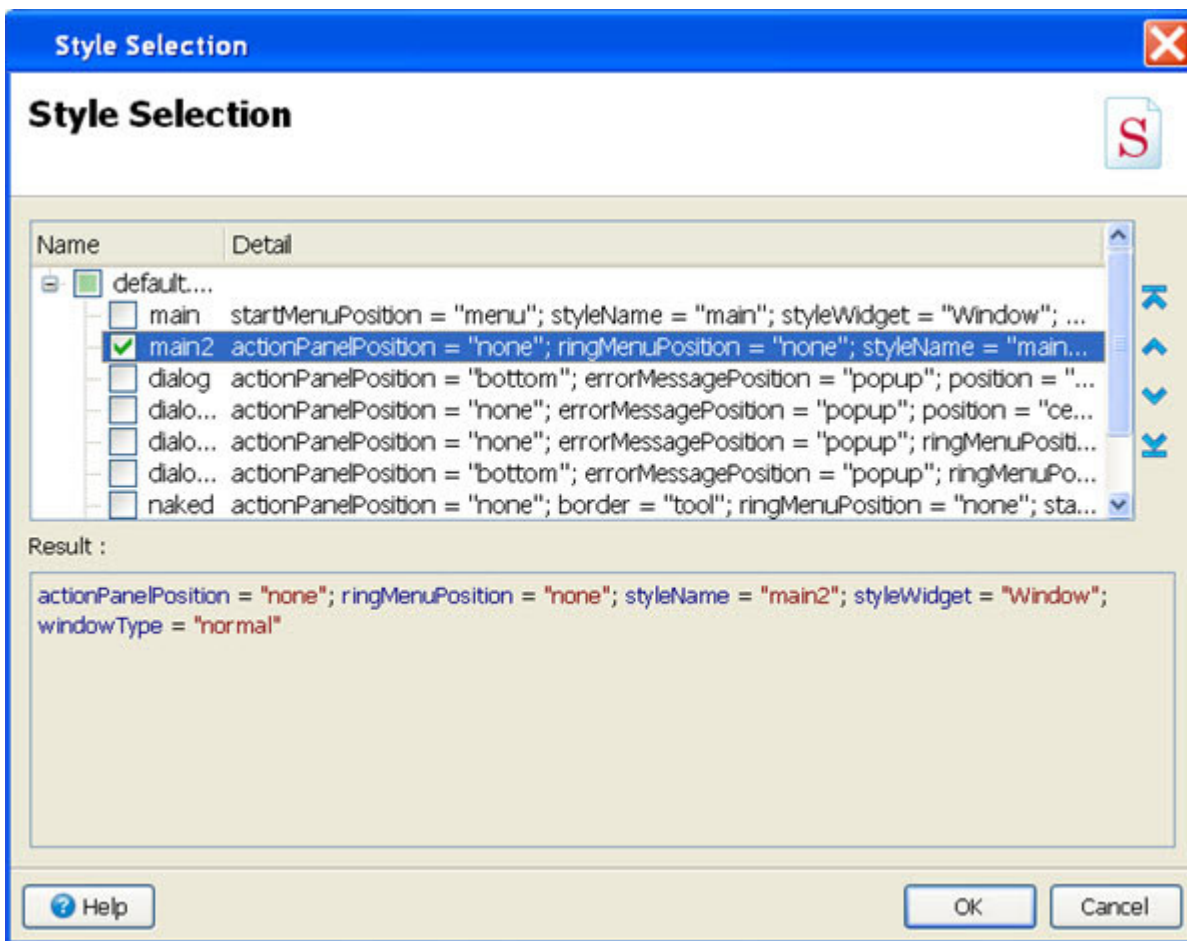


Figure 6-7. Style Selection dialog

Select a checkbox or checkboxes to select individual styles, or select/de-select all using the file name.

- **Name** - name of the Style in the Presentation Styles file.
- **Detail** - the contents of the style.
- **Result** - the attributes resulting from your selection.

If the same attribute is defined in several styles that you have selected, you can use the Up/Down arrow buttons to change the priority order of the styles.

Using Localized Strings

Localized Strings allow you to customize the text displayed by your application, for internationalization or site-specific text. The `localizedStr` property allows you to set localization for an individual widget or form item, where applicable. Setting the checkbox for localized strings to "True" will mark the related strings for localization.

Once the form definition is completed, you can use the `gsform` utility from the command line to generate a text file of the localized strings. For example:

```
gsform -m myform.4fd > myfile.str
```

The format of the file will be "string key" = "string text". You can copy the file and change the string text as needed for localization. Do not change the string key, as this is generated by Studio. See the *Genero Business Development Language User Guide* topic **Localized Strings** for additional information.

The localized strings file must be added to a library for your project in Project Manager, and the library linked to your application node. When the application is Built, the localized string will be compiled and linked into your application.

Studio Form Containers and Widgets

Topics:

- **Containers** - used to group items on the form. Every form item must be contained within a container. A parent container can also have child containers. If there is more than one child container, they must be grouped in a horizontal or vertical layout. See Align/Layout.
 - Grid
 - ScrollGrid
 - Group
 - Folder
 - Table
 - Tree
 - Hrec
 - Data Control
 - HBox
 - VBox
- **Widgets** - designed for data handling (formFields), triggering actions (action views), or decoration.
 - Button
 - ButtonEdit
 - Canvas
 - CheckBox
 - ComboBox
 - DateEdit
 - Edit
 - Field

- HLine
- Images
- Labels
- Phantom Field
- ProgressBar
- RadioGroup
- Slider
- SpinEdit
- TimeEdit
- TextEdit
- WebComponent
- Additional form elements - can be added to form object in Structure View using right-click context menu.
 - ActionDefaultList
 - TopMenu
 - TopMenuGroup
 - TopMenuCommand
 - TopMenuSeparator
 - Toolbar
 - ToolbarItem

Containers

Containers are used to group items on the form. Every form item must be contained within a container. A parent container can also have child containers. If there is more than one child container, they must be grouped in a horizontal or vertical layout.

Grid - container

A Grid is the basic container that allows you to group other items such as widgets and objects. For example, the Grid container makes it easier to display a single record of program data. Grids have no visual representation when the form is displayed. Set the container's properties in the **Properties View**.

Result, previewed in GDC:



Figure 6-8. Grid

ScrollGrid - container

A ScrollGrid is basically a Grid with a scrollbar. A ScrollGrid may be a container of, or contained within, both Grids and ScrollGrids. Set the container's properties in the **Properties View**.

Result, previewed in GDC:

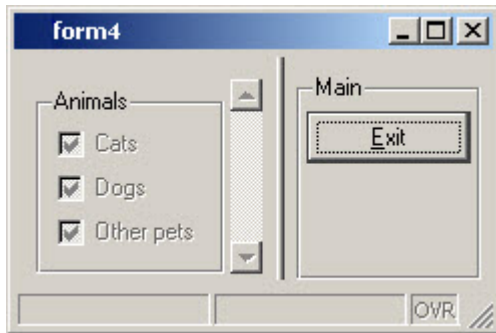


Figure 6-9. ScrollGrid

Group - container

A Group is a "visual" container, used to display a titled box around the elements it contains. Set the container's properties in the **Properties View**.

Result, previewed in GDC:



Figure 6-10. Group

Folder- container

A Folder manages Pages displayed in tabs. You can add or remove the order of Pages using the contextual menu (right-click to display). The Page title may be changed from the **Properties View**. Properties can be specified there for the Folder level and for each page.

Result, previewed in GDC:

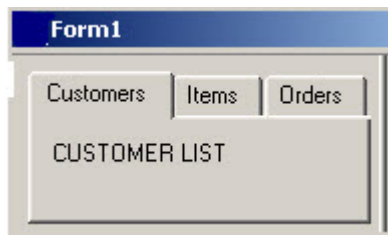


Figure 6-11. Folder

Table - container

A Table container provides record-list presentation in columns and rows. Each record is displayed on a separate line. The default number of rows and columns is set in Preferences. Set the container's properties in the **Properties View**.

Expanding or reducing the height of the table container in the Form Design window (dragging it) will change the number of rows in the container. To add or delete columns or edit a column title, select the column title and right-click to display a menu of options. Drag and drop the columns to change the initial order.

Each column in the Table is represented by a **TableColumn** object. Like formField widgets, the table columns can have properties associated with a database table and column. Set the object's properties in the **Properties View**.

Columns with an aggregate property will display aggregate calculation in summary line of table.

Result, previewed in GDC:

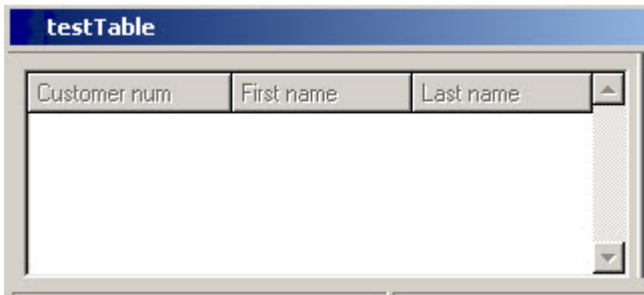


Figure 6-12. Table

Tree - Container

The TREE container allows you to specify the layout of a typical GUI tree view, display data in a parent-child relationship. + and - icons allow the user to expand/contract a branch of the tree. In Form Designer, you specify the layout for the tree; the tree content is provided in your BDL program.

A tree is made of rows and columns, very similar to a table container, with the exception that the rows in a tree can be nested.

The tree container is created by default with two Edit columns, and two phantom columns that do not display on the screen. The phantom columns of the tree specify the id and the parent id of each row, enabling nesting. The other columns display data rows in a normal format without nesting, as in a table.

Properties of the tree container specify the idColumn and parentIdColumn. Additional properties specify images to be used for the expand/contract icons and leaf icons, for example.

The rows in the tree are automatically defined in your form as a screen array. In your BDL program, define a corresponding dynamic array that matches the screen array, using the screen record in the Form to determine column names and their order. Your program must populate the rows of the container at runtime. See Tree Views for detailed information.

Display in Form Designer:

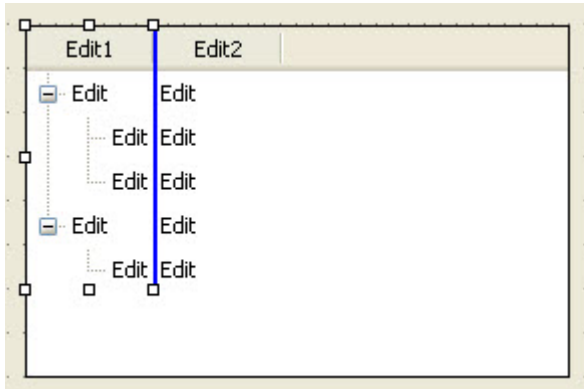


Figure 6-13. Tree

Hrec - container

An Hrec item is a special container that allows you to use spacers to align widgets in your form. Spacers are one or more blanks defining an invisible element that expands automatically. Without spacers, blank areas are ignored and the resulting form may not be what you expect.

An Hrec item is just one line high. Draw it the width that you want. Then, you can add spacers to the right or to the left using the *Spacer left* and *Spacer right* toolbar icons or Container menu items. Set the container's properties in the **Properties View**.

Result, previewed in GDC:

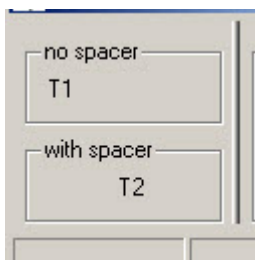


Figure 6-14. Hrec

Data Control

Although this item appears on the Container menu, it is actually a wizard that allows you to add multiple fields to the form, based on database table columns. You must have added the database schema to the DB Schemas tab. The database schema is displayed to allow you to select the table and columns. See “Data Control Wizard” on page 6-27.

HBox - container

The HBox container automatically packs the contained elements horizontally from left to right. Contained elements are packed in the order in which they appear in the structure. By combining VBox and HBox containers, you can define any alignment you choose. Set the widget's properties in the **Properties View**.

VBox - container

The VBox container automatically packs the contained elements vertically from top to bottom. No decoration is added when you use a VBox container. By combining VBox and HBox containers, you can define any alignment you choose. Change the properties for this widget from the **Properties View**.

Widgets

Widgets are designed for data handling (formfields), triggering actions (action views) or decoration.

Button: action view



Figure 6-15. Button

A Button widget defines a standard push button with a label or a picture. Set the widget's properties in the **Properties View**.

A Button is an action view; an action defined in your **4gl** program code can be associated with the button, such as MENU or INPUT. When the button is clicked, the program code defined for the action will be triggered.

ButtonEdit - data handling/action view



Figure 6-16. ButtonEdit

A ButtonEdit widget defines a line edit box with a button on the right side. Set the widget's properties in the **Properties View**. The line edit box is a formField, which can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the table Name and column name properties. The button is an "action view", which can trigger an action defined in your **4gl** program code. Frequently this action is designed to open a popup window that allows the user to browse a list of values for the formField.

Canvas - decoration



Figure 6-17. Canvas

A Canvas widget defines an area reserved for drawing. Set the widget's properties in the **Properties View**.

CheckBox - data handling

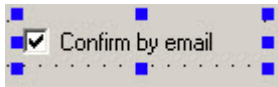


Figure 6-18. CheckBox

The CheckBox widget defines a boolean entry with a box and a text label. Set the widget's properties in the **Properties View**.

This is a data handling widget that represents an underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the table name and column name properties. The value stored in the formField is determined by whether the Checkbox is checked, and is specified in the valueChecked and valueUnchecked properties.

ComboBox - data handling



Figure 6-19. Combobox

A ComboBox is a data handling widget that defines a dropdown box of values, allowing the user to select a value for the underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the sqlTabName and colName properties. Set the widget's properties in the **Properties View**.

The Edit Items dialog

The values in the list of items can be managed (add, delete) by right-clicking the widget in the Form Design window and selecting **Edit Items**. Edit the values in the columns of the dialog:

- **Name** - the value to be stored in the underlying formField represented by the ComboBox, for example, "W".
- **Text** - the value in the list to be displayed to the user, for example "West Region".
- **Localize string** - Specify whether the text is a localized string (true/false).

DateEdit - data handling

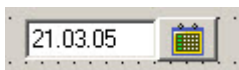


Figure 6-20. DateEdit

A DateEdit is a data handling widget that defines a line edit box with a button that opens a calendar window, allowing the user to select a date value for the underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Set the widget's properties in the **Properties View**.

Edit - data handling

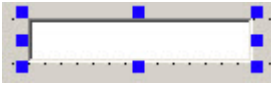


Figure 6-21. Edit

The Edit widget defines a simple line edit box for data input or display. It is a data handling widget for an underlying `formField` - an object that lets you define a field in the form that can be used to manipulate your data. Set the widget's properties in the **Properties View**.

Use the `fieldType` property to specify whether the `formField` is `FORM_ONLY` (the data stored in the field will not relate to a database column), or `TABLE_COLUMN` (the data corresponds to a database column). In the data group, the `tableName` and `columnName` properties identify the database column.

Field - data handling



Figure 6-22. Field

This item type defines a generic form field for data input or display. The real item type (i.e. the widget) and the attributes must be defined in a special database schema file having the extension **val**.

This form field specification allows you to enforce reusability, by using the database schema file to centralize the definition of form fields.

HLine - decoration

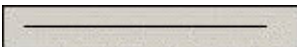


Figure 6-23. HLine

HLine appears in the form as a horizontal line. Set the widget's properties in the **Properties View**.

Image - decoration/data handling

Image items define areas where a picture file can be displayed. Set the widget's properties in the **Properties View**.

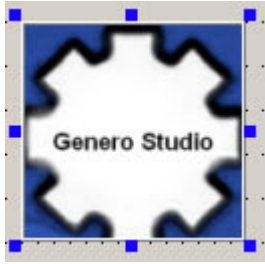


Figure 6-24. Image

A **Static Image** displays an image that has its source file defined by the image property. This type of image item must be used to display text that does not change often, such as background pictures or logos. The item is not a form field.

The image file can be changed from the BDL program by using the API provided to manipulate the user interface (see the *Genero Business Development Language User Guide* topic **Dynamic User Interface** for more details). It is not possible to change the image with a DISPLAY TO instruction.

A **formField Image** is a widget that gets the image file based on the underlying formField. The value of the formField is the image file specified with a URL. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties.

This type of image item must be used to display values that change often during program execution, like database information. The picture can be changed from the BDL program by using the DISPLAY TO instruction to set the value of the field.

Labels - decoration/data handling

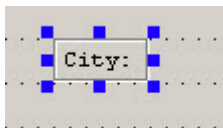


Figure 6-25. Label

Static label - this Label widget contains static text, but also supports some additional attributes, allowing it to be customized. Double-click the widget, or right-click and select **Edit Text**, to edit the text in the Text dialog.

formField label - this Label widget takes the text displayed on the label from the value in the underlying formfield. The formfield can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. The text automatically changes when the values in the table column change.

Change the properties for these widgets in the **Properties View**.

Phantom Field - data handling

A PHANTOM field can be used to specify a formfield that is listed in a screen-record shown in the Records tab, but does not appear in the Layout section of the form. Phantom fields are never displayed to the user, although they can be used by dialog instructions of BDL programs. If you want to implement a screen-array with

all the columns of a database table defined in the schema file, but you don't want to display all the columns in the TABLE container of the LAYOUT section, you must use PHANTOM fields.

Phantom fields are used to store the id and parent id of the nodes in a Tree object.

Since Phantom fields are never sent to the front end, you can use them to store critical data that must not go out of the application server.

To add a Phantom field to your form, right-click on the form object and select **Add Phantom** from the context menu.

ProgressBar - data handling



Figure 6-26. ProgressBar

A ProgressBar is a data handling widget that can indicate the current progress of an operation, based on the value of the underlying formField. This kind of form item does not allow data entry; it is only used to display integer values.

The position of the progress bar is defined by the value of the corresponding form field. The value can be changed from the Genero BDL program by using the DISPLAY TO instruction to set the value of the field.

The VALUEMIN and VALUEMAX properties define respectively the lower and upper integer limit of the progress information. Any value outside this range will not be displayed. Change the properties for this widget from the **Properties View**.

RadioGroup - data handling

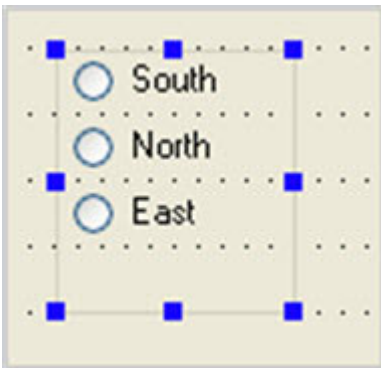


Figure 6-27. RadioGroup

A RadioGroup data handling widget presents its underlying formField as a set of radio buttons from which the user may select a value. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Adding/deleting values for the radio buttons is managed from the contextual menu. Change the properties for this widget from the **Properties View**

Slider - data handling

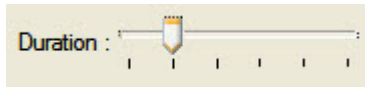


Figure 6-28. Slider

The SLIDER item type defines a horizontal or vertical slider, a classic widget for controlling a bounded value. This widget serves as a data handling widget for the underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Change the properties for this widget from the **Properties View**

The user moves a slider along a horizontal or vertical groove; the slider's position is translated into a value within the legal range defined by the valueMin and valueMax properties.

Important: This type of field can not be used to implement Query by Example (the CONSTRUCT statement).

SpinEdit - data handling

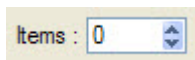


Figure 6-29. SpinEdit

The SpinEdit item type defines a spin box widget (spin button). This widget serves as a data handling widget for the underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Change the properties for this widget from the **Properties View**.

The user can choose a value either by clicking the up/down buttons to increase/decrease the value currently displayed, or by typing the value directly into the spin box.

Important: This type of field can not be used to implement Query by Example (the CONSTRUCT statement).

TextEdit - data handling

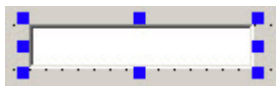


Figure 6-30. TextEdit

A TextEdit data handling widget defines a multiline edit box for data input or display. The underlying formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Change the properties for this widget from the **Properties View**

TimeEdit - data handling

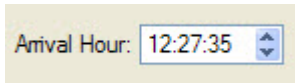


Figure 6-31. TimeEdit

The TIMEEDIT item type defines a time editor widget, which handles data for the underlying formField. The formField can be associated with a database column by changing the fieldType property to TABLE_COLUMN, and specifying the tableName and columnName properties. Change the properties for this widget from the **Properties View**.

This item type allows the user to edit times by using the keyboard or the arrow keys to increase/decrease time values.

Important: This type of field can not be used to implement Query by Example (the CONSTRUCT statement).

WebComponent - data handling

WebComponents are usually complex widgets displaying detailed information on the screen, such as charts, graphs, or calendars.

When you add a WebComponent to your form design, a standard image is displayed in the form, indicating that the object is a WebComponent. The componentType property in the WebComponent section of the Properties View allows you to select the specific WebComponent that you wish to add. Once you select the component, the properties specific to that component are also listed in the Properties View; any form design icon specified for that component will be displayed the form.

In order for a WebComponent to be listed in the componentType combobox, it must be described in an XML file contained in the directory that you specified in Studio Preferences, Tools>>Preferences, Form Designer, Form Element Settings. See “Using Web Components” on page 6-35 for additional information.

Additional Form Elements

Additional form elements can be added to a form object in the Structure View using right-click context menu.

ActionDefaultList - decoration

Action defaults can be used to define the appearance of action views (buttons, menu items, and toolbar items, for example) on the form.

Right-click the form object in the Structure View and select **Add ActionDefault List** from the context menu. To import an Action Default file (.4ad) select **Import Action Defaults**.

ActionDefault - Right-click the ActionDefaultList object to add an ActionDefault using the context menu. For each ActionDefault, specify the properties associated with text, images, accelerators, and comments. These properties will be automatically applied to any action views on the form that have the same name as the name property of the Action Default object.

TopMenu - action view

A Topmenu presents a pulldown menu on the form, composed of actions defined by the current interactive BDL instruction in your Genero BDL program - the MENUstatement, for example. When a Topmenu command is selected by the user, the BDL program triggers the action to which the TopMenu command is bound. See the *Genero Business Development Language User Guide* topic **BDL Tutorial Chapter 5 Enhancing the Form**.

The TopMenu object appears in the Form Structure View, but it does not appear in the design window. It implements a top menu on the form that is displayed to the user.

Display in GDC:

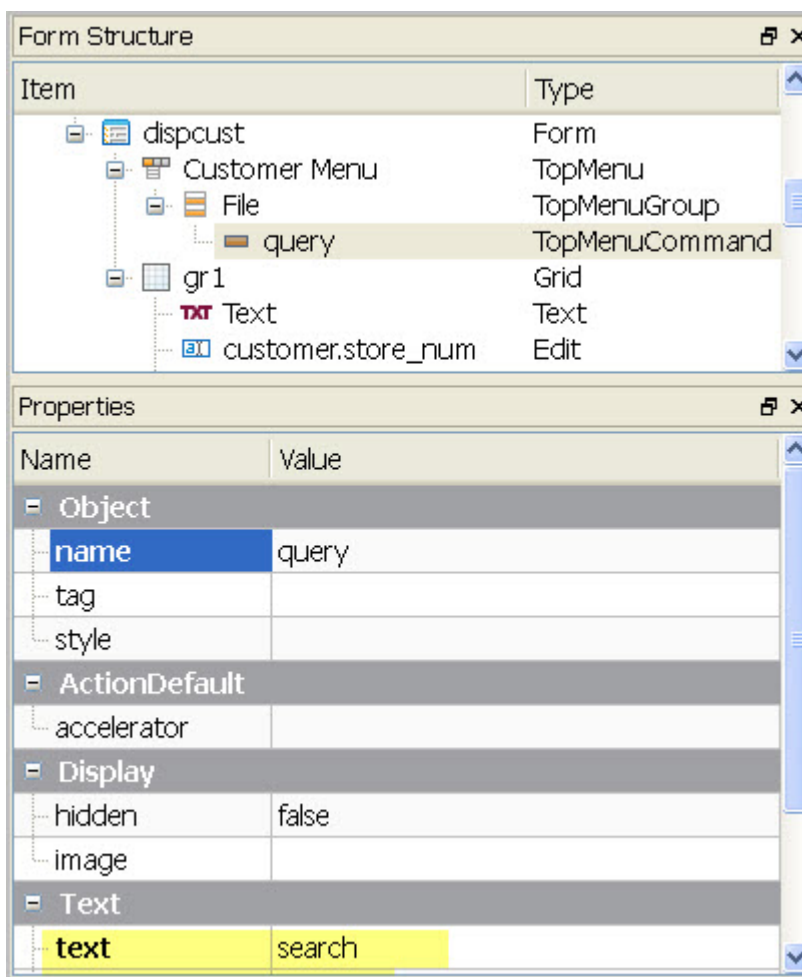


Figure 6-32. Form Structure tab showing a TopMenu

To add a top menu to a form, right-click the form object in the Structure View and select **Add TopMenu** from the context menu. To import an existing TopMenu file (.4tm) select **Import TopMenu**.

The following objects are children of a TopMenu object:

- **TopMenuGroup** - All commands that make up the menu are organized into groups. The group text property is the title for the pull-down menu group. To add this to a TopMenu, right-click the TopMenu in the Structure View and select **Add Group** from the context menu.
- **TopMenuCommand** - specifies the action the menu option must be bound to. The value of the text property of this form object must exactly match the name of an action specified in your BDL program. Right-click a TopMenuGroup in the Structure View and select **Add Command** to add this object to the form.
- **Separator** - an optional vertical line to be displayed on the menu, visually separating some of the commands. Right-click a TopMenuGroup in the Structure View and select **Add Separator** to add this object to the form.

Toolbar - action view

A Toolbar presents buttons on the form associated with actions defined by the current interactive BDL instruction in your Genero BDL program - the MENU statement, for example. When a Toolbar button is selected by the user, the program triggers the action to which the Toolbar button is bound. See the *Genero Business Development Language User Guide* topic **BDL Tutorial Chapter 5: Enhancing the Form**.

The Toolbar object appears in the Form Structure View, but it does not appear in the design window. It implements a toolbar on the form that is displayed to the user.

Display in GDC:

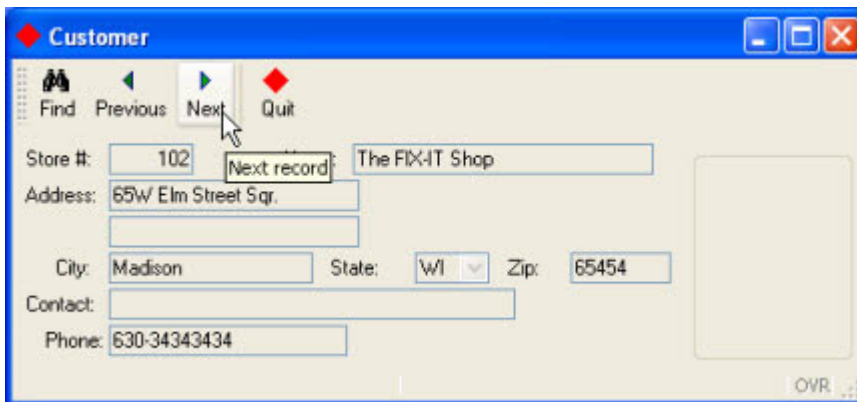


Figure 6-33. Form with Toolbar

To add a toolbar to a form, right-click the form object in the Structure View and select **Add ToolBar** from the context menu. To import an existing toolbar file (.4tb) select **Import ToolBar**.

ToolbarItem - specifies the action the toolbar button must be bound to. The value of the text property of this form object must exactly match the name of an action specified in your BDL program. Right-click a TopMenuGroup in the Structure View and select **Add Command** to add this object to the form.

Separator - an optional vertical line to be displayed on the toolbar, visually separating some of the commands. Right-click a Toolbar object in the Structure View and select **Add Separator** to add this object to the form.

Database Interface

Topics:

- Data Control Wizard
- Column Selection page
- Container Selection page
- Query Creation Page
- Dynamic Properties page

Data Control Wizard

Once you have added a database schema to your project, you can add fields for database columns to your form. The Data Control wizard allows you to add the form fields, and a selected container, to your Studio form definition. The same dialogs appear if:

- **Form Designer** is opened, and you select **Data control** from the **Container** menu.
- from the **File>>New** menu, you select the option **Design, CRUD Form from database** or **Genero Files, Form from database**
- from the right-click context menu for a Business Application Design **Module** icon, you select **Create .4fdm File From table**
- from the right-click context menu for a Business Application Design **Zoom** icon, you select **Create .4fdz File from table**

Column Selection page

Schema List and Column description

Select the database schema from the list of schemas that you have added to your project.

1. Select a table or view name in the schema list, and its columns are displayed in the **Column description** list. If a schema is not available in this list, open the to add it.
2. Select the columns that you wish and use the right arrows to transfer the columns to the **Selected Fields** list, and the left arrows to remove them from the selected fields list.
3. Once the columns are selected, use the up/down arrows to rearrange the column order. Then click **Next**. The Container selection page displays.

Container Selection Page

Container Type

Select the container to hold the fields for the database columns:

- Grid
- ScrollGrid
- Table
- Tree

Grid and ScrollGrid

Label and Field Alignment: Icons allow you to specify the following:

- Label left and field left
- Label right and field right

- Label left and field right
- Label right and field left

Label and Field Properties:

- **Maximum width** - in characters
- **Number of fields** - number of fields per line
- **Top border** - in lines
- **Bottom border** - in lines
- **Left margin** - in characters
- **Right margin** - in characters
- **Field gutter** - spaces between label and data for a field
- **Field gap** - gap between two fields on the same line
- **Repeat fields** - check to repeat the fields

TextEdit - specify maximum width height

Query Creation Page

This page displays only if you have chosen to create a new **CRUD Form From Database Table** for a generated application:

Edit Query

Enter join, where and order by clauses

Join

Left table	Left column	Operator	Right table	Right column
customer	store_num	INNER JOIN	orders	order_num

Additional tables

Table	Alias

Where

Order by

Table	Column	Order

Help OK Cancel

Figure 6-34. Edit query dialog

Sections:

- **Join** - if two tables will be involved in the query, this section is enabled so that you can enter an SQL Join condition for the SQL Query.
- **Additional tables** - you can add a table here that does not have any fields in the form. For example, list the orders table here if you wish to join customer and items tables but the orders table is required to enable the join.
- **Where** - you can add additional conditions to the SQL Where clause, to restrict the rows returned by the SQL Query to a subset of the data.
- **Order By** - you can change the default order in which the SQL Query will return the rows.

Dynamic Properties Page

This page lets you change the label displayed on the form for a form field, or the widget used to display the value.

New Form from Database

Dynamic properties
Enter dynamic properties' values

Enter a value for each dynamic property on each field:

field	label	widget
customer.sto...	STORE NO.	Edit
customer.sto...	store_name	Edit
customer.addr	addr	Edit
customer.ad...	addr2	Edit
customer.city	city	Edit
customer.state	state	ComboBox
customer.zip...	zipcode	ButtonEdit
customer.co...	contact_name	CheckBox
customer.ph...	phone	ComboBox
		DateEdit
		Edit
		FFImage
		FFLabel
		Field
		Phantom
		ProgressBar
		RadioGroup

Buttons: ? Help, < Back, Next >, Finish, Cancel

Figure 6-35. Dynamic properties page

Double-click a label or widget property for a field in order to change it.

Click **Finish** and the database columns and container are added to the form.

Properties List

Topics:

- Properties of the form
- Properties of widgets and containers

Properties of the form

Object

- **name**: name assigned to the form.
- **databaseName**: reference to a database shown in the DB Schemas tab.
- **tag**: string identifier for the form.
- **style**: the name of the style to apply to the form. If a Style File property has been specified for the form, this property is enabled for form items.

Form

- **style File**: a file containing the styles to be used in the form; click the ... button to select a style file for the form. See Using Styles.
- **windowStyle**: If a style file has been specified, the name of the window style to be used by the parent window.
- **minWidth**: minimum number of characters in a line. If set, used by the Genero Desktop Client to calculate the size of the form for display.
- **minHeight**: minimum number of lines. If set, used by the Genero Desktop Client to calculate the size of the form for display.
- **version**: defines a user version string

Geometry

- **gridWidth**: width in characters. Default is set in Preferences. Maximum is 4000.
- **gridHeight**: number of lines in form. Default is set in Preferences. Maximum is 250.

Display

- **image**: Defines the URL of the image resource associated with the form.
- **spacing**: Normal or Compact; indicates spacing hint to display form elements.

Text

- **localizedStr**: check this to apply localization to the form text.
- **text**: text to be displayed in title bar.

PER comments

- All comments imported from the .per file format are grouped here.

Properties of widgets and containers

When an item on a form is selected, the properties that are valid for that form item are displayed in the Properties View, where their values can be viewed and/or changed.

Some properties for widgets, containers and objects are used frequently, and are grouped together for easier reading, and for common management. The composition of the groups can vary, depending on the selected form item.

Refer to the *Genero Business Development Language User Guide* for details on each property.

Object

- **name:** specifies the name of the form item
- **tag:** defines a specific string to identify the form item.
- **style:** defines a presentation style for the form element. Click the ... button to choose from the current available styles. See Using Styles.

Aggregate

- **aggregate:** specifies if this column should have an algorithm applied and displayed in table aggregate row at bottom of table.
- **type:** type of aggregate. Value can be sum, avg, min, max, count, or program.
- **name:** name of the aggregate. (table.column format if associated with database.)
- **text:** text to display in the aggregate row. This can be set for the table or for the column. If no text is set for the column, the table text value will be used.

Action Defaults

- **acceleratorName:** first id="acceleratorName"; accelerator key for an Action Default
- **acceleratorName2:** second accelerator key for an Action Default
- **acceleratorName3:** third accelerator key for an Action Default
- **acceleratorName4:** fourth accelerator key for an Action Default

Appearance

- **unhidable:** indicates that the table column cannot be hidden.
- **unmovable:** indicates that the table column cannot be moved.
- **unsizable:** the table columns can not be re-sized.
- **unsortable:** the table does not allow rows to be sorted.

Color

- **color:** specifies the foreground color of the text displayed by a form item. For backward compatibility, color can be combined with these options: **reverse**, **left**, **blink**, and **underline**.
- **colorCondition:** defines a Boolean condition based on field values to set the color attribute dynamically.

Comment

- **comment:** specifies a message to display on the Comment line
- **localizedStr:** localized string, if any, associated with the comment. See Using Localized Strings.

Constraints

- **noEntry:** prevents a user from entering a value during input; may be specified in the database table schema instead.
- **notNull:** indicates that the field does not accept null values; may be specified in the database table schema instead.

- **required:** requires a user to supply a value during input; may be specified in the database table schema instead.
- **case:** specifies the case of character data. Value can be:
 - **None:** no control
 - **Downshift:** converts to lowercase any uppercase character data
 - **Upshift:** converts to uppercase any lowercase character data
- **verify:** requires that data be entered twice when the database is modified
- **queryEditable:** allows a combobox field to be editable during a CONSTRUCT.
- **wantNoReturns:** forces the field to reject RETURN keys.
- **wantTabs:** forces the field to consume TAB keys.
- **wantFixedPageSize:** forces the table to have a fixed height when the parent window is re-sized.

Display

- **hidden:** makes a field invisible on the front end, but can be used in programs.
- **fontPitch:** can be fixed or variable
- **image:** specifies the image file containing the icon to be displayed in a button, or the picture of an image form item.
- **invisible:** does not echo characters on the screen during data entry.
- **justify:** specifies the text justification.
- **gridChildrenInParent:** when a Group is inserted into a grid, items put inside the Group are automatically children of this grid.
- **sliderOrientation:** select vertical or horizontal orientation
- **splitter:** indicates that the container (typically, a VBox or HBox) must have a splitter between each child element held by the container.
- **scrollbars:** use to specify scrollbars for a form item. Values can be none, horizontal, vertical, or both.
- **stretch:** specifies how a widget must resize when the parent container is re-sized. Values can be none, x, y, or both

Display Like

- **displayTabName** applies column attributes to a field. This property references the table.
- **displayColName** applies column attributes to a field. This property references the column.

Formatting

- **century:** specifies how century is handled for date data types. Value can be:
 - **C:** use the past, future, or current year closest to the current date.
 - **F:** use the nearest year in the future to expand the entered value.
 - **P:** use the nearest year in the past to expand the entered value.
 - **R:** prefix the entered value with the first two digits of the current year.
- **format:** output format to apply to DECIMAL, SMALLFLOAT, FLOAT or DATE values

FormField

- **fieldType:** (for formfield items only). specifies the category of the values stored in the item. Value can be:

- **TABLE_COLUMN**: the value is defined in terms of a database column; the `databaseName` property of the form must not be "formonly". See `sqlTabName` and `colName`.
- **TABLE_ALIAS**: the value is defined in terms of an alias that is assigned to a database table.
- **FORM_LIKE**: the value is like **FORM_ONLY**, but takes its description from a database column
- **FORM_ONLY**: the value is not defined in terms of a database column
- **sqlTabName**: (formfield only) the name of the database table; required when the value for `fieldType` is not **FORM_ONLY**. See also `colName`.
- **colName**: the name of the database column; required when the value for `fieldType` is not **FORM_ONLY**. See also `sqlTabName`.
- `fieldId`
- **widget**: defines the type of widget to be used for presentation. This property is read-only. The widget type can be changed from the context menu for the form's design view or the structure view.
- **dataType**: data type of the property value, for formfield items only). For form-only formfield objects, the data type has to be specified. For other formfield objects, data type is determined from the database column type and can't be modified.
- **defaultValue**: (for formfield items only). Assigns a default value to a formfield during data entry.
- **sample**: provides the text to be used as a sample to compute the width.
- **include**: lists a set of acceptable values during data entry.
- **autoNext**: causes the cursor to automatically advance during input to the next field when the current field is full.
- **autoScale**: causes the form element contents to automatically scale to the size given to the item.
- **picture**: imposes a data-entry format on `CHAR` or `VARCHAR` fields.
- **program**: invokes an external program to display `TEXT` or `BYTE` values.
- **scroll**: can be used to enable horizontal scrolling in a character field.
- **initializer**: specifies an initialization function for the item.
- **step**: the integer value between two marks on a slider widget.
- **items**: defines a list of values to be used by the form item.
- **valueChecked**: defines the value to be associated with a checked checkbox.
- **valueUnchecked**: defines the value to be associated with an unchecked checkbox.

Geometry

- **posX**: position of upper left corner of a form item.
- **posY**: position of upper left corner of a form item.
- **gridWidth**: item size.
- **gridHeight**: item size.

Limits

- **valueMax**: defines the upper limit of the value.
- **valueMin**: defines the lower limit of the value.

Matrix Details

- **repeat**: repeat fields checkbox
- **rowCount**: number of rows
- **columnCount**: number of columns
- **stepX**: Distance between columns; the default step is 1
- **stepY**: Distance between rows; the default step is 1

Miscellaneous

- **doubleClick**: defines the action to be sent when the user double-clicks on a TABLE row.
- **rowHeight**: force rowHeight to specific size for all rows in table.
- **totalRows**: total number of rows in a table container.
- **action**: action name to be sent to the program when the item is activated.
- **tabIndex**: defines the tab order for the form item when displayed using Genero Desktop Client.

Object

- **name**: specifies the name of the form item
- **tag**: defines a specific string to identify the form item.
- **style**: defines a presentation style for the form element. Click the ... button to choose from the current available styles. See Using Styles.

Size

- **sizePolicy**: indicates sizing hint to display form elements
- **width**: defines the initial width of the form item in pixels
- **width unit**: unit of measure of width
- **height**: defines the initial height of the form item
- **height unit**: unit of measure of height

Text/Title

- **text**: defines the label associated with a form item.
- **localizedStr**: localized string, if any, associated with the comment. See Using Localized Strings.

Tree

- **parentId**: specifies the form field that contains the identifier of the parent node of a tree node. This attribute is mandatory.
- **idColumn** : specifies the form field that contains the identifier of a tree node. This attribute is mandatory.
- **expanded column** : specifies the form field that indicates whether a tree node is expanded
- **isNode column** : specifies the form field that indicates whether a tree node has children
- **image expanded** : sets the global icon to be used when a tree node is expanded
- **image collapsed** : sets the global icon to be used when a tree node is collapsed
- **image leaf** : defines the global icon for leaf nodes of a tree container
- **image column** : defines the form field containing the image of a field.

Validate Like

- **validateTabName** - applies attributes that are defined in the .val database schema file for the specified column. This property references the table.
- **validateColName** - applies attributes that are defined in the .val database schema file for the specified column. This property references the column.

Web Component

- **componentType**: specifies the WebComponent that you wish to add to the form. The choices available from the combobox are dependent on the WebComponents described in XML files that are stored in a directory specified in Form Designer Preferences.

Using Web Components

Topics:

- Adding a WebComponent Widget to a Form
- WebComponent Setup (advanced)

Adding a WebComponent Widget to a Form

Web Components are usually complex widgets displaying detailed information on the screen, such as charts, graphs, or calendars.

Each WebComponent widget is described in an XML description file having the extension **.wcsettings**. All the description files must be stored in a single directory. This directory must also contain any icon files to be displayed on the form design page for the WebComponent. Specify the directory using Tools>>Preferences, Form Designer, Form Element Settings, WebComponent Settings. Once you have set this directory, you may add the widgets to your form design documents:

- Select **Widget>>WebComponent** from the Studio menu, or click the WebComponent toolbar icon, and add the widget to the form.
- Then, choose the specific WebComponent from the combobox for the **componentType** property.

When you add a WebComponent widget to your Studio form design, a standard image is displayed in the form, indicating that the object is a WebComponent. The **componentType** property in the WebComponent section of the Properties view allows you to select the specific WebComponent that you wish to add. Once you select the component, the properties specific to that component are also listed in the Properties view; if an icon associated with this component has been created for the form design page, it will be displayed.

WebComponent Setup (advanced)

To make Web Components available for Studio form designers, the description of the Web Component must be described in an XML file having the extension **.wcsettings**. A separate XML file must be created to describe each type of WebComponent, **storecalendar.wcsettings** for example. The **componentType** property in the Form Design Properties view is a combobox that displays the names of the available WebComponent widgets ("storecalendar" for example).

You can display an icon on the form design document when a web component is selected; otherwise the default icon is displayed. The name of the icon file must be the same as that of the **wcsettings** file, for example, **storecalendar.wcsettings** and **storecalendar.jpg**.

The form designer must specify the directory that contains the `wcsettings` and icon files; see [Adding a Web Component to your Form](#).

WebComponent description files

Example file `chart.wcsettings`:

```
<?xml version="1.0" encoding="utf-8"?>
<WebComponent>
  <DynamicProperty name="type" type="TEXT" label="type"
    description="type of chart"
    initialValue="" />
  <DynamicProperty name="caption" type="TEXT" label="caption"
    description="caption"
    initialValue="" />
  <DynamicProperty name="subcaption" type="TEXT" label="subcaption"
    description="subcaption" initialValue="" />
  <DynamicProperty name="xaxisname" type="TEXT" label="X label"
    description="label of X axis" initialValue="" />
  <DynamicProperty name="yaxisname" type="TEXT" label="Y label"
    description="label of Y axis" initialValue="" />
  <DynamicProperty name="numberPrefix" type="TEXT" label="numberPrefix"
    description="number prefix" initialValue="" />
  <DynamicProperty name="labels" type="STRINGLIST" label="labels"
    description="labels" initialValue="" />
  <DynamicProperty name="values" type="STRINGLIST" label="value"
    description="values"
    initialValue="" />
</WebComponent>
```

The XML schema description for the `.wcsettings` file is contained in an XSD file named `wcsettings.xsd`, located in the Studio installation directory `conf/schema`.

Chapter 7. Meta-schema Manager

Topics:

- “Genero Studio Meta-schema Manager”
- “The Studio Meta-schema File (.4db)” on page 7-8
- “Extracting the Meta-schema from a Database” on page 7-16

Genero Studio Meta-schema Manager

Topics:

- Overview
 - Genero Studio Meta-schema file (.4db)
 - When the Metadata is Used
 - BDL Column Definition file (.sch)
 - When the .sch file is Used
- Creating the Meta-schema File (.4db)
- Adding the Meta-schema to your Application
- The DB Schemas Tab
- The Database Menu
- New Database Schema
- Generate SCH files and creation scripts
 - Advanced Properties

See also: The Meta-schema File (.4db), Extracting the Meta-schema From a Database

Overview

Genero Studio Meta-schema file (.4db)

A Studio Meta-schema is an XML file that is a central repository of the Database Metadata - the information about the tables, columns, and relations of a relational database. It also provides the appropriate default values when using the tables, columns, or relations in Studio Forms or Studio-generated applications.

When the Metadata is Used

The Metadata is not used by Studio when an application is executed. Studio uses the Genero runtime system and the specific database client software to access any databases referenced in a Genero application. However, the Meta-schema file is required when you use Studio to:

- Examine the structure of a relational database's tables, columns, and other attributes
- Create a Genero Studio form definition based on a database table
- Generate application code

As a result, it's important that the Meta-schema file match the current structure of the database itself. If changes are made in the database structure, any Meta-schema files that describe the database must also be updated.

Genero BDL column definition file (.sch)

Genero BDL .sch files contain definitions of the database tables and columns. This .sch file is automatically created when you compile a Genero Studio Meta-schema file (.4db), or you can create it using the Studio Database main menu options.

When the .sch file is used

In Studio, the BDL .sch file is used when a .Genero BDL program (.4gl file) is compiled. If you compile a BDL program from the command line, you must have a copy of the .sch file in the same directory.

Creating the Meta-schema file (.4db)

Meta-schemas can be created using the Studio main menu **Database** options:

- **Database>>New Schema** - to extract the Meta-schema information from any supported database to which your computer has access. This requires your system to have access to the database. You are prompted for parameters for accessing the database, based on the database management system.
- **Database>>Import .SCH file** - to import an existing Genero BDL column definition file (.sch), converting the format to a Studio Meta-schema file (.4db). Provide the Meta-schema name and path in the dialog window that opens.

When a Meta-schema file (.4db) is created, it is automatically added to the list in the DB Schemas Tab if you have saved the file in the current project. By default, the .4db file is saved in the Target directory of the current project.

New Empty Meta-schema

Use File>>New, Design (MDA), Schema (.4db) to create an empty .4db file where you can add tables, indexes, columns, and their relationships. See The Database Schema.

Use File>>Save to save the .4db file in your project.

Adding the Meta-schema to your Application

If you used the main menu option File>>New to create your project, the default structure of your project includes nodes for a **project**, **application**, **library**, and **databases**; the dependencies between the default nodes has been predefined. When you save your Meta-schema file in the Databases node of the project, the dependency for the application node in the project already exists.

However, if you created your own project structure, you must:

1. Add the Meta-schema file (.4db) to an application or library node in the structure, if you have not already saved it in the project. For example:

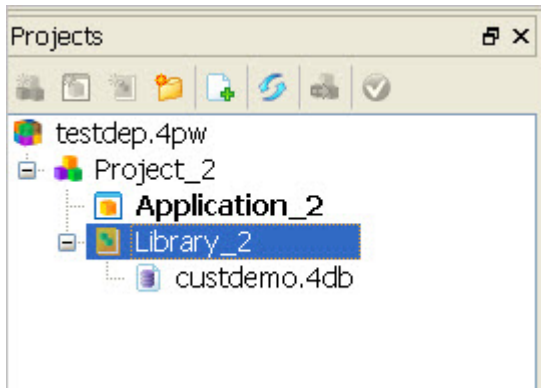


Figure 7-1. Adding a meta-schema file

2. Add the dependency on the Meta-schema file for any application nodes - right-click the **application node** and select Advanced Properties, Dependencies. Check the box for the node containing the Meta-schema file (.4db). For example:

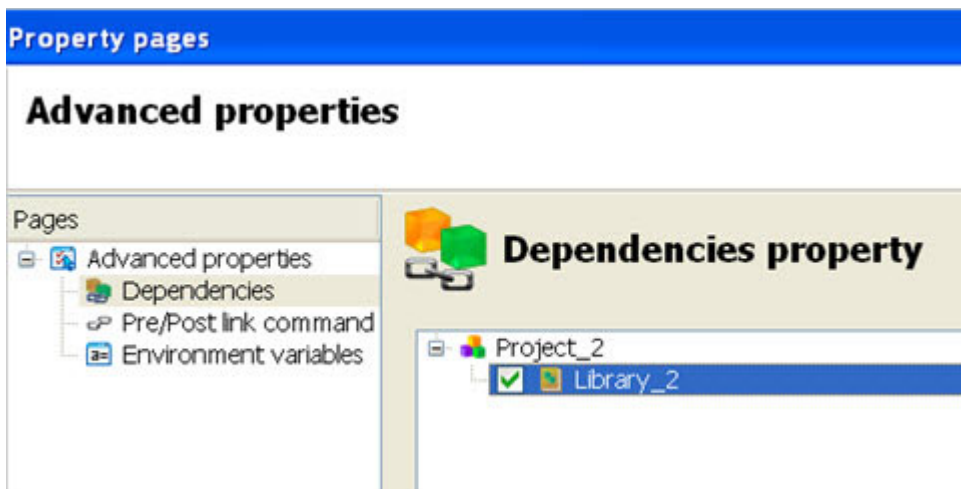


Figure 7-2. Dependencies property

The DB Schemas tab

The DB Schemas Tab displays the database tables, with their column information. Double-click in the database name in the Schemas Tab to display a record of the Meta data for each table in the Main Window.

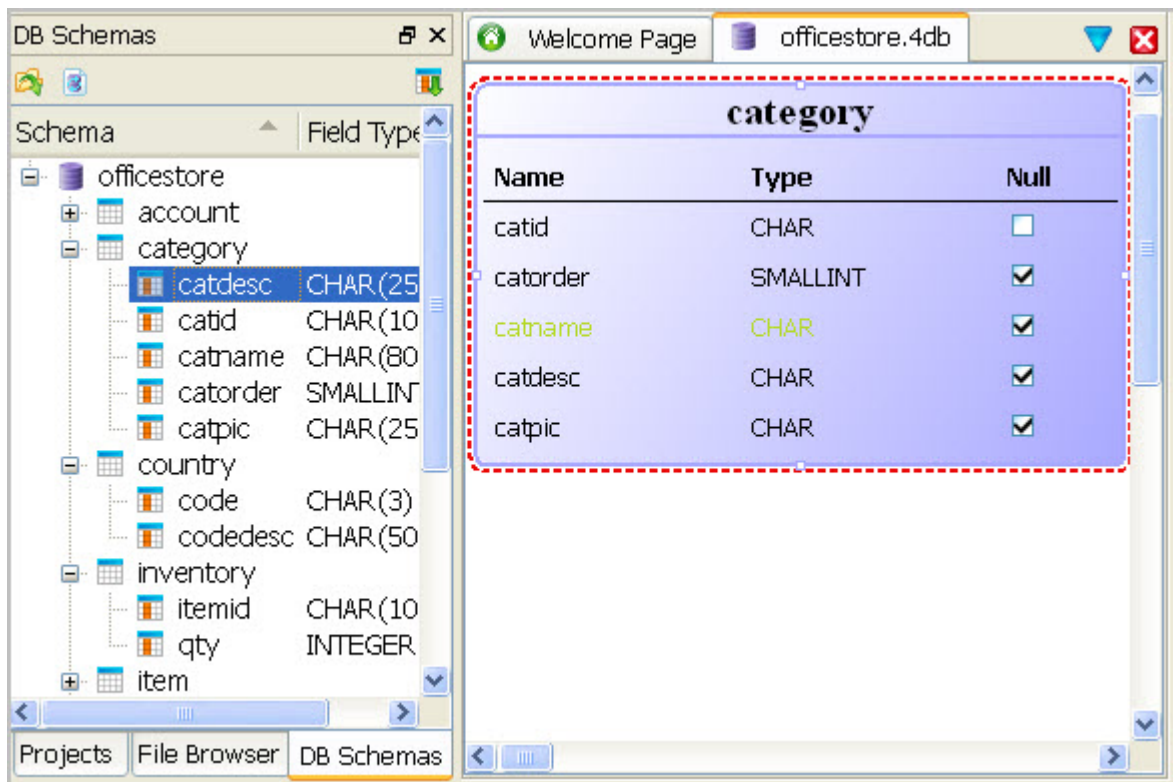


Figure 7-3. DB Schemas

A database Meta-schema is added to the DB Schemas tab when you:

- **save a new database Meta-schema file (.4db) in the current project;** the Meta-schema is available in the DB Schemas tab when you work on projects in that workspace.
- **add the database Meta-schema file** to the list of meta-schemas in Preferences (use the Studio main menu option Tools>>Preferences, Meta-schema Manager); the Meta-schema is available in the Db Schemas tab for all projects.

Use the icons at the top of the tab to:

- **open a Meta-schema** - open the .4db file in a tab in the Studio central work area; this is also available as an option on the right-click menu
- **update from database** - merge any changes made in the structure of a database into your Meta-schema file. This requires that you have access to the database. This is also available as an option on the right-click menu. Any information that you have added to the Meta-schema will be preserved. When the changes are merged, the Meta-schema Manager verifies that the database objects in the original Meta-schema still exist. If one object is not found, its status is set to "Missing". See Missing status for additional information.
- **sort column in database order** (far right icon) - change the sort order of the columns, from alphabetical order (the default) to the order in which the columns are defined in the database.

Once a Meta-schema file (.4db) has been created, you can enrich it with information that is not present in your database. The Properties view displays the properties that can be modified:

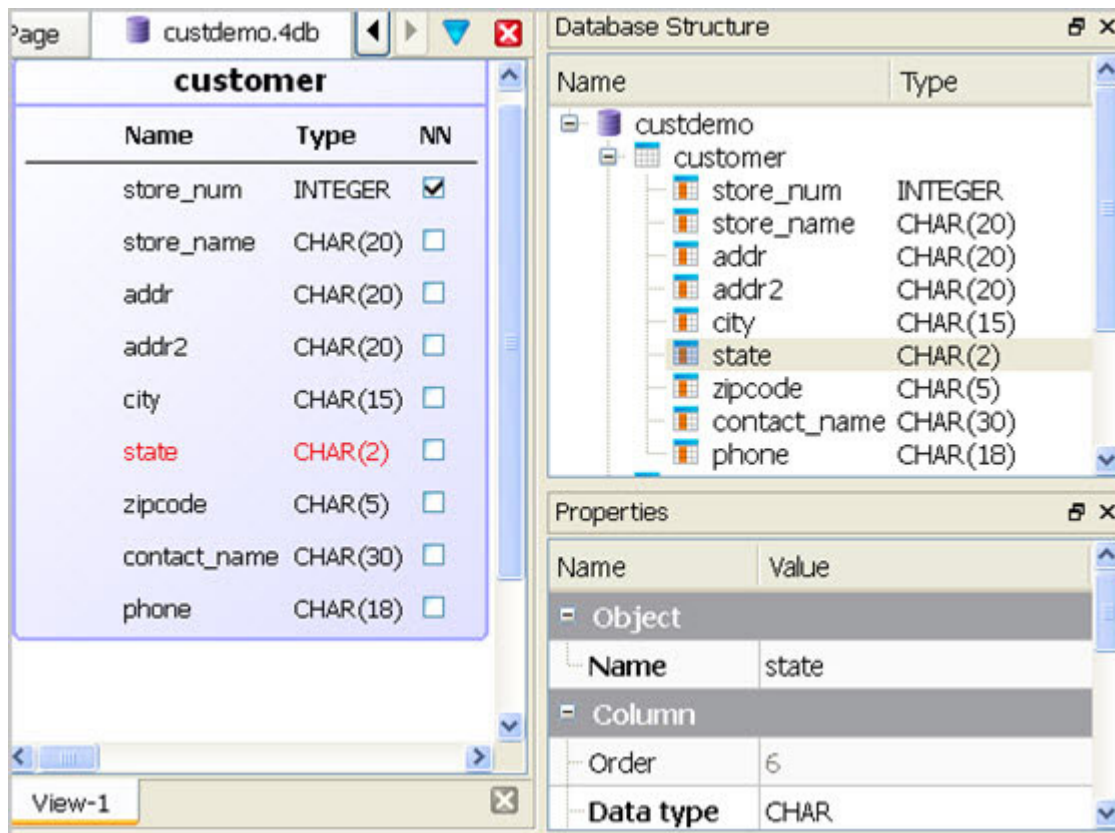


Figure 7-4. Enrich your meta-schema file

Right-click in the Schema tab, and select **Update from database** to synchronize the schema with the database, in case of changes. All information that you enter is kept in the Meta-schema, even when a database table has disappeared from the target database.

Right-click in the Schema window to add tables, columns, indexes, and foreign key relations.

The Database Menu options

- Extract schema file (Studio .4db file is created from database)
- **Update schema** (Studio .4db file is updated from database)
- Import SCH file (Studio .4db file is created from BDL .sch file)
- Generate SCH file (BDL .sch file is created from Studio .4db file)
- Generate database creation script (BDL .4gl file is created from Studio.4db file)
- **Layout** - rearrange the items in the database structure view
- **Advanced Properties** - specify the extraction and generation options

Generate SCH files and Database Creation Scripts

When the database schema (.4db file) is open in the Studio Main Window, the **Database** main menu option offers the following:

- **Generate .SCH file** - generate a BDL schema file (.sch) from the .4db file. This file is automatically created by Studio when a .4db file is compiled, or you can

generate it manually, The BDL schema file is used when Studio compiles a .Genero BDL program (.4gl files). See Extracting a Schema for additional information.

- **Generate database creation script** - generate an SQL command script to create the database that is described in the .4db file. This script can be used to create a new, empty database having the structure described in the .4db file. The following dialog appears, allowing you to specify options for the script generation:

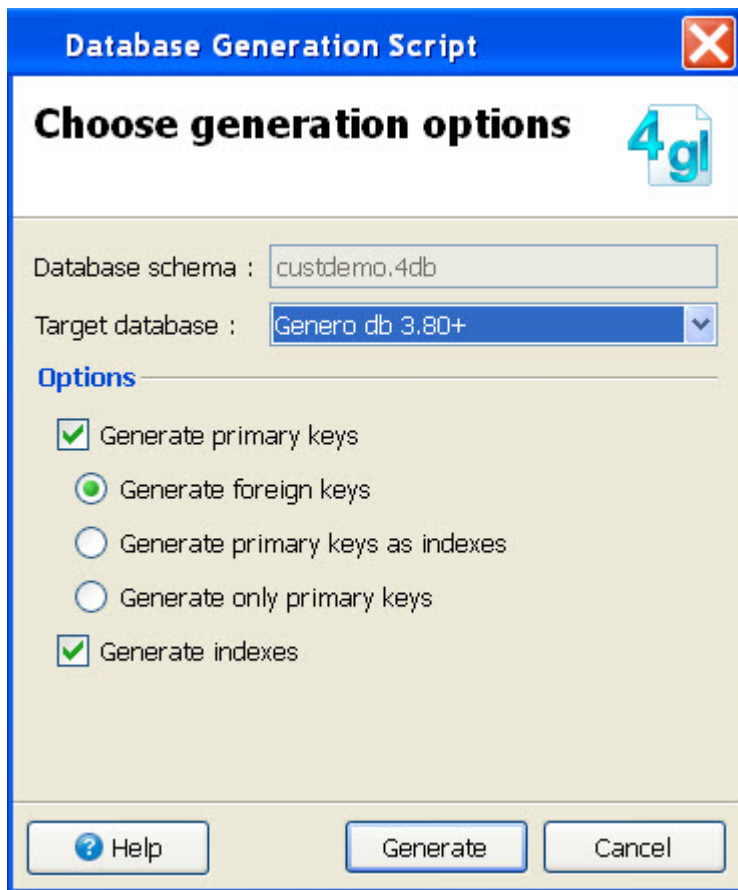


Figure 7-5. Database Generation Script dialog

The Generation tab of the **Advanced Properties** context menu item also allows you to specify these generation options for the script.

Note: If you have modified the database schema for an existing database, the generated script will first drop the existing tables including their data, and then recreate the structure of the database based on the modified schema. Prior to executing the script, **you must save your data externally**, and then load it back into the newly created database.

- **Advanced Properties** - right-click the background of the database schema in the central work area to display the dialogs for specific the extraction and generation options. Also see Extracting a Database Schema for additional information on schema extraction.

The file path of the generated file is shown in the File Path property in the Properties view.

Advanced Properties

Tabs on the Edit Schema Properties dialog allow you to specify Extraction and Generation options.

Extraction tab

This tab allows you to specify the options to be used when you extract a schema from an existing database.

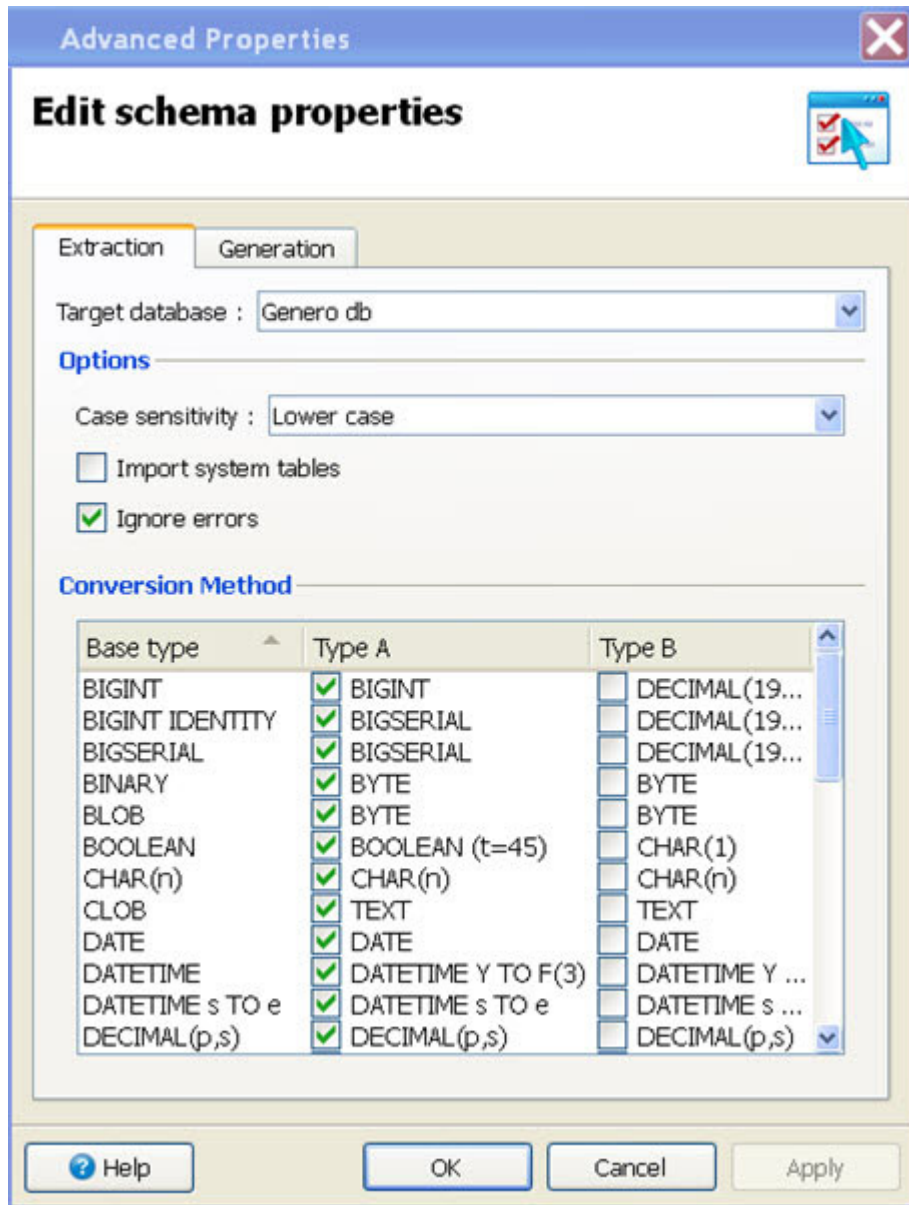


Figure 7-6. Edit schema properties dialog

- **Target database** - specify the database to use for the schema extraction
- **Case sensitivity** - specify how case in database object names should be handled
 - **Case sensitive**: case won't be changed on database objects

- **Lower case:** database object names will be converted to lower case
- **Upper case :** database object names will be converted to upper case
- **Import system tables** - check to include system tables in the schema
- **Ignore errors** - specify that conversion errors should be ignored. If this option is unchecked, the extraction will stop as soon as an error occurs (for example, if a table column has an unsupported type.)
- **Conversion Method** - selection the type of conversion you wish for the specific data types; the default choice is Type A.

Generation tab

This tab contains the same options as the dialog for generating a database creation script. Use the checkboxes in this tab to specify whether to include primary keys, foreign keys, and indexes in the database schema generation script.

The Studio Meta-schema File (.4db)

Topics:

- Open the File
 - The schema
 - Column Information
- Indexes and Constraints
 - Add/Edit User-defined Indexes
 - Add/Edit User-defined Foreign Keys
- New tables and columns
 - Adding tables
 - Adding columns
- Revert changes
- Missing status
- Filter View

See also: Extracting the Meta-schema from a Database, Genero Studio Meta-schema Manager

Open the File

The Studio Meta-schema file (.4db) can be displayed using **File>>Open**, or from the right-click menu of the Database Schemas view or Project view. Each table is described by a record in the Main work area. The structure of the database tables is displayed in the Structure view.

The schema (.4db)

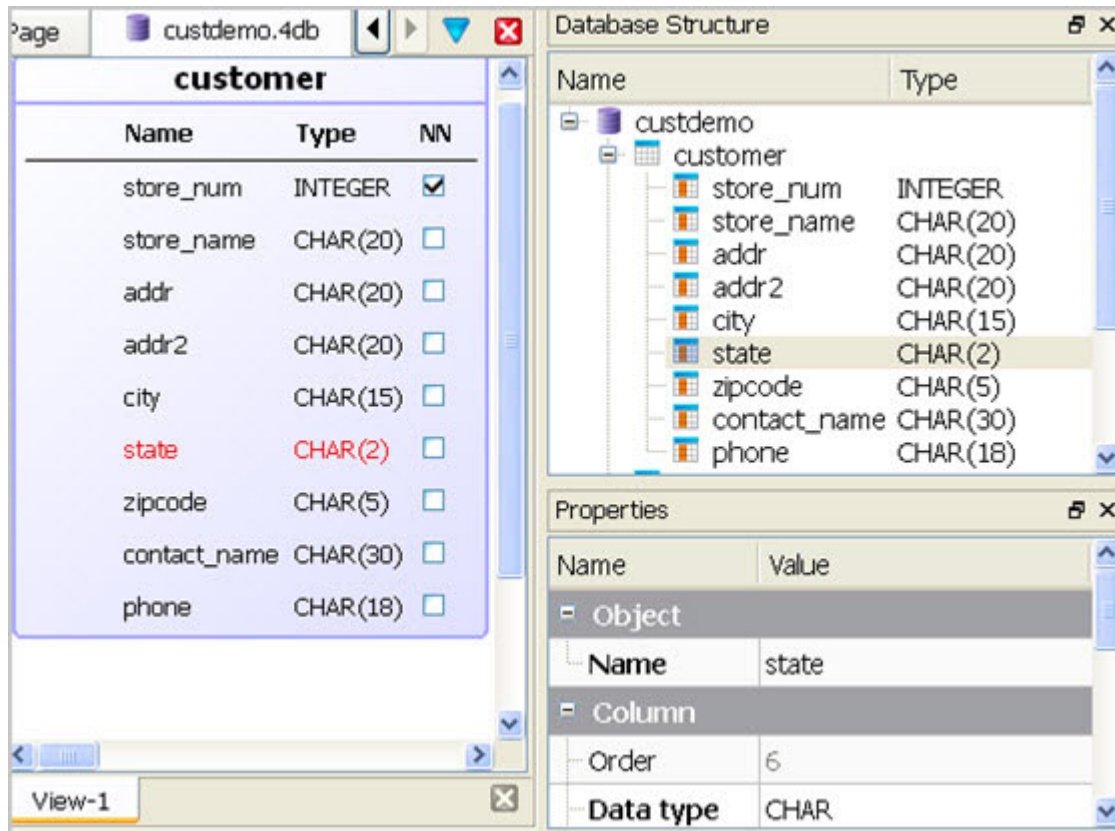


Figure 7-7. Viewing the schema (.4db)

Once a Meta-schema file (.4db) has been created, you can enrich it with information that is not present in your database. The Properties view displays the properties that can be modified.

Right-click the View tab at the bottom of the document to duplicate, rename, or delete a view, allowing you to have multiple views of the same database.

Update from Database - the Meta-schema can be re-synchronized with the database, in case of changes. All the information that you entered is kept in the Meta-schema, even when a database table has disappeared from the target database.

You can revert the database schema to the state when it was last retrieved.

Icons on the database view and in the structure indicate when a column's status has been changed - added (green +), modified (red !), or removed (red X).

Right-click in the page background of the central work area to display a menu of options:

- **Add** tables, columns, indexes or foreign keys.
- **Layout** - rearrange the items in the view of the database structure
- **Show Filter View** - open a filter dialog to select the specific objects to be displayed in the database structure view
- **Advanced Properties** - specify extraction options for the database schema, and generation options for a database creation script

Database Meta data can be extracted from a relational database into a database schema file, or it can be based on an existing database schema file.

Column Information

Select a column in the tree view to display its information in the Properties view. Some important values that you might want to change are:

- **Widget** - change the default widget that will be used for the column
- **Label** - change the default string that is used as the label for a form field

If you update the Meta-schema from the database, the changes you have made are preserved.

Indexes and Constraints

You can add user-defined columns, indexes, and constraints; right-click the table name in the Database Structure view to invoke the context menu options.

- Add index
- Add foreign key

Add Indexes

Right-click the table, and select the context menu option Add Index.

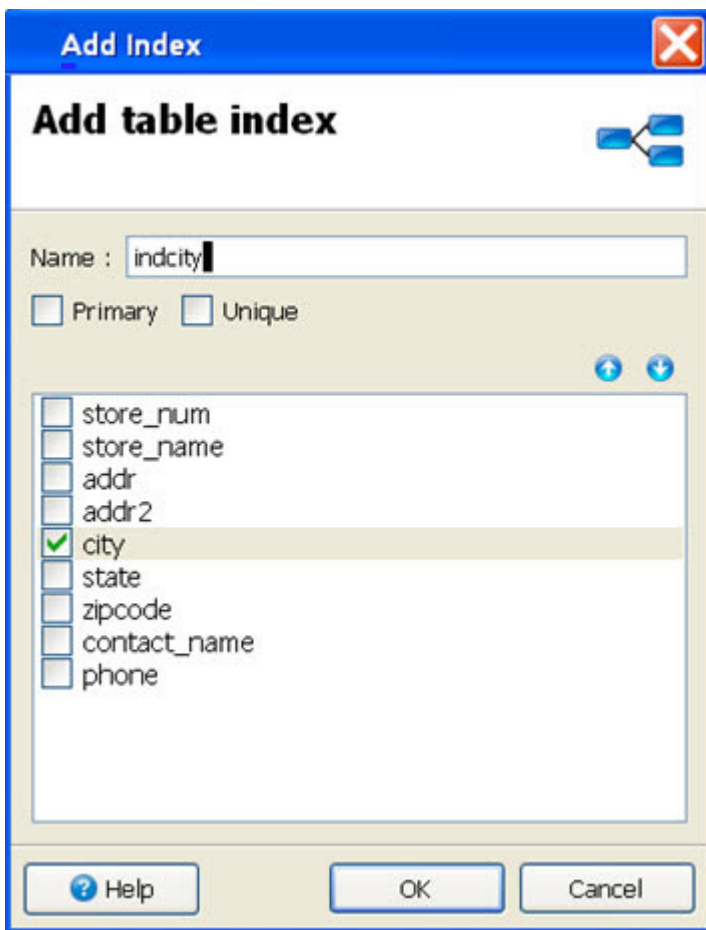


Figure 7-8. Add Index dialog

You can specify that this index is for a primary key or is unique.

Indexes that are part of the database structure, as well as any user-defined indexes, are displayed as part of the record in the Structure view.

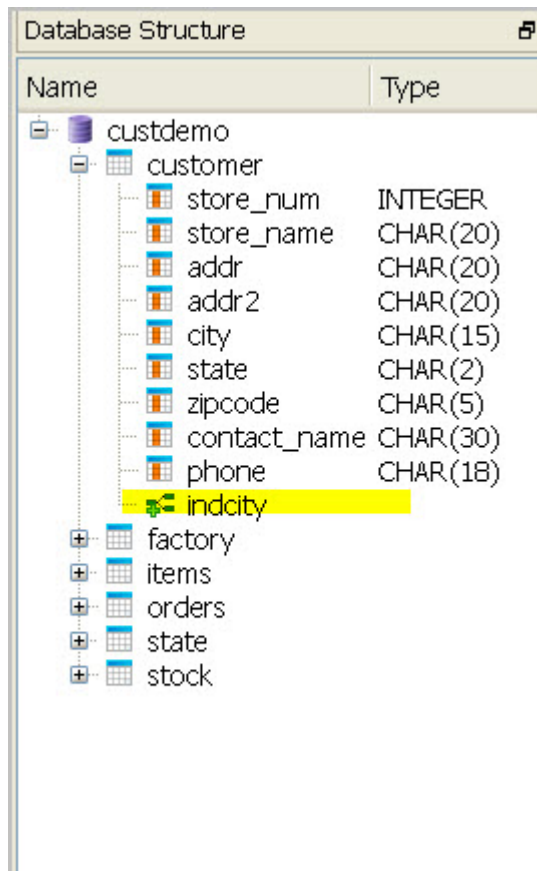


Figure 7-9. Database Structure view

Select the index to display its properties in the Properties view.

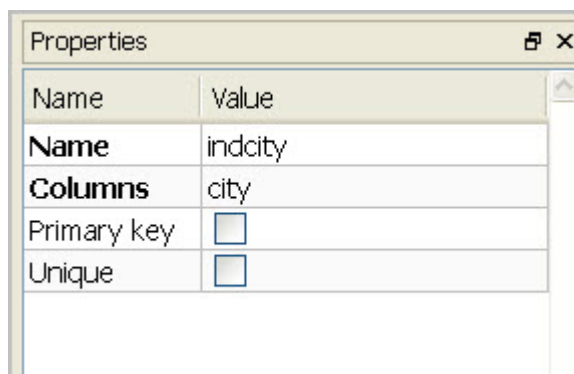


Figure 7-10. Properties view

If an index contains more than one column, the order of the columns is indicated. Checkboxes indicate whether the index is primary or unique.

Add Foreign Keys

Foreign Keys that are part of the database structure are displayed in the Table's Structure tab, as well as any user-defined foreign keys that you create. Right-click the table, and select the context menu option **Add Foreign Key**.



Figure 7-11. Add Foreign Key dialog

- **Table** lists the table that contains the foreign key
- **Referenced table** lists the database table that contains the primary key being referenced
- **Column** lists the table column that references the primary key column in the referenced table
- **Ref. Column** lists the name of the primary key column in the referenced table

The schema display in the Central work area displays the relationship.

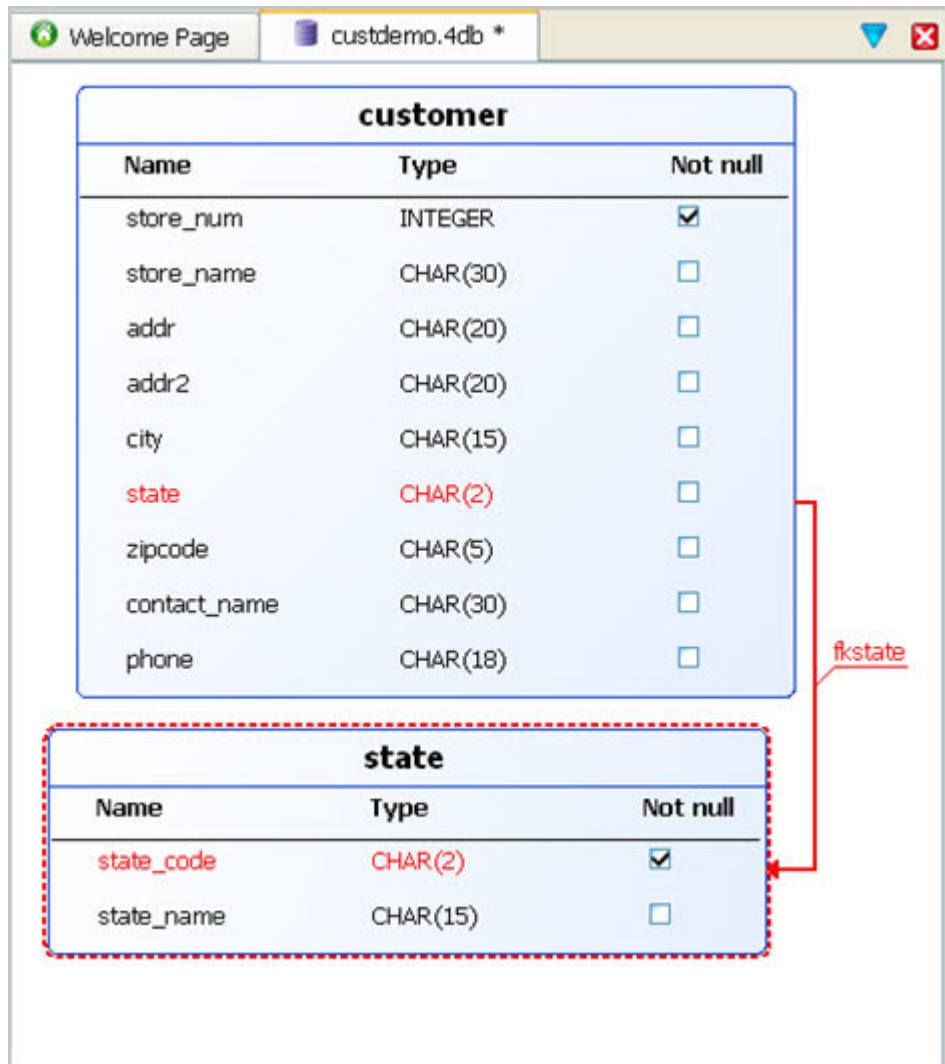


Figure 7-12. Foreign Key in Central work area

The foreign key constraint is added as a child node of the table node in the Database Structure view. Select the foreign key constraint to display its properties.

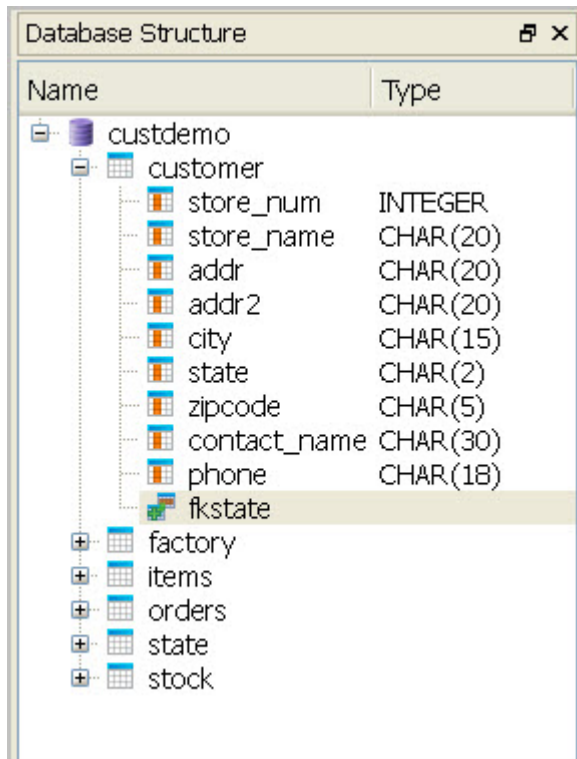


Figure 7-13. Foreign Key in Database Structure view

Alternative method for adding a Foreign Key

You can also define the foreign key by drawing the relationship between columns using the mouse. Right-click on the background of the window and select **Add Foreign Key**. Click the foreign key column in one table, and drag the mouse towards the primary key column in the table to be referenced. Once you release the mouse button, the relationship is displayed and the foreign key is added.

New tables and columns

Right-click in the schema (.4db) displayed in the Central Work Area to display the context menu to add a database table.

Add a table

Add a new table to the database schema. Right-click the table heading to add columns to the table.

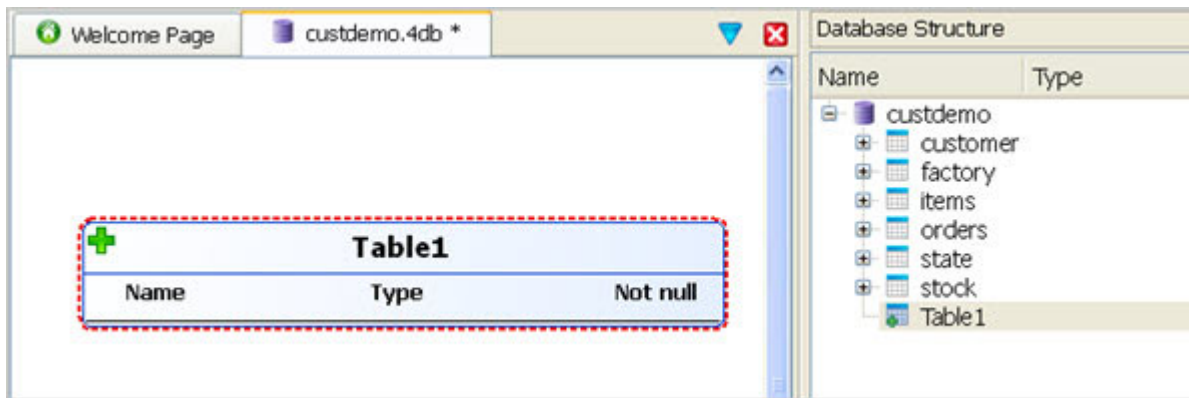


Figure 7-14. Adding a table

Add a column

Right-click on a new or existing table to add a column, providing the name, data type, and other properties in the Properties view.

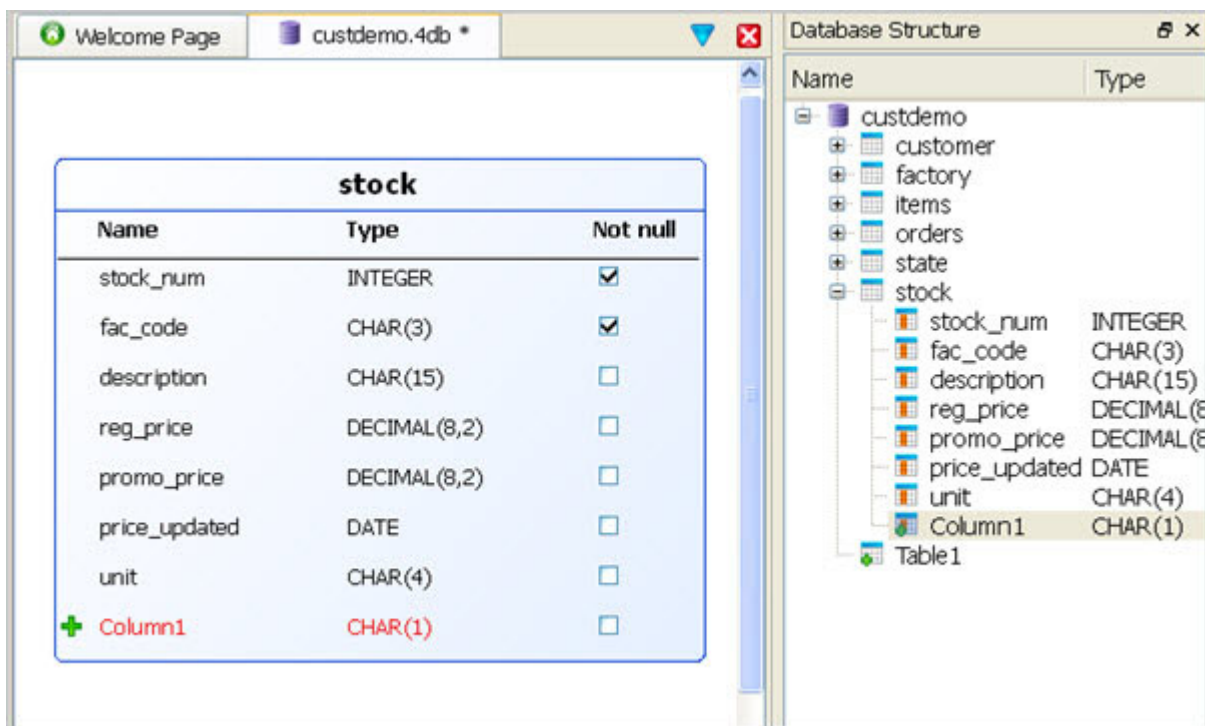


Figure 7-15. Adding a column

Revert changes

When you alter the schema by modifying an existing element, the original value is stored and used automatically if you revert the schema. Once changes have been made, you can select the modified element in the Structure view or the Document view, and choose the **Revert** option. The schema will be restored to its status when the schema was last extracted.

Missing status

If a Meta-schema has been updated and you choose to merge the result with an existing Meta-schema, any database objects from the original Meta-schema that no longer exist will have their status set to **Missing**. For example, if you have defined a foreign key for a table that no longer exists, the previous definition of the table will continue to be listed, but marked as missing. Messages alerting you to the missing tables and other database objects will be displayed in the Output view.

Database objects with a status of **Missing** can be removed from the Meta-schema. Deleting a missing table will also remove all foreign keys referencing this table.

Filter View

The right-click menu option **Show Filter View** displays a list of the objects displayed on the Database Schema view. Use the checkboxes to specify whether an object is shown or hidden.

Extracting the Meta-schema from a Database

Topics:

- Extracting the Meta-schema Information
 - Meta-schema File
 - Connection Information
 - Extraction Option Information
- Database Server Information for each Database Type
 - Genero db
 - IBM DB2®
 - Informix
 - MySQL
 - Oracle
 - PostgreSQL
 - SQLServer
 - SQLite

See also: The Meta-Schema File (.4db), Genero Studio Meta-schema Manager

Extracting the Information

To extract a Studio Meta-schema file (.4db) from a database, you must have access and permissions for the database. If you have trouble connecting to a database, make sure the database and the corresponding database client software are installed and configured properly.

The Studio main menu option **Database>>New Schema** guides you through the creation process. On the same menu, The Advanced Properties choice allows you to specify the extraction and generation options for the extracted database schema or generated database creation script.

Meta-schema File

Use the browse icon to search for the destination directory. Enter the name and path for the new schema file:

- **Insert the file in the project**- check this box to add the .4db file in the project. Select the project node where the .4db file should be added.
- **Merge with existing schema** - check this box if you have previously created a schema and wish to update it with the latest database information. Any changes that you made or user-defined items that you created will be preserved.

Click the **Next** button to continue.

Connection Information

This connection information is only used to extract the information for the Meta-schema file from the referenced database.

Database Connection Information - select one of the following:

- **Use explicit settings**
 - You can use a **previous connection** that was created for the same database. The combobox provides a list of the existing connections. Genero Studio will use the schema name that you entered to check for any related entry in the FGLPROFILE configuration file, and will use those values to define the connection. See The FGLProfile file in the Genero BDL manual.

or

You can enter the **Database Type** by selecting the desired type from the combobox, and the corresponding information for that type:

Database driver - the driver for the database type is automatically entered. If other drivers exist, they are available in the combobox.

The additional database server and Database User information that is required varies based on the database type:

- Genero db
 - IBM DB2
 - Informix
 - MySQL
 - Oracle
 - PostgreSQL
 - SQLServer
 - SQLite
- **Use external settings** - information in the FGLPROFILE configuration file is used to extract the corresponding connection information for the specified database.

Database User Information - the required information varies based on the database type selected.

When the information is complete, use the **Test Connection** Button to verify that the information is correct and that you are able to access the database.

Click the **Next** button to continue.

Extraction Options

Select the options for the Meta-schema file (.4db).

Click **Finish** to begin the extraction process.

The extracted Meta-schema is displayed in a tab in the Studio Project Manager work area. If you didn't already do so, save the Meta-schema file in a node in the project; the Meta-schema will be added to the DB Schemas tab in the Project.

Any application that uses the Meta-schema must have a dependency to the node where the schema was added. See Adding the Meta-Schema to your Application.

Database Server and User Information for each Database Type

Genero db

- the **ODBC Data Source** - name of the data source
- the **Schema name** - name of the schema
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

IBM DB2

- the **ODBC datasource** - the name of the DataSource that has been previously created in ODBC
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

Informix

- **Informix server** - name of the Informix database server instance (the Informix environment that contains the database to which you wish to connect). This is the same as the value of the INFORMIXSERVER variable.
- the **Database name** - Informix database name
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

Note: For Informix SE, the system variable **DBPATH** must be set.

MySQL

- the **Host name or IP address** - where the database is located
- the **Port** - port number used to access the database host
- the **Database name** - MySQL database name
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

Note: If you are using a remote MySQL database and a dynamic runner, the following environment variables must be set in Configurations:

- **MYSQLDIR**=<mysql installation directory>
- **MYSQL_UNIX_PORT**=\$MYSQLDIR/mysql.sock
- **MYSQLPORT**=<port used for mysql socket>

Oracle

- the **TNS name** - Oracle TNS service name
- the **Schema name** - name of Oracle schema
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

PostgreSQL

- the **Host name or IP address** where the database is located
- the **Port** - port number used to access the database host
- The **Database name** - PostgreSQL database name
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

SQL Server

- the **ODBC datasource** - the name of the DataSource that has been previously created in ODBC
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

SQLite

- **Database file** - the name of the file that contains the database
- **User name** - user name used for connection, or blank if not required
- **User Password** - user password for the connection, or blank if not required

Chapter 8. Graphical Debugger

Topics:

- “Genero Studio Graphical Debugger”
- “Testing your Program” on page 8-2
- “Breakpoints” on page 8-3
- “History View” on page 8-5
- “Watch Points” on page 8-5
- “Data View” on page 8-7
- “Backtrace Tab” on page 8-9
- “Debugger Commands” on page 8-9

Genero Studio Graphical Debugger

The **Graphical Debugger** allows you to test and control the behavior of your Genero application.

Topics:

- Local or remote Debug
- Launch the Debugger
- Open/Edit the Source File
- The Output View

Local or remote Debug

Check your active configuration before beginning a debug session.

- working **locally** - use a **local** Genero configuration to work on your local application.
- working on **remote** resources - use a **remote** Genero configuration or your **local** Genero configuration to access and debug your remote application.

Launch the Debugger

Graphical Debugger can be launched from within Project Manager:

- **Right-click a program node** and select **Debug Application** from the contextual menu; this will launch the selected application in debug mode.
- **Use the Debug toolbar icon**; this will launch the default application.

If the Debug option is not enabled, check the following:

- Is the source file directory physically available (delete / rename) ?
- Are you using the correct configuration to access this project ? (local / remote) ?

Once the debugger is launched, the program begins execution. If you have set a breakpoint in a Genero file (.4gl), execution will stop at that breakpoint, and the source file containing the breakpoint is opened in the Document Window.

Open/Edit the Source File

Sources are displayed in Code Editor. Search and select functions are restricted to modules used in the current debugging session.

You can navigate through the functions, and create and manage breakpoints on functions and code lines. Once breakpoints are set in a document, they will be available each time you open the document.

You can choose and group together any variables you specifically want to watch. This gives you the ability to follow a number of variables easily, and even to alter their values while the application is running, for testing purposes.

Functions

Once a source code module is open, its functions are displayed in the Code Structure view. Expand a function using the + icon to display the variables and records for that function.

Run the debug session

Use the commands on the **Debug menu** or the toolbar icons to control the execution of the program during the debug session.

The Output View

The Output view contains tabs that provide additional information when the program is being debugged, and allow you to enter commands or set values for variables:

- **Command** - to enter a new debug command manually. This allows you to enter a debug command and see the results listed immediately after the command.
- **Data** - displays values from global, module, functions and user-set variables, each category in its own column.
- **Breakpoints** - displays information on all breakpoints that have been set.
- **Backtrace** - to view or trace functions called while running the program.
- **Watchpoints** - displays information on all watch points that have been set.

Conditional breakpoints or conditional watchpoints can be set by entering a boolean expression in the Edit Expression Dialog.

Testing your Program

Topics:

- Run the Program
- Exit the Debugger

Run the Program

Commands on the **Debug menu** allow you to execute the application in debug mode. These commands are also available from the toolbar.

- **Debug** begins program execution, debugging up to the first breakpoint or interruption.
- **Next** executes the current line, and stops at the next source line in the **current** function. If this source code line is a call to another function:
 - the **Step In** command allows you to "step into" that function, executing the next source line inside the **called** function.
 - the **Step out** command allows you to step out of the called function before it ends, returning to the next source line following the CALL statement.

Note: Some complex instructions such as **CONSTRUCT** or **FOR** loops need two steps before going to the next instruction: one step to prepare the statement, one step to execute it.

- **Continue** resumes program execution, until another breakpoint is reached or the program terminates.
- **Interrupt** stops the execution of the program being debugged. This allows the user to interrupt a program that is in an endless loop, or that is displaying a form, for example, returning control to the debugger.
- **Up** displays the calling function. The up command advances toward the outermost frame each time it is processed. The Data tab display is based on the new context.
- **Down** is the opposite of *Up*: it comes back to the inner frame down to the last executed line.
- **Abort Last Task** terminates the debugger session.

These commands and other Debugger commands can also be entered manually in the Command tab of the Output view.

You can watch the value of specific variables as the program executes (Watchpoint view), and include the variables in a User variables page (Data view) where you can change the value for testing purposes.

Exit the Debugger

Choose **Abort last Task** from the **Debug** menu or toolbar to quit the Graphical Debugger. The Debugger automatically exits when the program being debugged terminates.

Breakpoints

Topics:

- Set a Breakpoint
- Breakpoints: Contextual menus
- The Breakpoint View
 - Conditions

Set a Breakpoint

To examine an area of code more closely, set a breakpoint at the desired line or function. A right-click on the function name invokes the contextual menu, which allows you to set, disable and delete breakpoints.

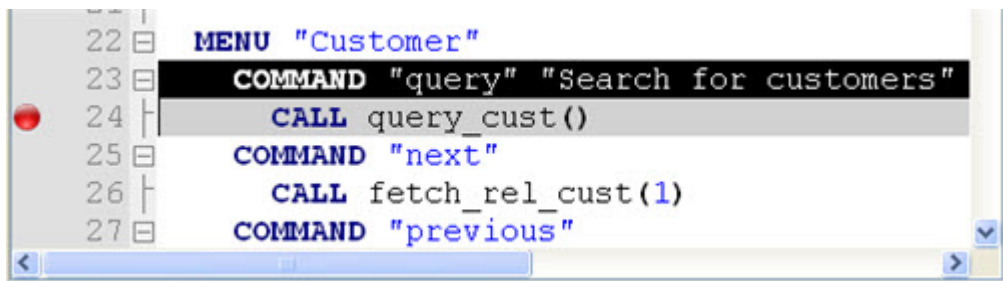


Figure 8-1. Set Breakpoint

The breakpoint line is highlighted, and there is a breakpoint icon in the gutter. When the program is run, execution will stop at this line, waiting for your action. Breakpoints are processed as private to each user. They are linked to a source code line, not to a line number; therefore, altering a source file will not affect the breakpoint's relative position in functions.

Breakpoint are stored when the Debugger is stopped, and set when the Debugger is started. Previously set breakpoints will be available each time the Debugger is run.

Enabled breakpoints are marked by a red dot; the dot has an empty center if the breakpoint is disabled.

Tip: Clicking in the gutter will also set / remove a breakpoint at the corresponding code line.

Breakpoints: contextual menus

Breakpoints also can be set and managed from the contextual menu in the main window:

- **Add/Delete breakpoint** - sets / removes a breakpoint for the current line.
- **Enable/Disable breakpoint** - If checked, the current breakpoint is active; otherwise it is disabled, but still exists and can be reactivated later.

The Breakpoints View

This Breakpoints view displays information on all breakpoints that have been set:

- The **Id** column displays the number assigned to the breakpoint by the Debugger.
- The **File, Line, and Function** columns identify the breakpoint's location in the source file.
- You can enable or disable the breakpoint by checking/unchecking the box in the **Enable** column.
- **Conditions** may be specified in the Condition column.
- **Hit Count** is the total number of times a Breakpoint has been hit
- **Skip Hits** specifies the number of times the breakpoint should be ignored

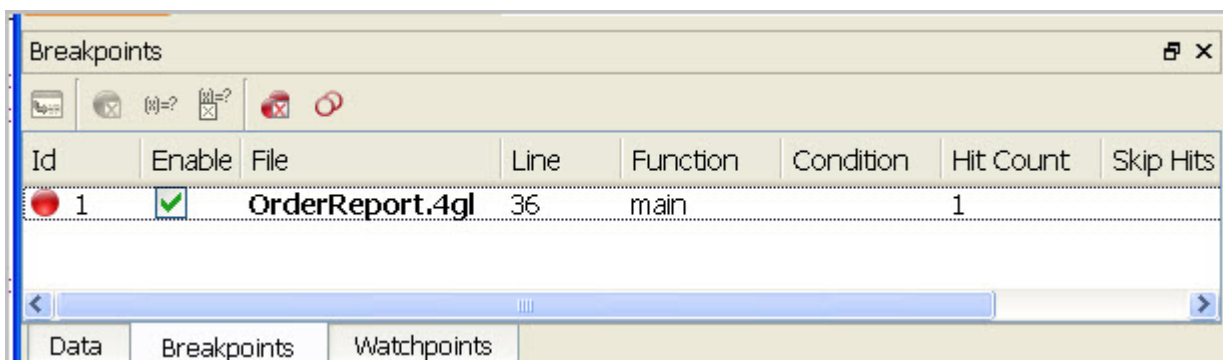


Figure 8-2. Breakpoints tab

Double-click a line in this window to scroll the source file and display the corresponding function.

Right-click a line in this window to invoke the contextual menu, containing options to delete the breakpoint, and to edit or remove a condition. You can also delete or disable all breakpoints.

Conditions

You can specify that the breakpoint is conditional, using a Boolean expression to determine whether to stop execution of the program. in the **Condition column** of the Breakpoints tab, enter the condition or press the ... button to open the Edit Expression dialog.

If a condition is defined, it will be checked each time the line is to be executed. Execution will stop at this breakpoint only if the condition is true.

Example:

`i > 50`

History View

The History tab is used to display all debug commands executed by the debugger, with their results. Program lines that are executed are displayed.

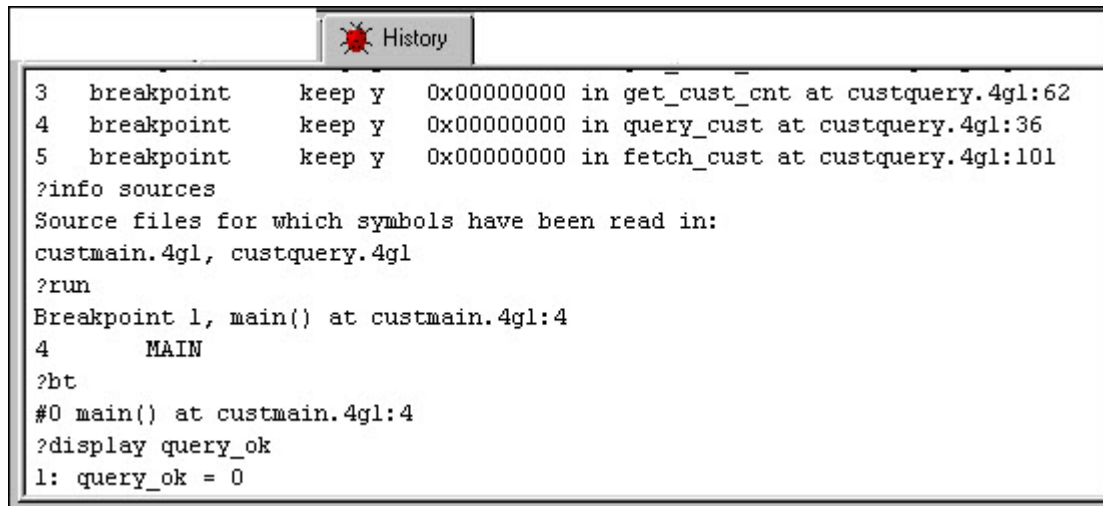


Figure 8-3. History view

Watch Points

Topics:

- Setting a Watchpoint
- Conditional Watchpoints
- The Watchpoints View

Watchpoints can be set to stop program execution each time the value of an expression changes. In order to set the watchpoint, the program must be running in the Debugger.

Setting a WatchPoint

1. Set a breakpoint in the function that has the variable expression in scope.
2. Start debugging the program.
3. When the program stops at the breakpoint, click the Data view to display the variables.
4. Right-click the variable to display the context menu. Choose **Add to watch** to add the variable to the Watchpoints view.

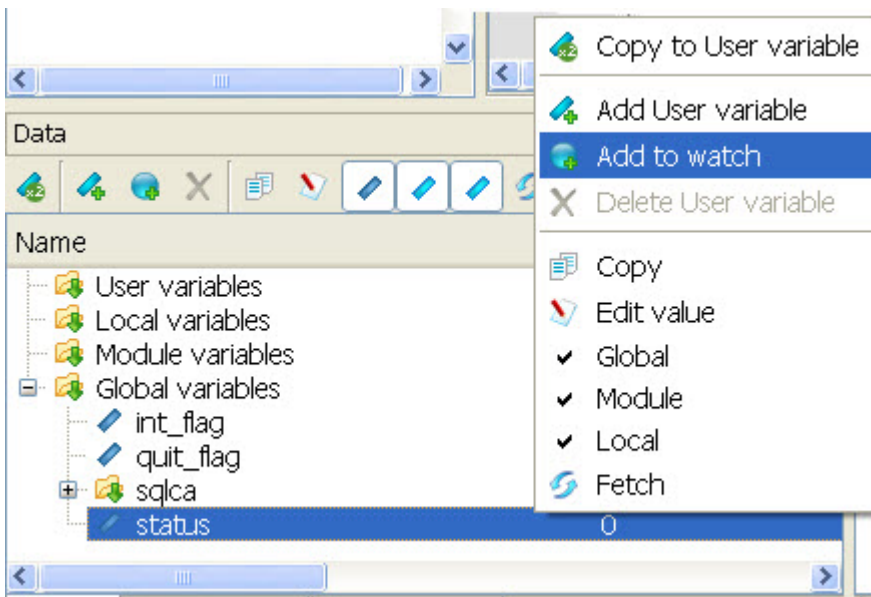


Figure 8-4. Setting a WatchPoint

5. Continue running the program. The program will stop when the value of a watched variable changes. You can see the values in the Watchpoints view. The icons at the top of the view allow you to choose to watch all local variables, module variables, or global variables.

Setting a watchpoint from the Command view

You can set a watchpoint using Debugger commands in the Command view.

1. Set a breakpoint in the function that has the variable expression.
2. Start debugging the program.
3. When the program stops at the breakpoint, enter the following command in the Command view:
`watch expression`
4. Continue running the program. The program will stop when the value of a watched variable changes. You can see the values in the Watchpoints view.

Conditional Watchpoints

You can specify that the watchpoint is conditional, using a Boolean expression to determine whether to stop execution of the program when the variable value has changed. Each time the value of the watched variable changes (and the BOOLEAN expression, if used, is TRUE), the program will stop, and the variable values will be displayed Watchpoints view.

In the **Condition** column of the Watchpoints view, enter the condition or press the ... button to open the Edit Expression dialog.

Or, you can enter the watch Debugger instruction and condition in the Commands view.

```
watch expression boolean-expression
```

Example:

```
watch i if i > 3
```

The Watchpoints View

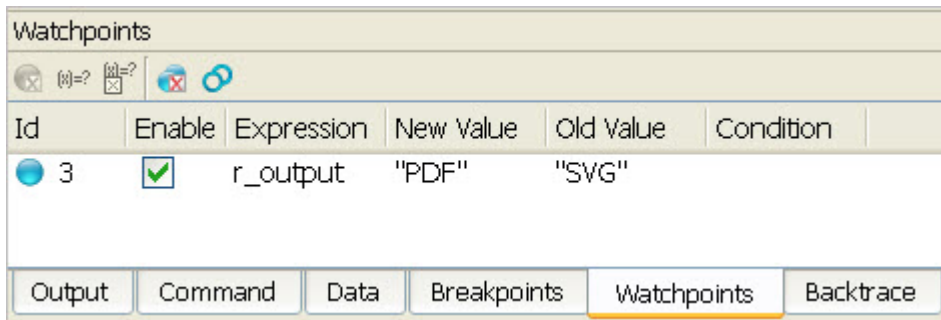


Figure 8-5. Watchpoints View

Each time the value of a watched variable changes, the program will stop, and the variable values will be displayed in the Watchpoints view. The columns in the table are:

- **ID** - the identification number of the watchpoint
- **Enable** - the state of the watchpoint (Enabled/Disabled); you can change the state by clicking the checkbox
- **Expression** - normally the variable name on which the watchpoint is set
- **New value** - the new value of the watchpoint variable
- **Condition** - a Boolean expression can specify that the watchpoint is conditional; each time the value of the variable changes and the condition is TRUE, the program will stop.

A watchpoint can be enabled/disabled by selecting the Enable checkbox. Select a watchpoint in the table, and right-click to display a menu that allows you to delete that watchpoint or all watchpoints, and edit the condition.

Setting a watchpoint describes the steps to set a watchpoint.

Data View

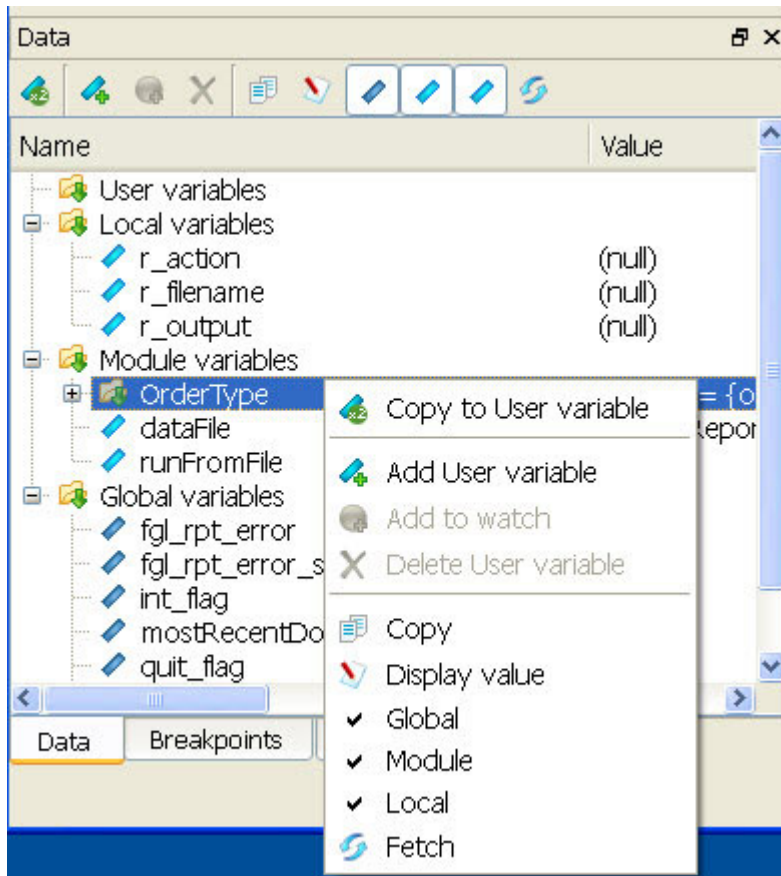


Figure 8-6. Data View

In the Data view you can examine or set the values of global, module, and function variables while your application is running. Variables are organized into groups and displayed in a tree. Expand the group to display its variables.

For each of these variables, the name, value, and type (record, column, data type) are listed. Double-click the value to edit variable values for testing purposes.

Use the integrated toolbar to:

- **Copy** a variable to the User variables group
- **Add, Delete, or Duplicate** a User variable
- **Add** a variable to watch list
- **Display** a variable value
- **Show** Global, Module, and/or Local variables
- **Fetch All** variables

Right-click a variable to display a context menu with some of the same commands.

Groups of variables

- **Global variables** show user-defined global variables as well as any Genero predefined variables used in the application, such as `INT_FLAG`.
- **Module variables** show the variables defined with module scope.

- **Local variables** show the variables defined with local scope for the currently executing function. Local variables are separated from argument for better visibility
- **User variables** allow you to group together variables from any of the other folders, for convenience in testing. To add a variable to the User variables set, or delete an existing User variable, right-click the variable listing to invoke the contextual menu.

User variables which are not in the scope of the current context are greyed.

Note: Double-click on the title bar of the window to dock/undock the window.

Backtrace Tab

In the Backtrace view you can view or trace the functions called while running the program.

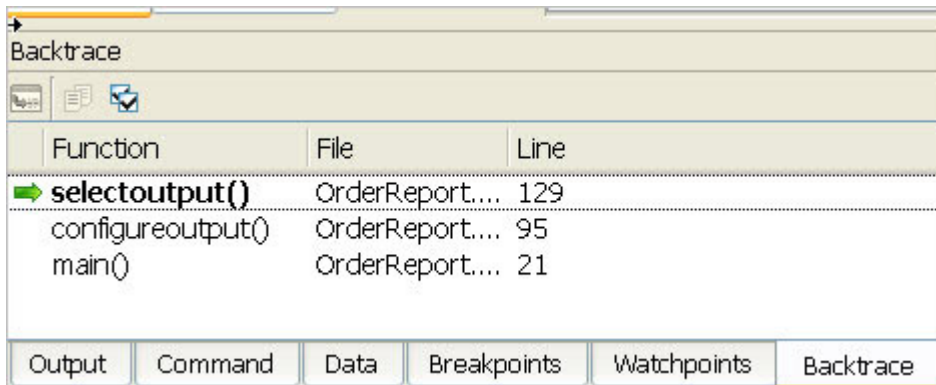


Figure 8-7. Backtrace view

The first line displayed is the current line. Then the calling function is displayed, and so on, up to the main function. The Backtrace view is display-only. Lines are added in the output while the program is running.

An arrow points to the currently executing function.

Debugger Commands

Topics:

- Command View
- Record commands
- Command list

Command View

All the debugger commands that have been executed display in the Command view. Debugger commands may also be entered manually.

The output of the command, if any, appears immediately below the command.

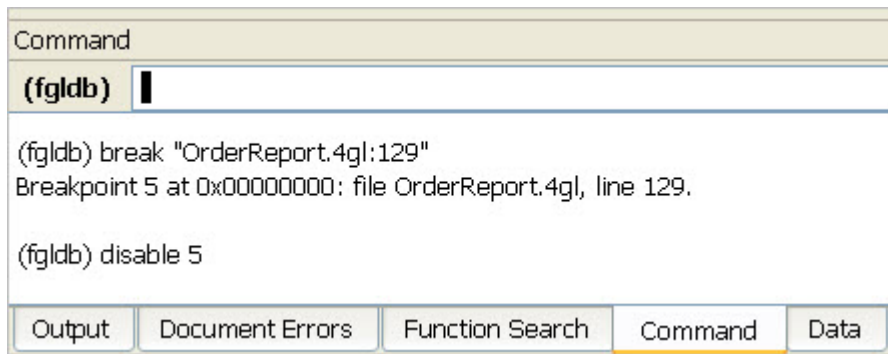


Figure 8-8. Command view

Record commands

Options on the Debug menu and toolbar icons allow you to record the debugger commands that you are executing, save them in a file, and to execute the commands again from that file.

- **Record macro** - start recording debugger commands.
- **Stop record** - stop command recording and store the commands in a file.
- **Play macro** - execute the recorded commands from the file.

Debugger Command List

Supported Commands

Table 8-1. Supported Commands

Command	Description
break	Sets a breakpoint at the specified line or function
backtrace	Prints a summary of how your program reached the current state.
clear	Clear breakpoint at the specified line or function
continue	Continues the execution of your program
define	Allows you to specify a user-defined sequence of commands
delete	Removes a breakpoint or auto-display expressions.
disable	Causes a breakpoint to be ignored
display	Prints the values of the expressions in a list each time the program stops
down	Selects and prints the stack frame called by this one
enable	Reactivates breakpoints that have previously been disabled.
file	Uses FILE as program to be debugged

Table 8-1. Supported Commands (continued)

Command	Description
finish	Continues until the current stack frame returns
help	Displays information about debugger commands
info	Provides information about the status of the program
list	Shows source code
next	Continues with the next source code line at the same level
output	Prints only the value of the specified expression, suppressing any other output
print	Displays the current value of the specified expression
quit	Leaves the debugger
run	Starts the program
set	Sets the value of specific variables in your program
source	Executes a file of debugger commands
signal	Sends an INTERRUPT signal to your program.
step	Continues with the next source code line and stops
tbreak	Sets a temporary breakpoint
tty	Specifies where your program should do input and output
until	Continues running until a specified location is reached
up	Selects and prints stack frame that called this one
watch	Set a watch point for an expression. A watch point stops the execution of your program whenever the value of an expression changes.
where	An alias for the command backtrace

break

This command defines a breakpoint to stop the program execution at a given line or function.

Syntax

```
b[reak] { line | module:line | function }
```

Note:

1. *line* is a source code line in the current module.
2. *module:line* is a source code line in a specific module.
3. *function* is a function name.

Usage

The break command sets a breakpoint at a given position in the program.

When the program is running, the debugger stops automatically at breakpoints defined by this command.

Example

```
(fgldb) break mymodule:5
Breakpoint 2 at 0x00000000: file mymodule.4gl, line 5.
```

backtrace

This command prints a summary of how your program reached the current state.

Syntax

```
backtrace
bt
```

Usage

The backtrace command prints a summary of your program's entire stack, one line per frame. Each line in the output shows the frame number and function name.

Example

```
(fgldb) bt
#1 addcount() at mymodule.4gl:6
#2 main() at mymodule.4gl:2
(fgldb)
```

clear

This command clears the breakpoint at a specified line or function.

Syntax

```
clear [function | line]
```

Note:

1. *function* - deletes any breakpoints set at entry to the specified function.
2. *line* - deletes any breakpoints set at or within the code of the line specified by number.

Usage

With the clear command you can delete specific breakpoints in your program. Use the clear command with no arguments to delete any breakpoints at the next instruction to be executed in the selected stack frame.

See the delete command to delete individual breakpoints by specifying their breakpoint numbers.

Example

```
(fgldb) clear mymodule:5
Deleted breakpoint 2
(fgldb)
```

continue

This command continues the execution of the program after a breakpoint.

Syntax

c[continue] [*ignore-count*]

Note:

1. *ignore-count* defines the number of times to ignore a breakpoint at this location.

Usage

The continue command continues the execution of the program until the program completes normally, another breakpoint is reached, or a signal is received.

Example

```
(fgldb) continue
<..program output..>
Program exited normally.
```

define

This command allows you to specify a user-defined sequence of commands.

Syntax

```
define command-name { command1 command2 ... }
end
```

Note:

1. *command-name* is the name assigned to the command sequence.
2. *command* is a valid debugger command.
3. **end** indicates the end of the command sequence.

Usage

The define command allows you to create a user-defined command by assigning a command name to a sequence of debugger commands that you specify. You may then execute the command that you defined by entering the command name at the debugger prompt.

User commands may accept up to ten arguments separated by whitespace.

Example

```
(fgldb) define myinfo
> info breakpoints
> info program
> end
(fgldb)
```

delete

This command allows you to remove breakpoints that you have specified in your debugger session.

Syntax

del[ete] *breakpoint*

Note:

1. *breakpoint* is the number assigned to the breakpoint by the debugger.

Usage

The delete command allows you to remove breakpoints when they are no longer needed in your debugger session.

If you prefer you may disable the breakpoint instead. See the disable command.

Example

```
(fgldb) delete 1
(fgldb) run
Program exited normally.
(fgldb)
```

disable

This command disables the specified breakpoint.

Syntax

`disable breakpoint`

Note:

1. *breakpoint* is the number assigned to the breakpoint by the debugger.

Usage

The disable command instructs the debugger to ignore the specified breakpoint when running the program.

Use the enable command to reactivate the breakpoint for the current debugger session.

Example

```
(fgldb) disable 1
(fgldb) run
Program exited normally.
(fgldb)
```

display

This command displays the specified expression's value each time your program stops.

Syntax

`disp[lay] expression`

Note:

1. *expression* is your program's expression that you wish to examine.

Usage

The display command allows you to add an expression to an automatic display list. The values of the expressions in the list are printed each time your program stops. Each expression in the list is assigned a number to identify it.

This is useful in tracking how the values of expressions change during the program's execution.

Example

```
(fgldb) display a
1: a = 6
(fgldb) display i
2: i = 1
(fgldb) step
2: i = 1
1: a = 6
16      for i = 1 to 10
(fgldb) step
2: i = 2
1: a = 6
17      let a = a+1
(fgldb)
```

down

This command selects and prints the function called by the current function, or the function specified by the frame number in the call stack.

Syntax

```
do[wn] [num]
```

Note:

1. *num* is the number of frames to move down the stack. The default is 1.

Usage

This command moves down the call stack, to the specified frame, and prints the function identified with that frame. To print the function called by the current function, use the down command without an argument.

```
(fgldb) down
#0 query_cust() at custquery.4gl:22
22  CALL cleanup()
(fgldb)
```

enable

This command enables breakpoints that have previously been disabled.

Syntax

```
enable breakpoint
```

Note:

1. *breakpoint* is the number assigned to the breakpoint by the debugger.

Usage

The enable command allows you to reactivate a breakpoint in the current debugger session. The breakpoint must have been disabled using the disable command.

Example

```
(fgldb) disable 1
(fgldb) run
Program exited normally.
(fgldb) enable 1
(fgldb) run
Breakpoint 1, at mymodule.4gl:5
```

file

This command allows you to specify the name of the program being debugged.

Syntax

`file filename`

Note:

1. *filename* is the name of the program being debugged.

Usage

The `file` command can be used to change to a different file during a debugging session.

finish

Continues until the current stack frame returns.

Syntax

`finish`

Usage

The `finish` command instructs the program to continue running until just after the function in the selected stack frame returns, and then stop. The returned value, if any, is printed.

Example

```
(fgldb) finish
```

help

This command provides information about debugger commands.

Syntax

`h[elp] [command]`

Note:

1. *command* is the name of the debugger command for which you wish information.

Usage

The `help` command displays a short explanation of a specified command.

Enter the `help` command with no arguments to display a list of debugger commands.

Example

```
(fgldb) help delete
```

info

This command describes the current state of your program.

Syntax

`i[nfo] [{ breakpoints | sources | program | stack | files |
 line (linespec | module:line | function) }]`

Note:

1. **breakpoints** lists the breakpoints that you have set.
2. **sources** prints the names of all the source files in your program.
3. **program** displays the status of your program.
4. **stack** summarizes how your program got where it is.
5. **files** lists the names of the executable file and core dump files currently in use, and the files from which symbols were loaded.
6. **line** *linespec* maps the specified source code line to program addresses.
7. **line** *module:line* maps the specified source code line to program addresses.
8. **line** *function* prints the program addresses for the first line of the function named *function*.

Usage

The `info` command describes the state of your program.

The command `info line` *linespec* prints the starting and ending addresses of the compiled code for the source line specified. See the `list` command for all the ways that you can specify the source code line.

Example

```
(fgldb) info sources
Source files for which symbols have been read in:
mymodule.4gl, fglwinexec.4gl, fglutil.4gl, fgldialog.4gl, fgldummy4js.4gl
(fgldb)
```

list

This command prints source code lines of the program being executed.

Syntax

```
l[ist]
```

Usage

The `list` command prints source code lines of your program, including the current line.

Example

```
(fgldb) run
Breakpoint 1, at mymodule.4gl:5
5   call addlist()
(fgldb) list
5   call addlist()
6   call addname()
.
14  end function
(fgldb)
```

next

This command continues running the program by executing the next source line in the current stack frame, and then stops.

Syntax

```
n[ext]
```

Usage

The next command allows you to execute your program one line of source code at a time. The next command is similar to step, but function calls that appear within the line of code are executed without stopping. When the next line of code at the original stack level that was executing when you gave the next command is reached, execution stops.

After reaching a breakpoint, the next command can be used to examine a troublesome section of code more closely.

Example

```
(fgldb) next
5 call addlist()
(fgldb) next
6 call addname()
(fgldb)
```

output

This command prints only the value of the specified expression, suppressing any other output.

Syntax

`output expression`

Note:

1. *expression* is your program's expression that you wish to examine.

Usage

The output command prints the current value of the expression and nothing else, no newline character, no "expr=", etc.

The usual output from the debugger is suppressed, allowing you to print only the value.

Example

```
(fgldb) output b
```

print

This command displays the current value of the specified expression.

Syntax

`p[_rint_] expression`

Note:

1. *expression* is your program's expression that you wish to examine.

Usage

The print command allows you to examine the data in your program.

It evaluates and prints the value of the specified expression from your program, in a format appropriate to its data type.

Example

```
(fgldb) print b
$l=5
(fgldb)
```

quit

This command terminates the debugger session.

Syntax

```
q[uit]
```

Usage

The quit command allows you to exit the debugger.

Example

```
(fgldb) quit
<system prompt>
```

run

This command starts your program.

Syntax

```
r[un] [arg1 arg2 ... ]
```

Note:

1. *arg* is an argument to be passed to your program.

Usage

The run command causes your program to execute until a breakpoint is reached or the program terminates normally.

Example

```
(fgldb) run
Breakpoint 1, at mymodule.4gl:3
3      call addcount()
(fgldb)
```

set

This command allows you to set the value of specific variables for the duration of your debugger session.

Syntax

```
set { [environment] varname[=value] [verbose {on | off}] [annotate ]
```

Note:

1. *varname* is the environment variable to be set to *value*.

Usage

The set command changes the values of these variables:

When setting an **environment** variable, *value* may be any string. If the *value* parameter is omitted, the variable is set to a null value. The variable is set for your program, not for the debugger itself.

When **verbose** is set to **on**, the debugger will display additional messages about its operations, allowing you to observe that it is still working during lengthy internal operations.

Example

```
set verbose on
```

Important: On UNIX systems, if your SHELL variable names a shell that runs an initialization file, any variables you set in that file affect your program. You may wish to move setting of environment variables to files that are only run when you sign on, such as `.login` or `.profile`.

source

This command executes a file of debugger commands.

Syntax

```
source commandfile
```

Note:

1.

commandfile is the name of the file containing the debugger commands.

Usage

The source command allows you to execute a command file of lines that are debugger commands. The lines in the file are executed sequentially.

The commands are not printed as they are executed, and any messages are not displayed. Commands are executed without asking for confirmation.

An error in any command terminates execution of the command file.

Example

```
(fgldb) source mycommands
```

signal

This command sends an INTERRUPT signal to your program.

Syntax

```
sig[nal] SIGINT
```

Usage

The signal SIGINT command resumes execution of your program where it has stopped but immediately sends an INTERRUPT signal. The source line that was current when the signal was received is displayed.

Example

```
(fgldb) signal SIGINT
Program exited normally.
16      for i = 1 to 10
(fgldb)
```

step

This command continues running the program by executing the next line of source code, and then stops.

Syntax

`s[tep] [count]`

Note:

1. *count* defines the number of lines to execute before stopping.

Usage

The `step` command allows you to "step" through your program executing one line of source code at a time. When a function call appears within the line of code, that function is also stepped through.

A common technique is to set a breakpoint prior to the section or function that is causing problems, run the program until it reaches the breakpoint, and then step through it line by line.

Example

```
(fgldb) step
4 call addlist(a)
(fgldb)
```

Important: The `step` command cannot be used to step through a function that was compiled without debugging information. Execution continues until it reaches a function that does have debugging information.

until

This command continues running the program until the specified location is reached.

Syntax

`u[ntil] { line | module:line | function }`

Note:

1. *line* is a source code line in the current module.
2. *module:line* is a source code line in a specific module.
3. *function* is a function name.

Usage

The `until` command continues running your program until either the specified location is reached, or the current stack frame returns. This can be used to avoid stepping through a loop more than once.

Example

```
(fgldb) until addcount()
```

up

This command selects and prints the function that called this one.

Syntax

up [num]

Note:

1. *num* is the number of lines to move up the stack. The default is 1.

Usage

The up command advances toward the outermost frame, to frames that have existed longer.

watch

This command watch sets a watch point for an expression. A watch point stops execution of your program whenever the value of an expression changes.

Syntax

watch *expression* [boolean-expression]

Note:

1. *expression* is the expression to watch.
2. *boolean-expression* is an optional boolean expression.

Usage

The watch point stops the program execution when the value of the expression changes.

If *boolean-expression* is provided, the watch point stops the execution of the program if the expression value has changed and the *boolean-expression* evaluates to TRUE.

Example

```
(fgldb) watch i if i >= 3
```

where

The command where is an alias for backtrace.

Syntax

w[here]

Usage

The where command, like backtrace, prints a summary of your program's entire stack, one line per frame. Each line in the output shows the frame number and function name.

Example

```
(fgldb) where
#1 addcount() at mymodule.4gl:6
#2 main() at mymodule.4gl:2
(fgldb)
```

Unsupported Commands

These debugger commands are not supported in Genero Studio:

- tbreak

- source
- define
- tty

tbreak

This command sets a temporary breakpoint.

Syntax

`tbreak { line | module:line | function }`

Note:

1. *line* is a source code line in the current module.
2. *module:line* is a source code line in a specific module.
3. *function* is a function name.

Usage

The `tbreak` command sets a breakpoint for one stop only. The breakpoint is set in the same way as the `break` command, but the breakpoint is automatically deleted after the first time your program stops there.

Example

```
(fgldb) tbreak 12
Breakpoint 2 at 0x00000000: file custmain.4gl, line 12.
(fgldb)
```

tty

This command resets the default program input and output for future run commands.

Syntax

`tty filename`

Note:

1. *filename* is the file which is to be the default for program input and output.

Usage

The `tty` command instructs the debugger to redirect program input and output to the specified file for future run commands.

The redirection is for your program only; your terminal is still used for debugger input and output.

Example

```
(fgldb) tty /dev/ttyS0
(fgldb)
```

Chapter 9. Genero Report Writer

Topics:

- “Getting Started”
- “BDL Programs” on page 9-11
- “Modifying the Output” on page 9-33
- “Report Design Document” on page 9-43
- “Reference” on page 9-91
- “RTL Class Reference” on page 9-190

Getting Started

Topics:

- “Introduction to Reports”
- “Run Reports - the GRW Demo” on page 9-3
- “Steps to a Report” on page 9-6
- “Configuring Fonts and Printers” on page 9-9

Introduction to Reports

Topics:

- Overview
 - Report Creation
 - The Report Design Document
 - The Report Output
- How it Works

See also: Running the GRW Demo, Steps to a Report, The BDL File, Report Design

Overview

Genero Report Writer provides a GUI interface to design reports that are based on data generally retrieved from a database; the layout of the report is specified using the drag-and-drop report designer. Gathering the data for the report is separated from the layout, enhancing security.

Report Creation

- First you write a Genero BDL application (**.4gl** file) to retrieve the data, from a database for example. Genero BDL statements and Reporting API functions in your application get the report data and output it to Genero Report Writer. A great deal of the code to retrieve the data can be generated using the Report Data Wizard.
- Once the BDL file is created, you can use the **fglcomp** tool to extract the data schema for the report (**.rdd** file) from this file. The **.rdd** file is used to populate the Data View in the Genero Report Designer, and the resulting data objects can be dragged and dropped onto the design page.
- You design the report layout, using Genero Report Designer, dragging and dropping the report elements, including data objects, onto the report design page, to create a Report Design Document (**.4rp** file).

- You can specify the default report output options using the Report Designer menu; or, functions in the Genero report API can be called in your BDL application to provide finer granularity for the output options.
- End-to-end streaming (see How It Works) from the Genero DVM to the Genero Report Viewer provides "print-as-you-go" processing; the data is produced on the fly.

The Report Design Document

The Report Design Document (.4rp) specifies the appearance of the report and its data:

- Report elements (containers (such as Mini Page and Horizontal Box), and their child elements (such as Word Box . Image Box. and data objects), can be dragged and dropped onto the report page and re-arranged.
- Properties in the Properties View describe the report element. The default values for element properties can be changed, or calculated using expressions. An expression can be used to change the appearance of the element, or change it conditionally (for example, an expression might turn the background color red if the value for the report element is greater than 1000).
- The report design elements can contain values that are based on other values in the report (totals and subtotals, for example).
- Various Report Design Documents can be created using the data from the same .4gl file, and can be output in different formats.

See Designing a Report.

The Report Output

A Genero report is based on XML standards (XML reports, XML data, SVG output). Since the data is in XML format, you can have multiple views of the same data. The report can be output to various formats, or directly to a printer or other output devices.

You can use Genero Report Writer to create reports for a variety of purposes. For example, you can produce:

- Lists
- Reports with a customized format, such as invoices, corporate documents, or accounting reports, and that include images
- Reports on pre-printed forms
- Labels
- Business Graphs such as MapCharts, Category Charts, and XY Charts
- XML documents
- HTML documents
- Excel spreadsheets
- Documents in Microsoft RTF format
- ASCII-based reports to print legacy Genero reports "as is"

The rapid delivery of pages, with no need for temporary files or temporary tables, consumes less memory and speeds up the printing of very large reports.

See Steps to a Report for a complete outline of the report creation process.

See Configuring Fonts and Printers for tips about printers, fonts, and custom paper sources.

How It Works at Runtime

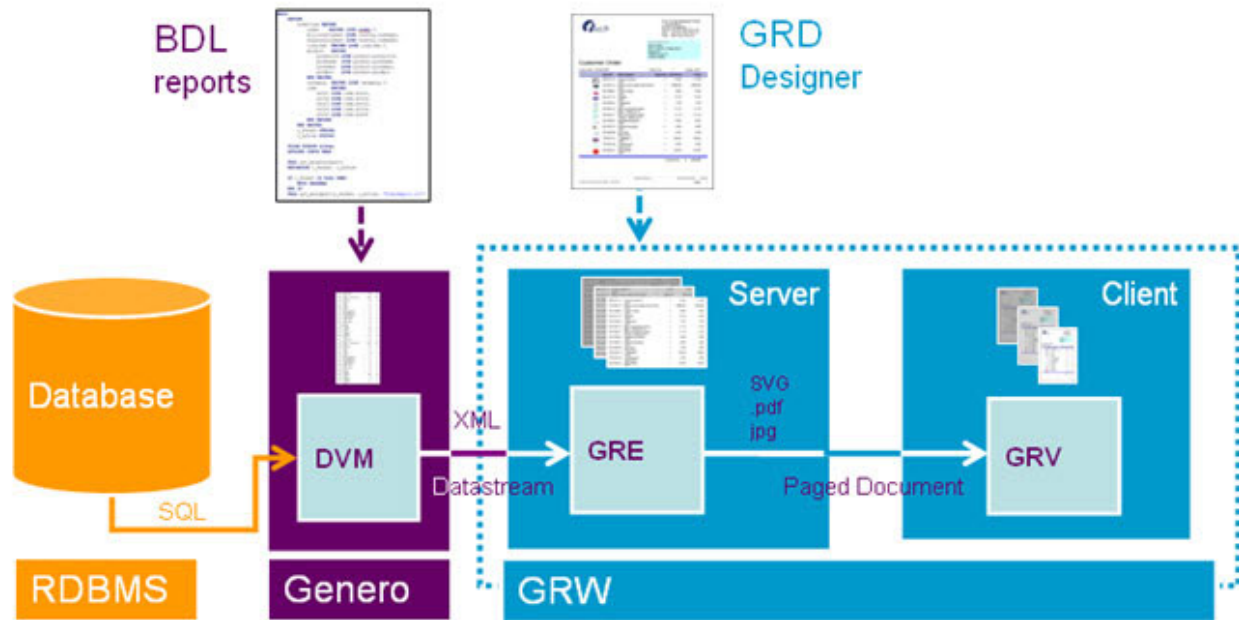


Figure 9-1. Genero Report Writer Workflow

The following Genero components are used to produce a report from Genero Report Writer:

- The Genero Business Development Language (BDL) defines the data needed from the database
 - **The Genero DVM (Dynamic Virtual Machine)** executes the BDL code to retrieve the data
 - **The Genero DVM** streams data from the database to the **Genero Report Engine (GRE)**
- The Genero Report Designer (GRD) graphically lays out the report
 - **The Genero Report Engine (GRE)** lays out paged streams to **Genero Report Viewer (GRV)**
- The Genero Report Viewer (GRV) (part of the **Genero Desktop Client**) displays the report on the client; alternatively, the report can be displayed as a PDF or Image, as an HTML or XML file, or output to an Excel spreadsheet.

Run Reports - the GRW Demo

Topics:

- Overview
- Demo Report Options
- Genero Report Viewer

See also: Steps to a Report, The BDL File

Overview

When Genero Studio launches, the Welcome Page lists the demo projects under Recent Projects. Select **Reports** to open the demo project in Studio Project Manager.

The GRW demo project provides a Reports project consisting of:

- BDL application files
- The corresponding **report definition files**, created using the Report Designer
- A **form definition file** for the forms used to prompt the user for the output format, created using the Studio Form Designer
- The .rdd file containing the data schema for the database used with the demo reports

The default GRW demo project, **Reports.4pw**, is in **Documents and Settings\<username>\My Documents\My Genero Files\ samples\Reports**.

To run the demo:

- Expand the **Reports** project in the Project Manager, and right click **Order Report**.
- Select **Execute** from the context menu.

Demo Report Options

When the report begins to run, BDL code in the demo program displays a dialog box to allow you to select the report you wish to run, and the output format:

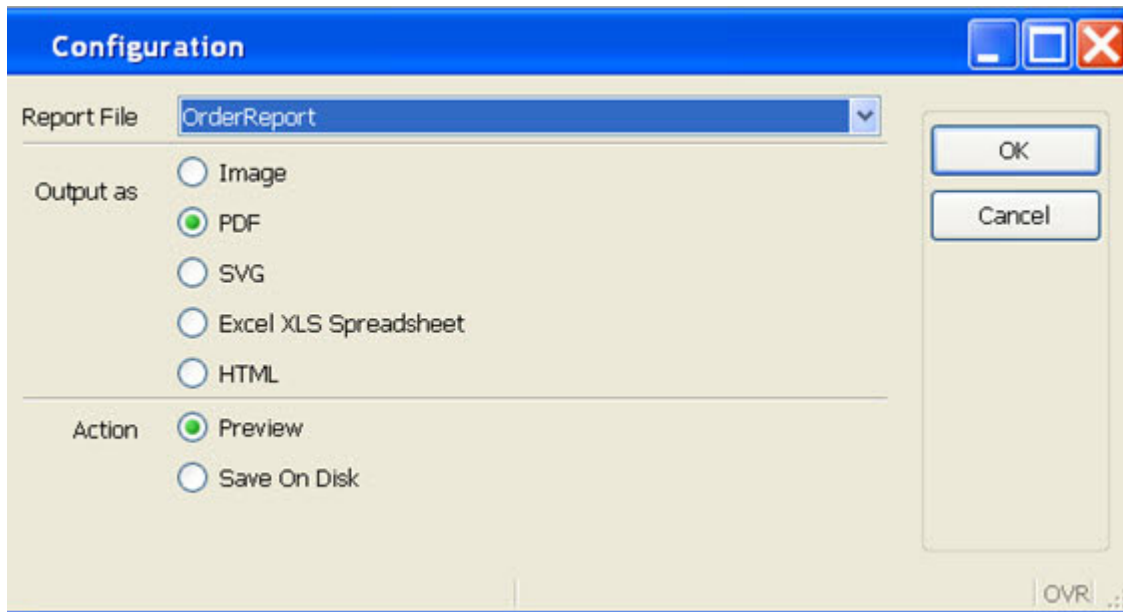


Figure 9-2. Reports Demo form

Select the desired report from the dropdown list box:

- **List Demo** - a report in list format
- **Order Report** - a customer order
- **Order Stock** - stock information including a bar code
- **Order Labels** - pages of mailing labels for customers
- A variety of **Map Charts** and **Category Charts**

Next, select the output format:

- **Image** - an image is created in the GRW demo project directory, in .jpg format by default.
- **PDF** - a pdf file will be created in the GRW demo project directory. The file can be viewed with an Acrobat PDF reader.
- **SVG** - an SVG file will be created in the GRW demo project directory. The Genero Report Viewer (a part of the Genero Desktop Client) can be used to preview and print the report.
- **Excel XLS Spreadsheet** - the report data is output in Excel format
- **HTML** - the report document is output in HTML format

Select the desired action:

- **Preview** - display an SVG format report in the Genero Report Viewer.
- **Save on disk** - save a copy of the report on disk. The report will be saved in the GRW demo project directory. The name of the report will be "report", with an extension of .pdf or .svg, or image000#.jpg where # is incremented sequentially from 1 for each successive image that is saved to disk.

Genero Report Viewer

If you choose **SVG** as the output format, and **Preview** as the action, the report displays in the Genero Report Viewer, which is automatically installed as part of the Genero Desktop Client:

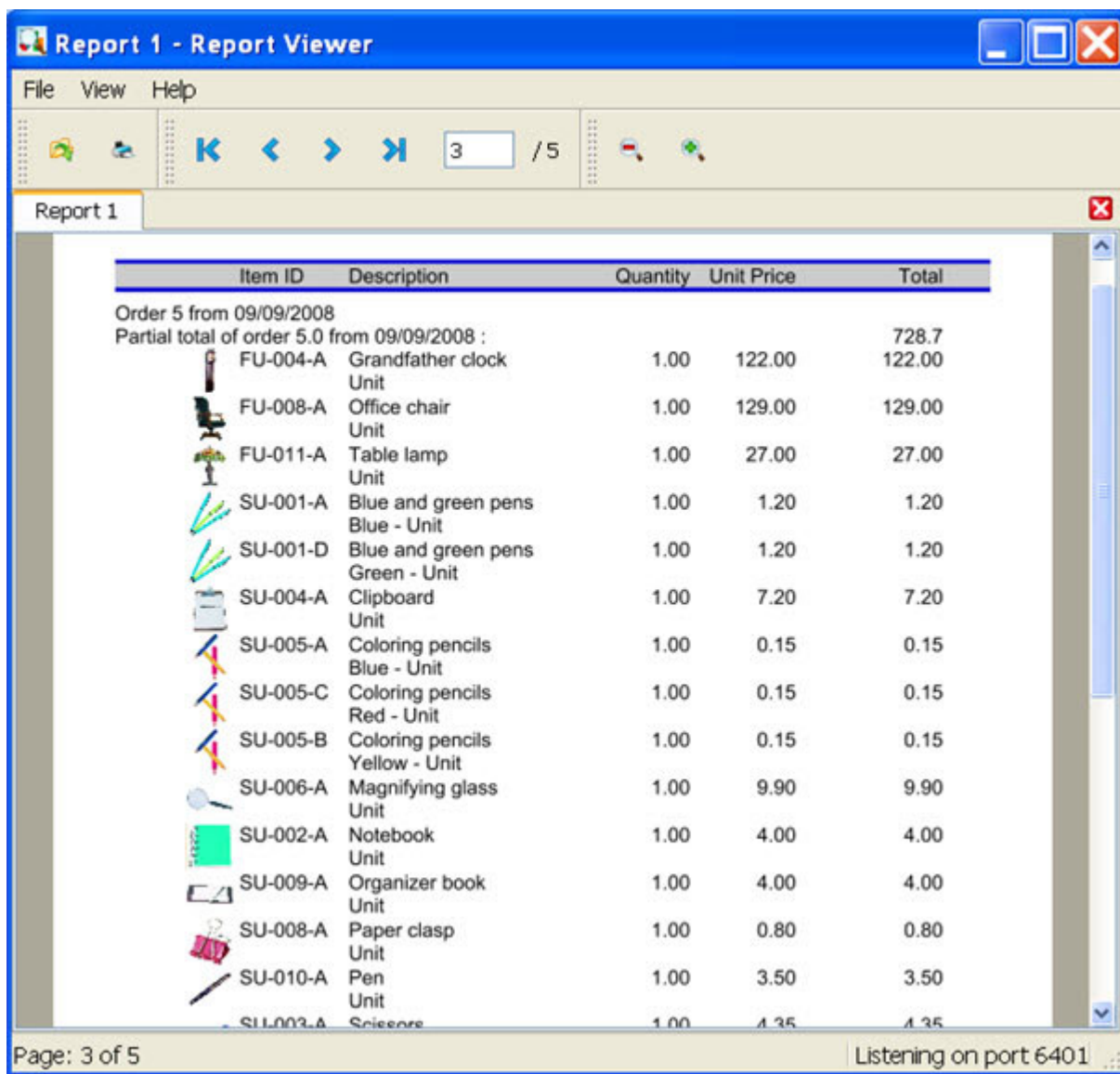


Figure 9-3. Genero Report Viewer

Use the **File>>Print** menu option in Report Viewer to send the report to a printer.

Steps to a Report

Topics:

- **Prerequisite**
 - “Set the Report Writer environment” on page 9-7
- **Creating a Report**
 1. Create a Project
 2. Create an Application
 - Add an application node
 - Add external dependencies
 3. Create the BDL file
 4. Add the Data Schema to the Project
 5. Create the Report Design Document

6. Build the Report Application in Project Manager
- **Running a Report**
 - Run the Report
 - The Report Viewer

See also: Introduction to Report Writer, The BDL file, Designing a Report

Set the Report Writer environment

- From the Genero Studio main menu, select **Tools>>Configurations**.
- In the **Compile/Runtime** section of the Genero Configuration Management window, the default environment sets are listed.
- The **Report Writer** environment set contains the variables that must be set for Genero Report Writer. This environment set should be active (has a check mark in the box).

Creating a Report

1. Create a Project

- In the Studio Project Manager, select the New Project icon to create a new project node for the report.

2. Create an Application

- Add an application node.

Select the project node that you created, and use the New Application icon to add a new application node to the project. This node will contain the files for the report.

- Add external dependencies.

Select the application node. In the Properties View, the following library, **libgre.42x**, should be entered for the external dependencies property. This library contains some mandatory functions that you must use in creating reports, as well as functions you can use to change the default behavior of Report Writer. See Reporting API for additional information about these functions.

If you have other .42x libraries that you wish to add, press the ... button for the external dependencies property in the Properties View. When the Edit List dialog opens, you can add additional filenames.

3. Create a BDL File

In Genero Report Writer, retrieving the report data is separated from the design of the report. Genero BDL is used to write the code for a **.4gl** file that retrieves the data from a database and outputs it to a report:

- Use the **File>>New** main menu option of Studio Project Manager to create a **Genero BDL file** for the report. See The Genero BDL Report Program for details about the report driver and the Report program block.
- Use the **File>>Save** main menu option to save this file (*<filename>.4gl*) and add it to the report application node.

4.

Create the Data Schema

- The **.rdd** file (the data schema for the report data) must be extracted from the BDL file that you created. This **.rdd** file will be used later, to populate the Data View of the Report Design Document.

You can execute the following from the command line:

```
fglcomp --build-rdd <sourcefile>.4gl
```

Or, you can add **--build-rdd** to the Compiler options property for the BDL file in Studio Project Manager, to generate the .rdd file automatically each time the .4gl is compiled.

See Fetching Report Data for additional options that can provide data for the report.

5. Create a Report Design Document

- Use the **File>>New, Reports** menu option of Genero Studio to create a new Report Design Document for the report, which will use the data from the .rdd file.
- Report output can be displayed in various formats. In the Report Design Window, you can use the **File>>Report properties** menu options to change the default format for the report design document.
- Save the report design document (*<filename>.4rp*), and add it to the report application node in Studio Project Manager.
- Use the **File>>Save All** menu option of Studio Project Manager to save the project with all its changes.

Tip: Different Report Design Documents can be created using the data from the same .4gl file.

6. Build the Report Application in Project Manager

Right-click the report application node or the report project node in the Studio Projects tab and select **Build** to compile and link the application.

Run the Report

- Right-click the report application node in the Project Manager and select **Execute** to run the program.

Note: If the report data will come directly from a database, your BDL program must have access at execution to the database where the data is stored.

- By default a new report is in SVG format, and the output is set to Preview using the Genero Report Viewer; you can change these defaults in your Report Design document, or in your BDL program code.

The Report Viewer: The Report Viewer is a component of the Genero Desktop Client that is automatically installed by the installation program of the GDC; it can be used to display reports that are in the SVG format only.

Important: SVG format for reports is currently not supported for previewing when running with the Web Client.

The **File>>Print** menu option allows you to send the displayed report to a printer.

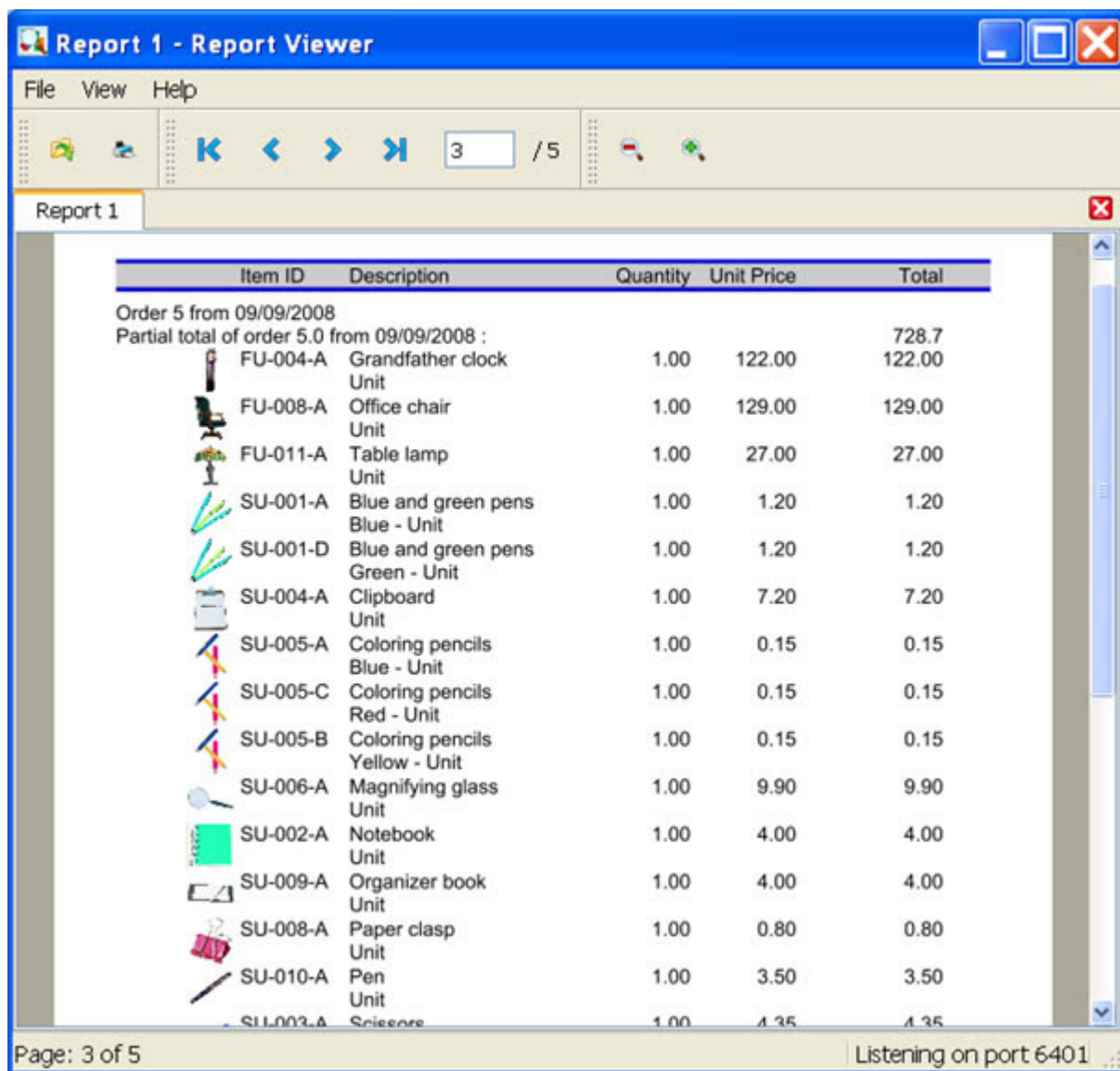


Figure 9-4. Genero Report Viewer

Configuring Fonts and Printers

Topics:

- Available Fonts and Printers
- Tips on installing common Fonts or new Fonts
- Specific Font Types

Available Fonts and Printers

Some scripts are provided in the `<GRE-install-dir>/bin` directory to get information about the available fonts and printers on that machine. These scripts are executed from the command-line:

- `fontinfo` - lists the fonts available on your server
- `fontinfo.pdf` - lists only the fonts that can be used in pdfs
- `printerinfo` - lists the available printers

For additional information about the use of **printerinfo**, see Support for Custom Paper Sources.

Tips on Installing common Fonts or new Fonts

The best results are achieved if the fonts on the server where Genero Report Engine (GRE) is installed, and those used in the Genero Report Designer, are the same:

- If GRE and the Designer are installed on the same machine, there is no problem.
- If GRE and the Designer are installed on different machines that have the same operating system and the same installed fonts, there is no problem.

On Windows systems you can use the scripts that are provided to get information about the available fonts on your machine.

A report that uses only a few positioned items will still lay out correctly, however, even if the fonts used in the Designer and the fonts used in the GRE server differ:

- If the server cannot find a specified font, it does not raise an error; it uses a fallback font instead. A warning has been added to the runtime system in debugging mode (`$GREDEBUG > 0`) when a fallback font is used because a font specified in a template file cannot be found on the system.
- If no font is specified, a default font is used.

Specify a font at the root of the document, to avoid potentially changing output when a new version of Java, or a different server, is used. In the Report Designer, set the font property for the Page Root node.

Not all fonts contain all possible characters: some fonts will not contain certain glyphs. In this case, GRE will attempt to take the missing characters from a different font that contains the glyphs. For example, the monospaced (fixed-width) font "Courier" does not contain graphics characters. If a report contains a grid that is drawn using graphics characters, GRE might substitute characters from a set that is not fixed-width, causing the layout to break. Avoid this problem by using a font like "Lucida Sans Typewriter", which is both fixed-width and contains the required characters.

Not all fonts can be used for embedding in PDF: the license flag contained in the font might prevent a font from being used. The utility `$<GREDIR>/bin/fontinfo.pdf` lists all fonts that can be used in PDF documents. While True Type fonts generally work, sometimes a Type 1 font will not appear in the list because it is available only as text (.pfa) but not as a binary file (.pfb). In this case, the binary font can be created by using font compilation tools such as `pfa2pfb` or `t1binary` from the **t1binaries** package.

Specific Font Types

Type 1 and Windows TrueType Fonts

Genero Report Engine is capable of reading both Type 1 and Windows TrueType fonts, so that it is possible to simply copy fonts from Windows to Unix. Since the TrueType directories differ from one Linux distribution to another, and between different Unix versions, it is possible to copy the fonts into the Java fonts directory `$JAVA_HOME/jre/lib/fonts` where they will be found by GRE. In order to check whether GRE sees the font you installed, run the executable `$<GREDIR>/bin/fontinfo` which lists all fonts.

Lucida family of Fonts

SUN-Java contains a basic set of fonts (Serif, Sans Serif and Monospaced) in the "Lucida" family. Some distributions of the Java Runtime Environment do not contain all fonts, but it is legal to copy the fonts from one distribution to another as stated in <http://java.sun.com/j2se/1.5.0/jre/README>. These fonts contain a large part of the unicode characters.

The available codes are listed in <http://java.sun.com/j2se/1.5.0/docs/guide/intl/font.html#lucida>.

Liberation fonts

Another option for free fonts is the "Liberation" fonts originally provided by Red-Hat. These fonts have the same metrics (character width) as the Microsoft fonts Arial, Times New Roman, and Courier, and a similar look. These fonts can be installed on both Windows and Linux. A description of the fonts can be found at http://en.wikipedia.org/wiki/Liberation_fonts. The fonts can be downloaded at <https://www.redhat.com/promo/fonts>.

Asian Fonts

When using Asian fonts in PDF or SVG documents:

- Set the fidelity property to "true" on any WORDBOX or WORDWRAPBOX using Asian characters.
- For reports running in compatibility, set the parameter fidelity to "true" in calls to the API function `fgl_report_configureCompatibilityOutput()`.

Make sure the specified fonts contain the required characters. The designer will display the characters correctly even though the selected font may not contain them; the runtime system does not have this behavior.

BDL Programs

Topics:

- "Writing the Genero BDL Report Program"
- "Configuring for Labels" on page 9-26
- "Report Data Wizard" on page 9-27

Writing the Genero BDL Report Program

Topics:

- Overview
- Tips
- Creating a Simple Report
 - Main Program block
 - The Report Driver
 - The Report Program block
- Fetching Report data
 - From a Database using SQL
 - From a Data File
 - From an XML File

- From a Web Service
- Output Options
 - Default Output
 - Output Options in the .4rp (Report Design Document)
- Creating the Data Schema file (.rdd)
- Allowing User to select Output options
- Running a Genero ASCII report using GRW (Compatibility Report)
 - Auto-formatting Reports that have no .4rp (Report Design Document)
 - Report Metadata in Compatibility Reports
 - Mixing Genero ASCII Reports, GRW Reports, and Compatibility Reports
- Sub Reports

See also: Change Output Options, Output Labels, Steps to a Report

Overview

Genero BDL is used to write the code for a **.4gl** file that retrieves the data from a database and outputs it to a report. This BDL program contains:

- The Report Driver, which specifies the data the report will include. If you are retrieving data from database tables, for example, a database cursor is defined to retrieve specific rows. The Report Driver stores the database values in program variables, and sends these - one record at a time - to the REPORT program block.
- The REPORT program block, which specifies the control blocks for a report, and sends the data to the Genero Report Engine. Variables can be defined to allow calculations on the data values, which are output along with the data values.

The Genero Report Engine uses the data and the Report Design document (.4rp - created using Genero Report Designer) to process the report and output it, in accordance with the Reporting API functions that have been called by the BDL program.

See Steps to a Report for a complete outline of the reporting process.

Some Tips for Legacy Report Designers provides information about the correlation between Report Designer and traditional 4GL commands in reports.

The **OrderReport.4gl** example program included in the GRW demo **Reports**:

- Retrieves the data and outputs it to the Report Engine.
- Allows the user to select the desired report and the output settings, by displaying a form (**Configuration.42f**) for input.
- Uses the Reporting API function `fgl_report_loadCurrentSettings()` to load the original settings from the .4rp file. (Additional optional functions can specify the changes to be made to those settings, based on the user input.)
- Uses the Reporting API function `fgl_report_commitCurrentSettings()` to commit the changes this program has made.
- The Form Design file for this form, (**Configuration.4fd**) is one of the files in the GRW demo project, **Reports**.

See the Reporting API for the specific functions that you can use to control the output of a report; some examples are in Change Report Output Options.

Tips

1. The Reporting API functions `fgl_report_loadCurrentSettings` and `fgl_report_CommitCurrentSettings` are required and must be called in sequence. (Calls to these functions bracket any calls to other functions that change the default output options.) These functions must be called in the program prior to executing the `START REPORT` command in the Report Driver. If your program does not change the default output options, you can simply call the single function `fgl_report_loadAndCommit` instead.
2. Always use `ORDER EXTERNAL` in the `REPORT` program block to tell the report that the input records have already been sorted by the `SELECT` statement, and there is no need for the program to resort the fields. This will improve performance.
3. Restrict your use of control blocks in the `REPORT` program block to `FIRST PAGE HEADER`, `BEFORE GROUP OF`, `AFTER GROUP OF`, `ON EVERY ROW`, or `ON LAST ROW`. See The Report Program Block.
4. Prefer the use of fields over expressions to hold report values. Since expressions don't have names, it will be difficult to reference them in the Report Design Document (they could only be referenced by index).
5. Prefer calculating values in the `REPORT` program block over calculating in the Report Design Document, to use the power of the DVM.
6. Don't be stingy when outputting fields; this will enhance the ability to use the BDL source for multiple report formats. Define a record, **orderline** for example, that contains all the data fields retrieved from the database by the database cursor; use that record definition in the `OUTPUT TO REPORT` statement:

```
OUTPUT TO REPORT <reportname> (orderline.*)
```

Creating a Simple Report

This example BDL program (`SimpleReport.4gl`) uses data from the **officestore** database to create a report. There are more complex examples in the Reports project provided as a demo with Genero Report Writer.

Type Definition

This section gives the database schema used in variable definitions, and defines a User Type that consists of a single record containing all the fields from all the referenced tables.

Creating a `TYPE` allows the record definition to be specified only once in the program; thereafter, the name of the `TYPE` is used in the program wherever that record definition would be required.

```
01 SCHEMA officestore
02
03 TYPE ORDERTYPE RECORD
04     orders    RECORD LIKE orders.*,
05     account   RECORD LIKE account.*,
06     country   RECORD LIKE country.*,
07     lineitem  RECORD LIKE lineitem.*,
08     product   RECORD LIKE product.*,
09     category  RECORD LIKE category.*,
10     item      RECORD LIKE item.*
11 END RECORD
12
```

MAIN program block

The **MAIN** program block contains the program logic that allows a user to run a report.

- Line 15 - defines a report handler object.
- Lines 18 and 19 - call the mandatory BDL functions that configure the report:
 - `fgl_report_loadCurrentSettings` accepts the name of the Report Design document (.4rp) as a parameter. The return value is a boolean indicating whether the load of the settings from the .4rp file was successful.
 - `fgl_report_commitCurrentSettings` returns a `SaxDocumentHandler` object, defined as **handler**.
- Line 26 - calls the Report Driver function, **runReportFromDatabase**, to run the report, passing **handler**.

Note: The example **runReportFromDatabase** function contains the **START REPORT** statement. Do not place any code between the call to **fgl_report_commitCurrentSettings** and the **START REPORT** statement that would allow the user to cancel the report

```
13 MAIN
14
15   DEFINE handler om.SaxDocumentHandler-- report handler
16
17   --call the mandatory functions that configure the report
18   IF fgl_report_loadCurrentSettings("myreport.4rp") THEN -- if the file
                                                    -- loaded OK
19       LET handler = fgl_report_commitCurrentSettings() -- commit the file
                                                    -- settings
20   ELSE
21       EXIT PROGRAM
22   END IF
23
24   -- run the report by calling the report driver contained
25   -- in your function runReportFromDatabase
26   IF handler IS NOT NULL THEN
27       CALL runReportFromDatabase(handler)
28   END IF
29 END MAIN
```

Important: The `helpers.42m` file contains these mandatory functions, and other BDL helper functions for Reports. This compiled file is part of the **libgre.42x** library, which must be included in any report application; list this library in the **external dependencies property** of any Genero Studio Project Manager application node that uses Genero Report Writer.

The Report Driver

The **runReportFromDatabase** function uses SQL to extract the data from the database officestore:

- Line 31 - **handler** previously created by the `fgl_report_commitCurrentSettings` function is passed to this function
- Line 32 - defines a record variable **orderline** using the User Type defined in **MAIN**
- Line 33 - defines **handler** passed as a parameter to this function
- Line 35 - makes a connection to the database in unique-session mode, since the program will not need to connect to other databases.

- Lines 37 through 52 - declare a cursor for the SQL statement.
- Line 52 - uses the **ORDER BY** clause of the SQL statement defines the sort order of the data.
- Lines 54 thru 59 are only used when the data is being retrieved from a database.
 - Line 54 - the BDL statement **START REPORT** *<reportname>* **TO XML HANDLER** must be used to instantiate the report driver, using the **handler** that was passed to this function; this specifies that the report data should be output in XML format.
 - Lines 55 through 57 - the **FOREACH** loop opens the cursor and fetches the data from the database into the record variable, one row at a time.
 - Line 56 - the **OUTPUT TO REPORT** statement outputs each data row to the REPORT program block.
 - Line 58 - the **FINISH REPORT** terminates the BDL report process.
 - Line 59 - closes the SQL cursor.

```

30
31 FUNCTION runReportFromDatabase(handler)
32   DEFINE orderline ORDERTYPE,           -- User Type defines record
33       handler om.SaxDocumentHandler -- definition for parameter
                                           -- passed to this function
34
35   DATABASE "officestore"    -- database connection
36   DECLARE c_order CURSOR FOR -- cursor declaration
37   SELECT orders.*,
38       account.*,
39       country.*,
40       lineitem.*,
41       product.*,
42       category.*,
43       item.*
44   FROM orders, account, lineitem, product, category, item, country
45   WHERE
46     orders.orderid = lineitem.orderid
47     AND orders.userid = account.userid
48     AND lineitem.itemid = item.itemid
49     AND item.productid = product.productid
50     AND product.catid = category.catid
51     AND country.code = orders.billcountry
52   ORDER BY orders.userid, orders.orderid, lineitem.linenum
53
54   START REPORT report_all_orders TO XML HANDLER handler -- handler that was
                                                           -- passed to this function
55   FOREACH c_order INTO orderline.*                      -- use cursor to fetch data
56     OUTPUT TO REPORT report_all_orders(orderline.*) -- send data to report
                                                         -- function
57   END FOREACH
58   FINISH REPORT report_all_orders
59   CLOSE c_order
60
61 END FUNCTION

```

See the Genero BDL manual for additional information about the use of cursors, connections, and BDL report statements.

The REPORT program block

This program block accepts the data from the driver, specifies the order in which the data was sorted, and outputs the data to be formatted as specified in the report design page (.4rp).

The **FORMAT** section specifies the control blocks for a report. The use of each control break is optional, depending on the requirements of your report document. We recommend that you restrict your usage to the following control blocks:

- **FIRST PAGE HEADER** - specifies the action that the runtime system takes before it begins processing the first input record.
- **BEFORE GROUP OF/AFTER GROUP OF** - specifies the action the runtime system takes before or after it processes a group of input records.
- **ON EVERY ROW** - specifies the action the runtime system takes for every input record that is passed to the report definition.
- **ON LAST ROW** - specifies the action the runtime system takes after it processes the last input record that was passed to the report definition and encounters the **FINISH REPORT** statement.

See BDL Reports in the Genero BDL manual for a complete discussion of the BDL statements associated with a **REPORT** block.

Note: Since **PAGE HEADER** and **PAGE TRAILER** are triggered based on the line count of the BDL report, which does not correspond with the actual page breaks, their usage should be avoided. Create page headers and footers in the report design document instead.

Example **REPORT** block from SimpleReport.4gl:

- Lines **63** through **67** - define variables to allow calculations on the data values; the calculations are output along with the data.
- Line **70** and **71** - the **ORDER EXTERNAL BY** statement informs the report that the data was retrieved, and will be output, in the specified sort order
- Lines **72** through **85** - the **FORMAT** section of the report uses control blocks to set the values of the report variables used to store calculations and to send the report data and calculations to the report. Unlike a Genero report, the **PRINT** statement does not contain any formatting of the data or the report line (**SPACES**, **LINE_NO**, and **COLUMN** operators aren't used, for example.) The report format is specified in the Report Design document (.4rp).
- Lines **74** and **75** - set the variable **overalltotal** to zero at the control break at the beginning of the report (**FIRST PAGE HEADER**).
- Lines **77** and **78** - re-set the variable **ordertotal** to zero each time the value of orderid changes in the data received from the report driver (**BEFORE GROUP OF**).
- Lines **81** through **85** - For each row of data received from the report driver (**ON EVERY ROW**), these lines store some calculations in variables. The **PRINT** statement outputs the database data and variables to the report, in XML format.

```

62 REPORT report_all_orders( orderline )
63   DEFINE
64     orderline      ORDERTYPE,
65     lineitemprice  LIKE lineitem.unitprice,    -- total price for item
66     overalltotal   LIKE orders.totalprice,    -- accumulator for total price
                                                -- for report
67     ordertotal     LIKE orders.totalprice      -- accumulator for total price
                                                -- for order
68
69   -- specify the order of the sorted data resulting from SQL statement
70   ORDER EXTERNAL BY orderline.orders.userid, orderline.orders.orderid,
71                     orderline.lineitem.linenum
72
73   FORMAT
74     FIRST PAGE HEADER
75     LET overalltotal=0                      -- initialize report total

```



```

76
77     BEFORE GROUP OF orderline.orders.orderid
78         LET ordertotal=0                                -- initialize ordertotal for
                                                            -- each new order
79
80     -- after calculations for each data row, output the data and
81     -- the calculations to the Report Engine
82     ON EVERY ROW
83         LET lineitemprice = orderline.lineitem.unitprice *
84         orderline.lineitem.quantity
85         LET overalltotal=overalltotal + lineitemprice
86         LET ordertotal=ordertotal + lineitemprice
87         PRINT orderline.*, lineitemprice, overalltotal, ordertotal
88
89 END REPORT

```

Fetching Report Data

You can obtain the data for the report in several ways.

From a Database using SQL: You can use SQL statements to retrieve the data for the report from database tables, requiring a connection to the database and the use of an SQL cursor. The SimpleReport.4gl program uses this technique, shown in the runReportFromDatabase function.

From a Data File: You can get the report data from a data file - one created by the BDL UNLOAD statement, for example. This example uses the **OrderReport.unl** file in the GRW demo Reports.

The only change to the MAIN program block would be the call to the runReportFromFile function in line 26.

```

13 MAIN
14
15     DEFINE handler om.SaxDocumentHandler -- report handler
16
17     --call the mandatory functions that configure the report
18     IF fgl_report_loadCurrentSettings("myreport.4rp") THEN -- if the file
                                                            -- loaded OK
19         LET handler = fgl_report_commitCurrentSettings() -- commit the file
                                                            -- settings
20     ELSE
21         EXIT PROGRAM
22     END IF
23
24     -- run the report by calling the report driver contained in your
25     -- function runReportFromFile
26     IF handler IS NOT NULL THEN
27         CALL runReportFromFile(handler)
28     END IF
29 END MAIN

```

The function **runReportFromFile** replaces the **runReportFromDatabase** function as the Report Driver. It uses the unload file **OrderReport.unl** to provide the data for the report.

- Line 35 Defines **ch** as a variable of type **base.channel**. The BDL Channel class provides read/write access to files.
- Line 36 defines the variable **dataFile** to specify the unload file containing the data.
- Line 38 assigns the report name to the **dataFile** variable.
- Line 39 creates the **ch** channel object.

- Line 40 opens the channel to the data file **OrderReport.unl** in read (r) mode.
- Line 43-45 The WHILE statement reads a line from the data file, providing the variable list enclosed in brackets. The line is output to the REPORT program block. The statement terminates when all the lines have been read.
- Line 48 closes the channel.

```

30
31 FUNCTION runReportFromFile(handler)
32   DEFINE
33     orderline OrderType,
34     handler om.SaxDocumentHandler,
35     ch base.channel,    -- definition of channel object
36     dataFile String     -- file containing report data
37
38   LET dataFile = "./OrderReport.unl"
39   LET ch = base.Channel.create()
40   CALL ch.openFile(dataFile,"r")
41
42   START REPORT report_all_orders TO XML HANDLER handler
43   WHILE ch.read([orderline.*])
44     OUTPUT TO REPORT report_all_orders(orderline.*)
45   END WHILE
46   FINISH REPORT report_all_orders
47
48   CALL ch.close()
49 END FUNCTION

```

From an XML File: You can get the report data from an XML file instead of getting it directly from a database. The XML file can be a ProcessLevelData file created by a Genero report (Output to an XML File) which has a special format, or it may be an arbitrary XML file.

From A ProcessLevelData file

This XML file has a special format, and is created using the Report Writer API function fgl_report_createProcessLevelData File().

The MAIN block of SimpleReport.4gl would change as follows:

- Line 26 - provides the name of the data file to be used for the report
- Lines 27 and 28 - use the BDL function fgl_report_runReportFromProcessLevelDataFile to run a report using a data file. The parameters are:
 - The report handler object
 - A String containing the name of the XML file that contains the data
- This function will replay the report from the file thereby replacing the running of the report (START REPORT, OUTPUT TO REPORT, FINISH REPORT statements).

```

13 MAIN
14
15   DEFINE handler om.SaxDocumentHandler, -- report handler
16           data_file String,             -- XML file containing data
17           report_ok boolean             -- return value from function 18
18
19   --call the mandatory functions that configure the report
20   IF fgl_report_loadCurrentSettings("myreport.4rp") THEN -- if the file
                                                         -- loaded OK
21     LET handler = fgl_report_commitCurrentSettings() -- commit the file
                                                         -- settings
22   END IF
23
24   --call the report driver to run the report

```

```

25 IF handler IS NOT NULL THEN
26   LET dataFile = "./OrderReportData.xml"
27   LET report_ok =
28     fgl_report_RunReportFromProcessLevelDataFile(handler, dataFile)
29 ELSE
30   EXIT PROGRAM
31 END IF
32
33 END MAIN

```

From Other XML files

XML files that were not created by the Reporting API function `fgl_report_createProcessLevelData File()` can be used as the data source for a Genero report. The file **OrderData.xml** in the demo sample files of the Reports project is an example of this type of file.

The function `fgl_report_runFromXML` is used to specify the data source:

- This function sets up the report engine based on the current settings that have previously been loaded by a call to `fgl_report_loadCurrentSettings()`, and may have been modified by calls to `fgl_report_selectDevice()` or `fgl_report_selectPreview()`. The function automatically calls `fgl_report_commitCurrentSettings()`.
- This function will replay the report from the file thereby replacing the running of the report (START REPORT, OUTPUT TO REPORT, FINISH REPORT statements).

The following example is from the **OrderReportXML.4gl** demo program in the Reports project:

```

1 MAIN
2 IF NOT fgl_report_loadCurrentSettings("Table.4rp") THEN
3   EXIT PROGRAM
4 END IF
5 IF NOT fgl_report_runFromXML("OrderData.xml") THEN
6   DISPLAY "RUN FAILED"
7   EXIT PROGRAM
8 END IF
9 END MAIN

```

From a Web Service: Web Services can be used to fetch data for a report. BDL functions allow you to access a web service and retrieve data, storing it in BDL program variables. See **Using Web Services** in the Studio manual, and the Genero Web Services manual for additional information and examples.

Output Options

The output format of your report can be created as an SVG report, a PDF file, or an image. The output will be sent to the destination specified by the output options. Only SVG format reports can be previewed and printed from the Genero Report Viewer.

Important: SVG format for reports is currently not supported for previewing when running with the Web Client.

Default Output

If you don't override the default options, on report execution the user will:

- Create an SVG report

- Preview the Report in Genero Report Viewer, which is a part of the **Genero Desktop Client**.

The **File>>Print** menu option of the Report Viewer allows the user to direct the output to a printer.

The Reporting API provides BDL functions to change the output in your BDL program by overriding the default options. Some examples are provided in Changing Output Options. Also see Allow User to Select Output.

Output Options in the .4rp (Report Design Document)

The **configuration menu (File>>Report properties)** of the Report Design work area allows you to override the following output options for a specific Report Design Document (.4rp):

- Paper Settings
- Output Format Settings

See also: Configuring Fonts and Printers

Creating the Data Schema file (.rdd)

Use the command-line option of the tool **fglcomp** to create a Report Writer data schema (.rdd file) based on the SQL statement in your BDL (.4gl) file. This **.rdd** file is used in the Report Designer to populate the Data View. The schema will contain the list of database columns that make up your data record (**orderline** in the example program). For example:

```
fglcomp --build-rdd SimpleReport.4gl
```

The output of this command will be **SimpleReport.rdd**. The .rdd file will be stored in the same location as the .4gl file.

Note: If your BDL program contains multiple .4gl files, run **fglcomp** against the file containing your REPORT program block.

Generating the .rdd file automatically

As an alternative, you can add **--build-rdd** to the **Compiler options property** for your BDL (.4gl) file to generate the .rdd file automatically each time the .4gl file is compiled. Select the .4gl file listing in the application node of the Project tab to display its properties in the Properties View. The .rdd file will be stored in the directory specified in the **Target Directory property** of the application node that contains the .4gl file.

Allowing the User to Select Output Options

In the GRW demo program **Order Report**, the following form allows the user to make choices regarding the output:

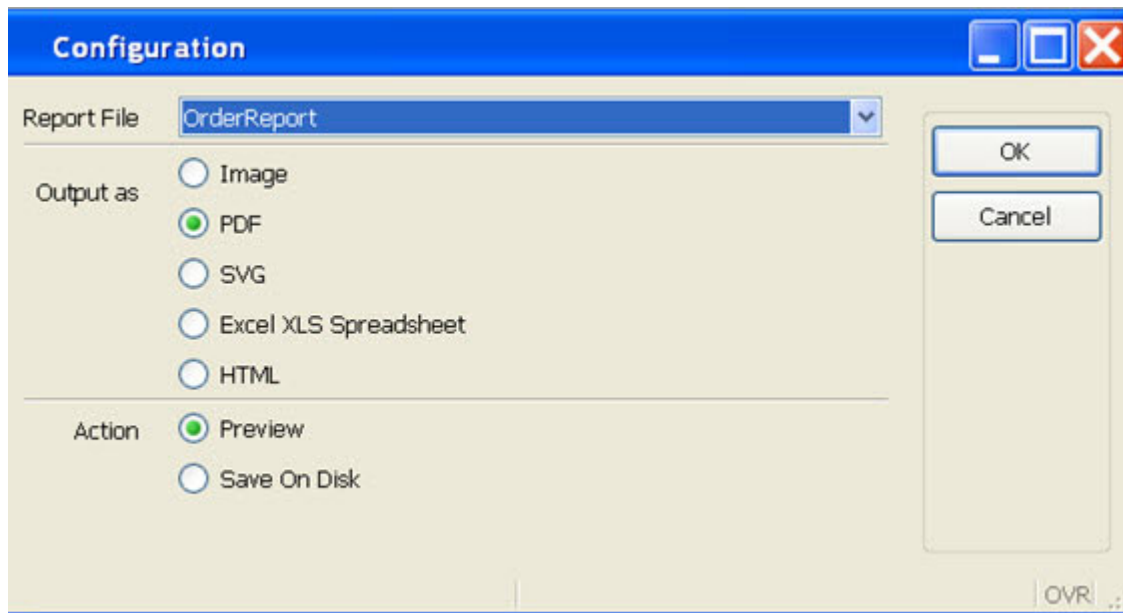


Figure 9-5. Report demo Form

The following code implements this functionality. The form definition file, **Configuration.4fd**, is included in the demo files for **Reports**, the GRW Demo.

The MAIN block of SimpleReport.4gl would change as follows:

- Lines 23 through 31 - call a function named `select_output` that displays the form and returns the user's choices
- Lines 25 through 27 - configure the report using the mandatory Reporting API functions plus the following additional functions:
 - `fgl_report_selectDevice` - sets the report output device; the parameter is one of: "Printer", "PDF", "Image", "SVG" .
 - `fgl_report_selectPreview` - sets the preview option; the parameter is TRUE (preview) or FALSE (no preview).

```

13 MAIN
14
15  DEFINE handler om.SaxDocumentHandler, -- report handler
16      data_file String,                -- XML file containing data
17      report_ok boolean,               -- return value from function
18      r_filename STRING,               -- filename of desired report
19      r_output STRING,                 -- output format
20      preview INTEGER                  -- preview indicator
21
22  -- call the mandatory functions that configure the report using the
23  -- choices made by the user
24  -- in your new select_output function
25  CALL select_output() RETURNING r_filename, r_output, preview
26  IF fgl_report_loadCurrentSettings(r_filename) THEN -- if the file loaded OK
27      CALL fgl_report_selectDevice(r_output)        -- changing default
28      CALL fgl_report_selectPreview(preview)        -- changing default
29      LET handler = fgl_report_commitCurrentSettings() -- commit changes
30  ELSE
31      EXIT PROGRAM
32  END IF
33
34  -- run the report by calling the report driver contained in
35  -- your function runReportFromDatabase
36  IF handler IS NOT NULL THEN

```

```

23     CALL runReportFromDatabase()
23 END IF
38
39 END MAIN

```

select_output - this new function displays a form allowing the user to specify the name of the report to be run (the name of the .4rp file), the output format, and whether a preview is preferred.

```

01 FUNCTION select_output()
02     DEFINE
03         r_output    STRING,    -- output format option
04         r_action    STRING,    -- used to set preview option
05         r_filename  STRING,    -- filename of Report Design document (.4rp filename)
06         preview     INTEGER    -- preview option TRUE/FALSE, to display
                                -- report in Report Viewer
07
08     CLOSE WINDOW SCREEN
09     OPTIONS INPUT WRAP
10     -- display form allowing user to select the .4rp filename and output options
11     OPEN WINDOW f_configuration WITH FORM "Configuration"
12     LET INT_FLAG=FALSE
13     LET preview = FALSE
14     INPUT BY NAME r_filename,r_output,r_action
15     ON ACTION CANCEL
16         EXIT PROGRAM
17     ON ACTION CLOSE
18         EXIT PROGRAM
19     END INPUT
20     CLOSE WINDOW f_configuration
21     CALL ui.interface.refresh()
22
23     -- set preview variable to match output option r_action
24     IF r_action IS NOT NULL THEN
25         IF r_action == "preview" THEN
26             LET preview = TRUE
27         END IF
28     END IF
29
30     -- return report name (filename of Report Design Document) and output
31     -- option variables to be used to configure the report engine
32     RETURN r_filename||".4rp", r_output, preview
33 END FUNCTION

```

Running a Genero ASCII Report Using GRW (Compatibility Report)

If you have an existing Genero BDL program (.4gl) that creates an ASCII report, you can execute the program using the Genero Report Writer without changing the existing Report Driver or the existing Report program block, and without creating a Report Design document (.4rp.) This type of report is referred to as a **Compatibility report**.

The following additional lines would be made in the MAIN program block of the .4gl file, to call the mandatory API functions that configure the report:

```

DEFINE handler om.SaxDocumentHandler

IF fgl_report_loadCurrentSettings(NULL) THEN    -- switch on Compatibility mode
                                                -- (run without .4rp file)
    LET handler = fgl_report_commitCurrentSettings() -- commit settings
END IF

```

Switching on Compatibility mode overrides the "TO ..." part of the START REPORT BDL statement.

The formatting of the report will be that specified in the REPORT program block of your BDL program. (See Auto-formatting for an alternative to the Compatibility format.) The report engine will use the default output settings, displaying a preview of the report in SVG format in the Report Viewer. Additional SVG preview options are available, using the API function `fgl_report_configureSVGPreview`. The API function `fgl_report_configureCompatibilityOutput` can optionally be used to change the default output settings.

The demo report **OrderReportASCII** illustrates this feature.

Auto-formatting Reports that have no .4rp (Report Design Document)

Beginning with version 2.40, auto-formatting of these reports that have no report design document (.4rp) is available, as an alternative to the Compatibility format. New API calls have been added to provide generic report formatting. Currently only one design is provided, a simple list design that is compatible with the List Report template that is shipped with Genero Report Writer. This type of auto-formatting is particularly well suited to produce Excel output from arbitrary reports. See `fgl_report_setAutoformatType` and `fgl_report_configureAutoformatOutput`.

Report Metadata in Compatibility Reports

New API functions allow you to add report metadata to Compatibility Reports. These functions are intended to be used for compatibility reports only. See Report Metadata functions. For regular reports, the metadata can be set in corresponding properties in the Properties View of the Report Designer.

Mixing Genero ASCII Reports, GRW Reports, and Compatibility Reports

Genero ASCII (text-based) reports and GRW (graphical) reports can be run from the same Genero program without requiring any additional calls.

However, running Compatibility Reports requires that Compatibility mode be switched on. Once Compatibility mode is invoked, it stays active throughout the program, running any subsequent Genero ASCII reports using the Genero Report Writer also. If your program needs to restore text-based output for a Genero ASCII report, you must call the API function `fgl_report_stopGraphicalCompatibilityMode`.

Sub Reports

Sub reports allow one report to be called from another. This allows reusing both the 4GL code producing the report output as well as the available designs. In addition, the mechanism can be used to break complex designs into smaller, potentially reusable parts.

On the 4GL side, a sub report call is detected when a START REPORT instruction is found within another report. The report containing the START REPORT instruction is called the **master report**, and the other report is the **sub report**.

Creating a master report data source

This code shows a master report **report_all_orders_twice** that calls the sub report **report_all_orders** from the ON EVERY ROW section two times. The invocation of a sub report requires the START REPORT, OUTPUT TO REPORT and FINISH

REPORT to be within the master report as shown. The START REPORT instruction is not allowed to be placed inside a function called from the report.

The START REPORT instructions for subreports must not include the TO clause (START REPORT *repname* TO XML HANDLER); doing so will yield unexpected results.

```
REPORT report_all_orders_twice()
  DEFINE
    orderline OrderType,
    ch base.channel

  FORMAT
    ON EVERY ROW

    LET ch = base.Channel.create()
    CALL ch.openFile(dataFile,"r")
    DISPLAY "MASTER ITERATION 1 "
    START REPORT report_all_orders
    WHILE ch.read([orderline.*])
      OUTPUT TO REPORT report_all_orders(orderline.*)
      IF fgl_report_getErrorStatus() THEN
        DISPLAY "FGL: STOPPING REPORT, msg=\"\",fgl_report_getErrorString(),\"\""
      EXIT WHILE
    END IF
  END WHILE
  FINISH REPORT report_all_orders
  CALL ch.close()

  DISPLAY "MASTER ITERATION 2 "
  LET ch = base.Channel.create()
  CALL ch.openFile(dataFile,"r")

  START REPORT report_all_orders
  WHILE ch.read([orderline.*])
    OUTPUT TO REPORT report_all_orders(orderline.*)
    IF fgl_report_getErrorStatus() THEN
      DISPLAY "FGL: STOPPING REPORT, msg=\"\",fgl_report_getErrorString(),\"\""
    EXIT WHILE
  END IF

  END WHILE
  FINISH REPORT report_all_orders
  CALL ch.close()

END REPORT
```

Detected sub report calls are shown in the report designer. They appear as nodes in the data view, the structure view and as images in the document view.

Creating the master report design

Create a new empty 4rp file. Select the master report data source as data schema.

In the data view the invocation of the sub report is displayed at its position in the 4gl source. While the execution of the sub report requires 3 invocations (START REPORT, OUTPUT TO REPORT and FINISH REPORT) only the location of the START REPORT item is shown in the designer.

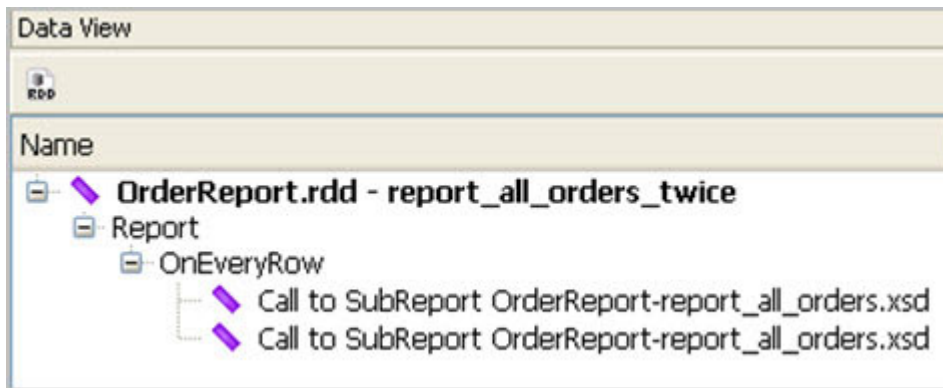


Figure 9-6. Sub reports in the Data View

A sub report invocation trigger is created in the document. This trigger has a URL property that allows selecting a 4rp design that matches the sub report schema.

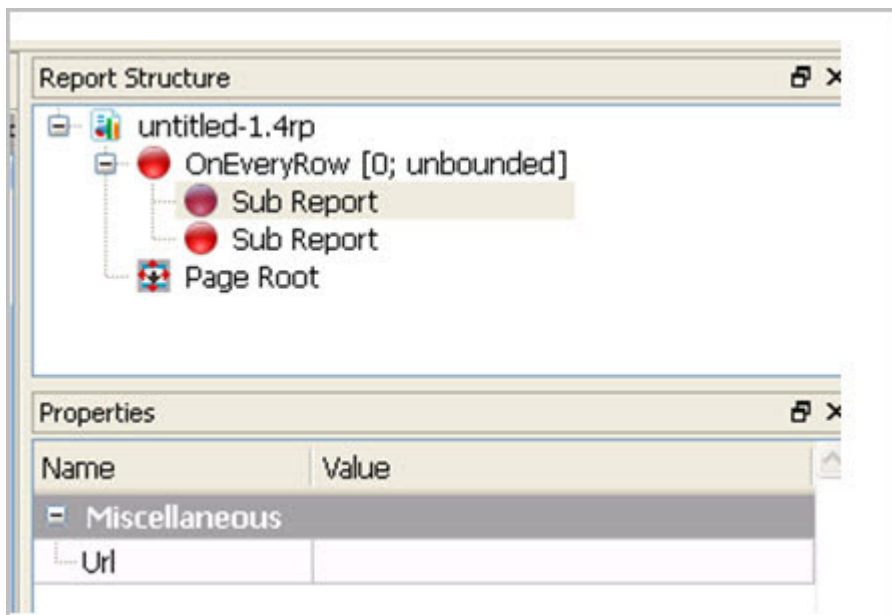


Figure 9-7. Sub reports invocation trigger in the Report Structure

Runtime behavior

At runtime, the two designs are combined. The sub report is inserted in the master report.

The behavior at the insertion point follows the same rules that generally apply to nested containers. For this reason the master report needs to be designed with awareness of the configuration of the root container of the sub report. For example, if the sub report occupies all the space it can get (MINIPAGE/max/max), the master report should be prepared to handle that. In particular, the use of ancestors other than MINIPAGE should be avoided, since they will become overfull if the sub report requires more than one page of space.

Configuring for Labels

Topics:

- Steps to Create Labels
 - In the Report Design Document
 - In your BDL report application
- Example

Steps to Create Labels In the Report Design Document

Create a design document for labels as follows:

1. Set the page size for the report to the size of one label.
2. Define one label inside the small page on your Design document; see **OrderLabels.4rp** in the Reports demo files.

In the BDL Application

Adjust your BDL application as follows:

- In the FORMAT section of the REPORT program block, output the data for one label ON EVERY ROW.
- In the MAIN program block, configure the Report Engine for labels using the following Reporting API functions:
 - fgl_report_selectLogicalPageMapping allows you to design one label in the Report Design Document that will be repeated as specified in fgl_report_configureLabelOutput.
 - Function fgl_report_setPaperMargins - the parameters are: top, bottom, left, and right margins.
 - Function fgl_report_configureLabelOutput - configures the physical layout of a label page. The parameters are:
 - paper Width
 - paper Height
 - label Width (set to null)
 - label Height (set to null)
 - number of labels per row
 - number of labels per column

Example

This example uses the OrderLabels.4rp example file from the Reports GRW demo.

In the SimpleReport BDL program, the MAIN program block that configures the report engine is changed as follows:

- Line 24 - loads OrderLabels.4rp as the Report Design Document
- Line 26 - the function fgl_report_selectLogicalPageMapping allows you to specify the format of the page as "labels"
- Line 27 - the function fgl_report_setPaperMargins sets margins using the following parameters:
 1. top margin
 2. bottom margin
 3. left margin
 4. right margin

- Line 28 - Function fgl_report_configureLabelOutput - configures the physical layout of a label page using the following parameters:
 1. paper Width
 2. paper Height
 3. label Width
 4. label Height
 5. number of labels per row
 6. number of labels per column
- ```

23 -- configure report engine; the functions prefixed fgl that are called here
 -- are part of the Reporting API
24 IF fgl_report_loadCurrentSettings("OrderLabels.4rp") THEN -- load the labels
 -- design document (.4rp file)
25 -- special settings for labels
26 CALL fgl_report_selectLogicalPageMapping("labels")
27 CALL fgl_report_setPaperMargins("5mm","5mm","4mm","4mm")
28 CALL fgl_report_configureLabelOutput("a4width","a4length",null,null,2,6)
29 END IF
30 LET handler = fgl_report_commitCurrentSettings() -- commit changes
31

```

The rest of the SimpleReport program would remain unchanged.

This example would print two labels across and six labels down per page.



Figure 9-8. Labels example

## Report Data Wizard

### Topics:

- “Using the Report Data Wizard” on page 9-28
  - “Table Selection Page” on page 9-28
  - “Define Joins Page” on page 9-28
  - “Result Set Page” on page 9-30
  - “Order By Page” on page 9-31
- Example Wizard Code
- Using the Example Code

See also: Writing the BDL Program, Steps to a Report

## Using the Report Data Wizard

A Wizard is now available to generate BDL code for your Genero report program (.4gl). Select **File>>New, Reports, Report Data** to display the dialogs that will guide you through the specification of the SQL query statement that will be used to extract the report data from your database.

### Table Selection Page

Select the database and tables for the query.

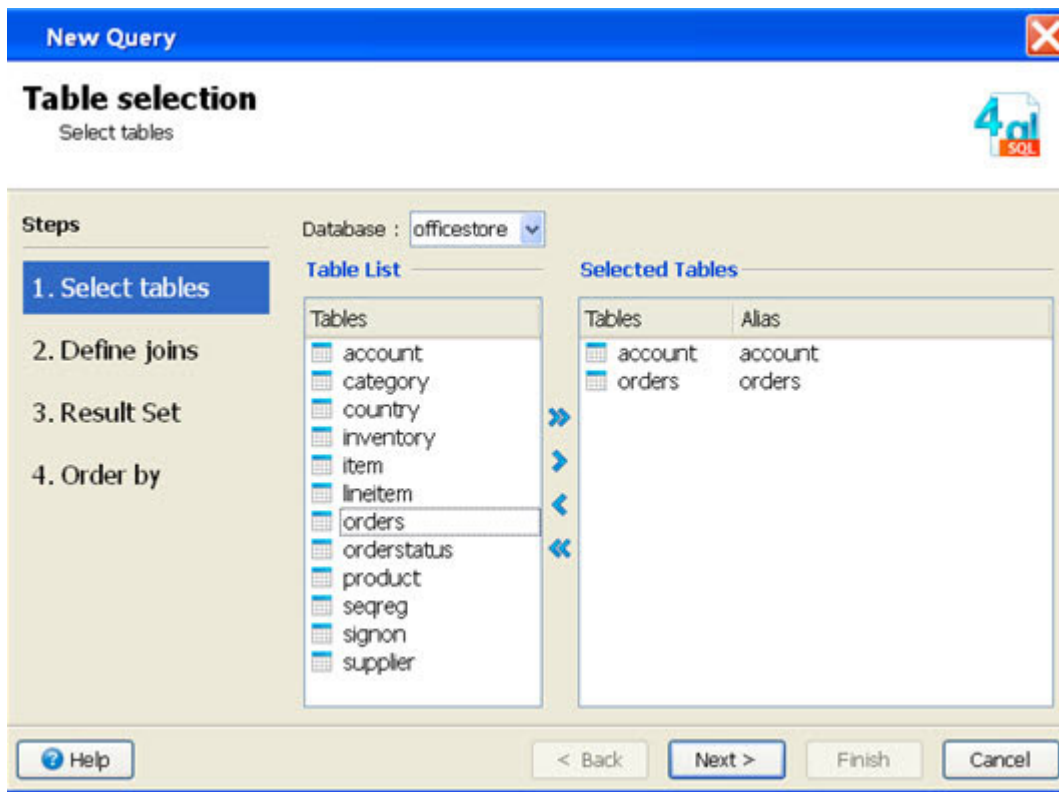


Figure 9-9. New Query wizard, Table Section page

Press **Next** to continue.

### Define Joins Page

This page allows you to define the joins between the tables to be used in the SQL query. The information is pre-filled from the database metadata in the Studio Schema file (.4db). You can select different columns in the tables, or select a different join method:

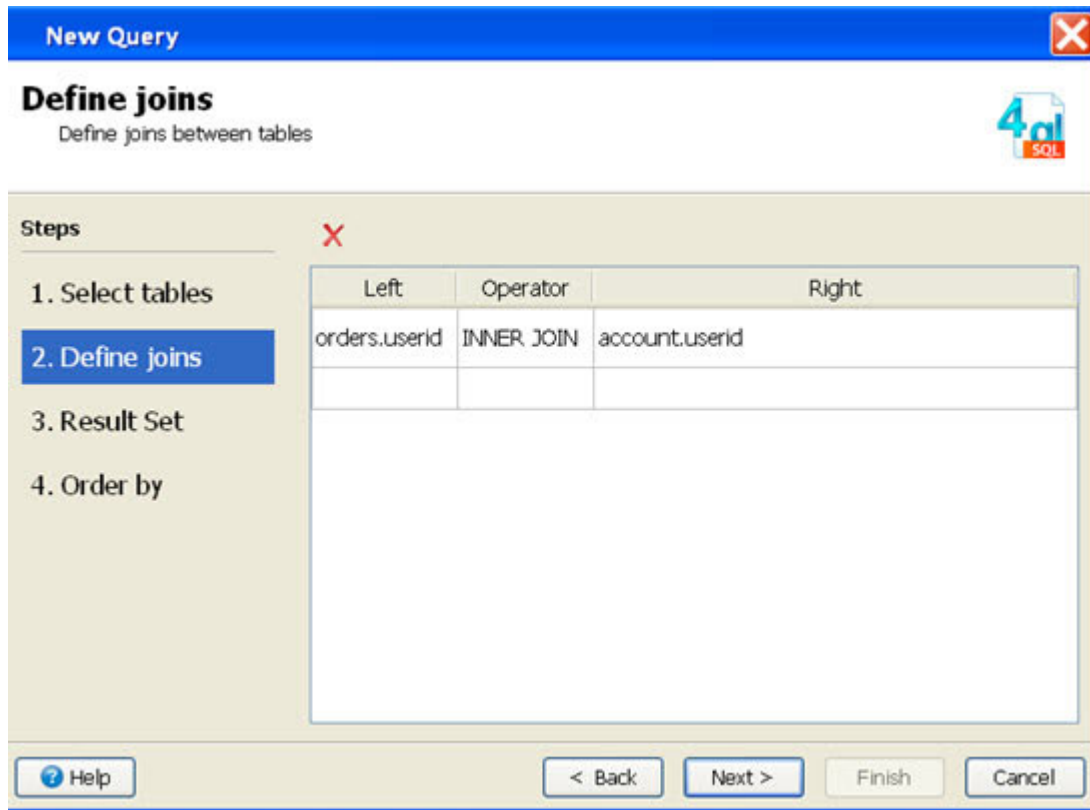


Figure 9-10. New Query wizard, Define joins page

- **Left column:** The table on the left side of the join operator.
- **Right column:** The table on the right side of the join operator.
- **Operator:** the method to be used to join the tables for the result set.

Press **Next** to Continue.

The following table illustrates the effect of the operators on the query result set:

Table 9-1. Effect of the join operators on the query result set

| Operator        | Description                                                                                                                                                                                                                                                                  | Example WHERE clause generated for BDL code                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| INNER JOIN      | Result set will contain columns from only the rows in the two tables where the specified column value matches.                                                                                                                                                               | WHERE orders.userid = account.userid                             |
| LEFT OUTER JOIN | Result set will contain columns from all the rows in the left table, joined with columns from the rows in the right table where the specified column value matches. Where there is no matching value, the columns from the right table in the result set will contain nulls. | WHERE (orders.userid = account.userid OR account.userid IS NULL) |

Table 9-1. Effect of the join operators on the query result set (continued)

| Operator         | Description                                                                                                                                                                                                                                                                    | Example WHERE clause generated for BDL code                                               |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| RIGHT OUTER JOIN | Result set will contain columns from all the rows in the right table, joined with columns from the rows in the left table in which the specified column value matches. Where there is no matching value, the columns from the left table in the result set will contain nulls. | WHERE (orders.userid = account.userid OR orders.userid IS NULL)                           |
| FULL OUTER JOIN  | Result set will contain columns from all the rows in the left table, and columns from all the rows in the right table, joined on the specified column value. Where there is no match in one of the tables, the columns from that table will contain nulls.                     | WHERE (orders.userid = account.userid OR account.userid IS NULL OR orders.userid IS NULL) |

## Result Set Page

Select the database table fields for the result set of the SQL query. The data in the result set is passed by the BDL REPORT function to the Genero Report Engine and the Report Definition file (.4rp).

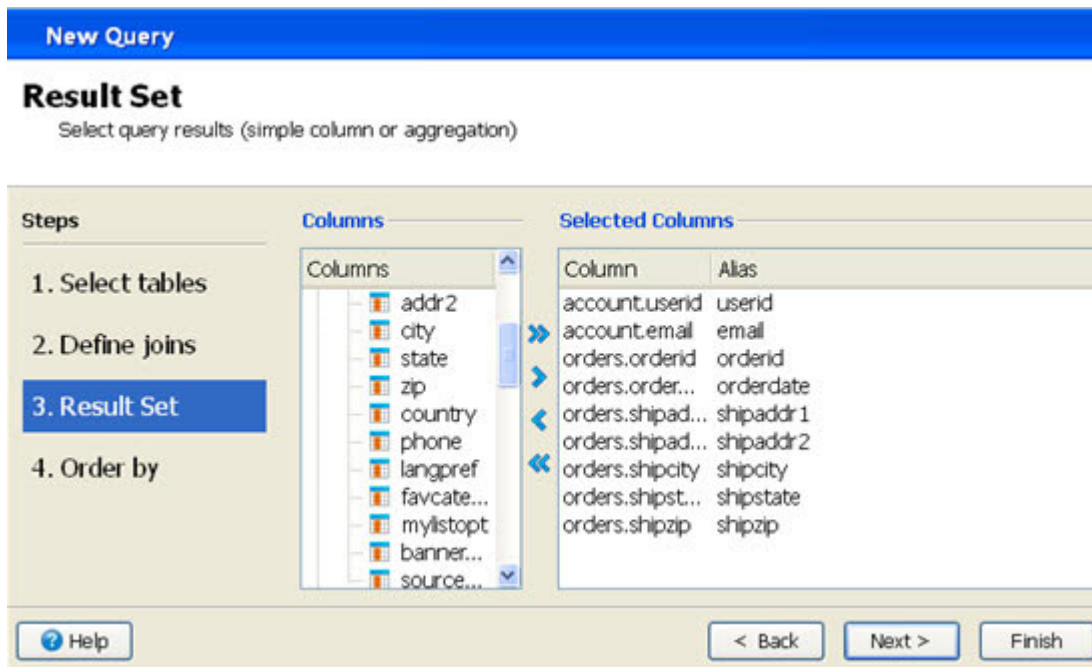


Figure 9-11. New Query wizard, Result Set page

Press **Next** to Continue.

## Order By Page

Select the column(s) by which the result set should be sorted.

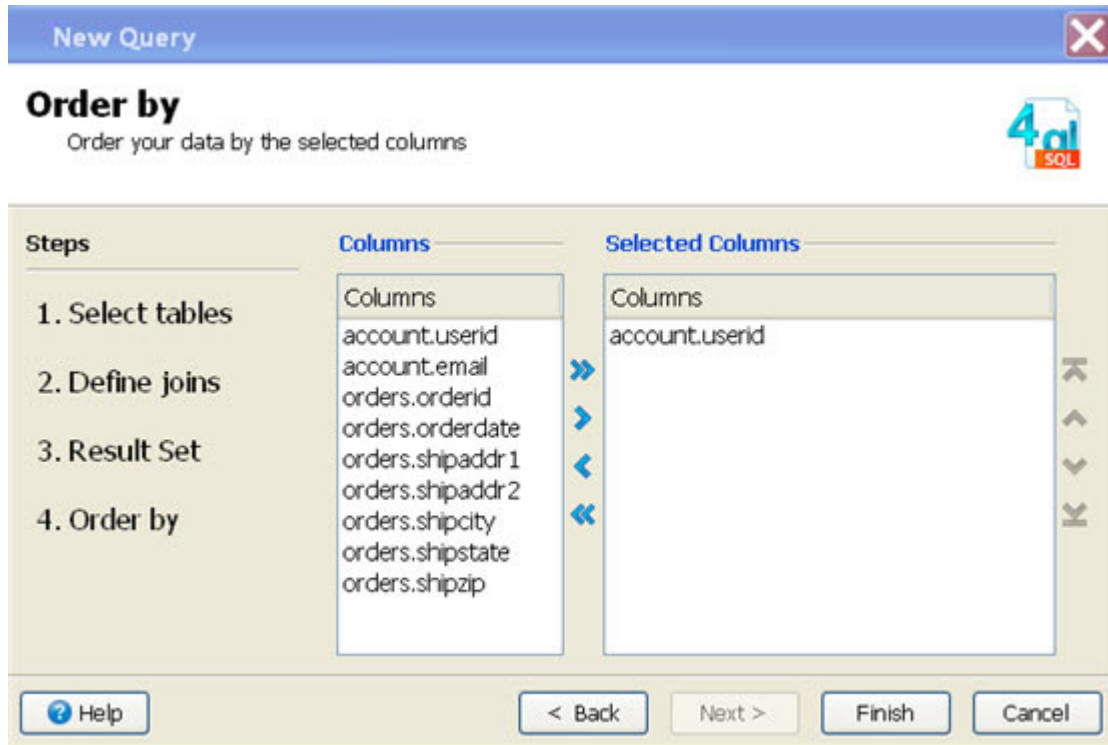


Figure 9-12. New Query wizard, Order By page

Press **Finish** to generate the code and close the Report Data Wizard.

The Wizard creates a BDL file (.4gl) containing the generated code, which you can save as part of your application files.

### Example Wizard Code

The following is the code that was generated by the example covered in “Using the Report Data Wizard” on page 9-28

```
01 SCHEMA officestore
02
03 TYPE officestore_data RECORD
04 account RECORD
05 userid LIKE account.userid,
06 email LIKE account.email
07 END RECORD,
08 orders RECORD
09 orderid LIKE orders.orderid,
10 orderdate LIKE orders.orderdate,
11 shipaddr1 LIKE orders.shipaddr1,
12 shipaddr2 LIKE orders.shipaddr2,
13 shipcity LIKE orders.shipcity,
14 shipstate LIKE orders.shipstate,
15 shipzip LIKE orders.shipzip
16 END RECORD
17 END RECORD
18
19 FUNCTION run_officestore_to_handler(handler)
20 DEFINE
```

```

21 data officestore_data,
22 handler om.SaxDocumentHandler
23 DECLARE cur CURSOR FOR
24 SELECT
25 account.userid,
26 account.email,
27 orders.orderid,
28 orders.orderdate,
29 orders.shipaddr1,
30 orders.shipaddr2,
31 orders.shipcity,
32 orders.shipstate,
33 orders.shipzip
34 FROM
35 account,
36 orders
37 WHERE
38 orders.userid = account.userid
39 ORDER BY
40 account.userid
41
42 START REPORT officestore_report TO XML HANDLER handler
43 FOREACH cur INTO data.*
44 OUTPUT TO REPORT officestore_report(data.*)
45 END FOREACH
46 FINISH REPORT officestore_report
47 CLOSE cur
48 END FUNCTION
49
50 REPORT officestore_report(data)
51 DEFINE
52 data officestore_data
53 ORDER EXTERNAL BY
54 data.account.userid
55
56 FORMAT
57 ON EVERY ROW
58 PRINT data.*
59 END REPORT

```

**Note:**

- Lines 3-17 define a user-type named **office\_data**, which is a RECORD consisting of sub-records for the fields that were selected in the Result Set page from the tables selected in the Table Selection page.
- Lines 19-48 define the **run\_officestore\_to\_handler** function.
  - Lines 23-40 declare a cursor for the SQL SELECT statement to retrieve the data. The table join and the sort order were specified in the Define Joins and Order By pages.
  - Lines 42-46 fetch the report data and send it to the REPORT function.
- Lines 50-59 define the REPORT function, which outputs the data to the Genero Report Engine.

See The BDL File for additional information about the BDL functions and statements used in conjunction with reports.

## Using the Example Code

You can create a report application using the generated BDL file, or the file can be incorporated as part of a larger application. We recommend that you use a different BDL file to contain the additional code that is required, rather than making changes in the generated file. This allows you to re-execute the Report Data Wizard without risking the loss of your changes.



### 1. Create the Report Data Definition file

Create the **.rdd** file that is used to populate the Data View of the Report Design Document, by executing the following from the command line:

```
fglcomp --build-rdd <generated_file>.4gl
```

Add the resulting *<sourcefile>.rdd* file to your application files.

### 2. Design the Report

Specify the **.rdd** file on the Data View page. When your design is complete, save the report definition file (**.4rp**) as part of your application. See *Designing a Report* for additional information about the design process.

### 3. Add Additional BDL Code to the program

- a. To configure the report, call the mandatory report API functions, which identify the report definition file and create the XML handler for the report:
  - `fgl_report_loadCurrentSettings` accepts the name of the Report Design document (**.4rp**) as a parameter. The return value is a boolean indicating whether the load of the settings from the **.4rp** file was successful.
  - `fgl_report_commitCurrentSettings` returns a `SaxDocumentHandler` object. Additional report API functions can be called to change the default output options, if needed.
- b. To connect to the Database, add a **CONNECT TO** or **DATABASE** statement to the program.
- c. Call the **run\_<databasename>\_to\_handler** function (which is in the file generated by the Report Data Wizard) to execute the report.

### Example MAIN function

The following simple function illustrates step 3:

```
01 MAIN
02
03 DEFINE handler om.SaxDocumentHandler -- report handler
04
05 DATABASE officestore --connect to the database
06
07 --call the mandatory functions that configure the report
08 IF fgl_report_loadCurrentSettings("myreport.4rp") THEN -- if file loaded OK
09 LET handler = fgl_report_commitCurrentSettings() -- commit the
 -- file settings
10 ELSE
11 EXIT PROGRAM
12 END IF
13
14 -- run the report by calling the report driver contained in your
 -- generated function
15 IF handler IS NOT NULL THEN
16 CALL run_officestore_to_handler(handler)
17 END IF
18
19 END MAIN
```

---

## Modifying the Output

Topics:

- “Change Report Output Options” on page 9-34
- “Using Localized Strings” on page 9-41

## Change Report Output Options

### Topics:

- Changing Options
- Some Examples
  - Send Output Directly to Printer
  - Set Page Margins
  - Set Print Quality
  - Print Double-sided
  - Select a Paper Tray
- Support for Custom Paper Sources
- SVG Preview and other Options
- Sending Report Data to an XML File
- Sending Report Data to an HTML File
- Sending Report Data to an Excel Spreadsheet
- Outputting Report Data in Microsoft RTF format
- Creating Multi-Page ISO Reports

See also: Designing a Report, Design HowTo, Configuring Fonts and Printers

### Changing Options

You can change the default paper settings for report design documents in Report Writer using the **Tools>>Preferences, Report Writer** main menu option. Change output options (paper settings and output format settings) for a specific report design document through the **File>>Report properties** main menu option.

More extension changes to the output options for a report can be made in your BDL report program (.4gl), using the BDL functions provided in the Reporting API. The BDL functions in the helpers.42m file allow you to change:

- Page settings
- Output device options
- Printer settings

The BDL functions to change the default output options are sandwiched between the mandatory configuration functions in your BDL program, as follows:

```
IF fgl_report_loadCurrentSettings(r_filename) THEN -- load the .4rp file mandatory
 CALL fgl_report_selectPreview(preview) -- functions to change the output
 -- options are called here
 LET handler = fgl_report_commitCurrentSettings() -- commit changes mandatory
END IF
```

### Some Examples

- **Send output directly to the printer** - you can send the output to a printer without previewing by calling the fgl\_report\_selectDevice function as part of your report configuration function with the parameter "Printer":  
CALL fgl\_report\_selectDevice("Printer")

**Note:** For other output formats, the parameter can be one of: "Printer", "PDF", "Image", "SVG", "HTML", "XLS" or "XLSX" (output to an Excel spreadsheet), "RTF" (output in Microsoft RTF format)

•

**Set page margins** - the logical margins of the report page are read from the .4rp file (Report Design document.) The `fgl_report_setPageMargins` function overrides those margins. The syntax is:

```
CALL fgl_report_setPageMargins (topMargin String, bottomMargin String,
 leftMargin String, rightMargin String)
```

```
CALL fgl_report_setPageMargins (".05cm", ".05cm", ".05cm", ".05cm")
```

- **Set Print Quality** - the `fgl_report_setPrinterQuality` function is used to control the quality used by the printer. By default this is not set. Setting this option reduces the set of usable printers to those matching it.

One of the following values are passed to the function as a parameter:  
`draft` | `high` | `normal`. The value is passed as a String.

```
CALL fgl_report_setPrinterPrintQuality("high")
```

- **Print Double-sided** - the function `fgl_report_setPrinterSides` is used to specify whether the report pages should be one-sided or two-sided. By default this is not set. Setting this option reduces the set of usable printers to those matching it. One of the following values are passed to the function as a parameter. The value is passed as a String:

- **one-sided**: Each consecutive print-stream page is printed on the same side of consecutive media sheets.
- **two-sided-short-edge**: Consecutive pairs of print stream pages are printed on the front and back sides of consecutive media sheets. The orientation of the pages is correct as if the pages were to be bound along the short edge of the paper.
- **two-sided-long-edge**: Consecutive pairs of print stream pages are printed on the front and back sides of consecutive media sheets. The orientation of the pages is correct as if the pages were to be bound along the long edge of the paper.

```
CALL fgl_report_setPrinterSides("two-sided-long-edge")
```

- **Select a Paper Tray** - function `fgl_report_setPrinterMediaTray` controls what tray of the printer to use. This function and the functions `fgl_report_setPrintermediaTray` and `fgl_report_setPrintermediaName` are mutually exclusive. Setting this option reduces the set of usable printers to those matching it.

One of the following values are passed to the function as a parameter:  
`bottom` | `envelope` | `large-capacity` | `main` | `manual` | `middle` | `side` | `top`. The value is passed as a String.

```
CALL fgl_report_setPrinterMediaTray("top")
```

## Support for Custom Paper Sources

On Windows systems the reporting API function `fgl_report_setSVGPaperSource()` can be used to identify a custom paper source for your report. This function must be used in conjunction with the following functions:

- `fgl_report_setSVGPrinterName(printerName)` - to specify the printer
- `fgl_report_configureSVGPreview("PrintOnNamedPrinter")` - to bypass the preview and print silently on the named printer

See Example code.

## The printerinfo Script

On Windows systems the script `<GRE-install-dir>/bin/printerinfo` is provided to list the supported paper sources that can be changed programmatically using this function. This script is executed from the command line.

The output of the script lists the device-specific source names in the left column, with the general constant in brackets on the right. Either can be used to identify the paper source in calls to `fgl_report_setSVGPaperSource()`:

The following excerpt lists seven paper sources for the printer **Balzac**:

```
PRINT SERVICES
Found 9 print services
 Service no 1 Printer : Balzac
 Service supports the following 18 paper sizes
 Letter 8.5 x 11 inches, 215.9 x 279.4 mm
 Legal 8.5 x 14 inches, 215.9 x 355.6 mm
 Executive 7.5 x 10 inches, 190.5 x 254 mm
 A4 (210 x 297 mm, 8.26 x 11.69 inches)
 A5 (148 x 210 mm)
 B5 (176 x 250 mm, 6.93 x 9.84 inches)
 Folio (210 x 330 mm)
 Comm10E (105 x 241 mm, U.S. Common 10 Envelope)
 DLE (110 x 220 mm)
 Custom Unknown, or a user defined size (9 formats)
 Service supports the following 7 paper sources
 Automatically Select (FormSource)
 Auto Select (Auto)
 Tray 1 (OnlyOne)
 Tray 2 (Lower)
 Multi-Purpose Feeder (Cassette)
 Manual Paper (Manual)
 Manual Envelope (EnvelopeManual)
```

Figure 9-13. printerinfo output

### Example Code

```
CALL fgl_report_setSVGPrinterName("Balzac")
CALL fgl_report_setSVGPaperSource("Tray 2")
CALL fgl_report_configureSVGPreview("PrintOnNamedPrinter")
```

The following excerpt of the printerinfo output is for a printer that has 27 paper sources. Since there are only 15 predefined constants for the paper sources, most of the paper sources are identified by integer constants, which can be used in calls to `fgl_report_setSVGPaperSource(.)`. Using the integer constant can prevent encoding issues when the device-specific name contains non-ASCII characters, such as "Sélection automatique".

```

Service no 3 Printer : HPCLJ4650DTN (Default Printer)
Service supports the following 19 paper sizes
Letter 8.5 x 11 inches, 215.9 x 279.4 mm
Legal 8.5 x 14 inches, 215.9 x 355.6 mm
Executive 7.5 x 10 inches, 190.5 x 254 mm
A3 (297 x 420 mm)
A4 (210 x 297 mm, 8.26 x 11.69 inches)

A5 (148 x 210 mm)
B4 (250 x 353 mm)
B5 (176 x 250 mm, 6.93 x 9.84 inches)
Comm10E (105 x 241 mm, U.S. Common 10 Envelope)
DLE (110 x 220 mm)
Custom Unknown, or a user defined size (9 formats)
Service supports the following 27 paper sources
Automatically Select (FormSource)
Printer autom. Selection (262)
Man. Feed in Tray 1 (261)
Tray 1 (260)
Tray 2 (259)
Tray 3 (258)
Not specified (1281)
Normal (1280)
Preprinted (1279)
Letterhead (1278)
Transparent Foil (1002)

```

Figure 9-14. printerinfo output

## SVG Preview and other SVG Options

Beginning with version 2.3, new options (data type String) have been added to the SVG previewer:

- **Preview:** The report is shown in a folder tab of the main preview window.
- **ShowPrintDialog:** This option pops up the system print dialog, allowing the user to select and configure a printer. If confirmed, the document is printed in the background.
- **PrintOnDefaultPrinter:** This option prints the report silently on the default printer. The previewer main window is not shown.
- **PrintOnNamedPrinter:** This option prints the report silently on a specific printer. The previewer main window is not shown.

These options can be configured by the API function `fgl_report_configureSVGPreview`. For example:

```
CALL fgl_report_configureSVGPreview("ShowPrintDialog")
```

Additional API functions provide more SVG options:

- `fgl_report_setSVGCopies` - function to specify the number of copies to be printed
- `fgl_report_setSVGPageRange` - function to select which pages should be printed
- `fgl_report_setSVGPaperSource` - function to select the input source of the printer
- `fgl_report_setSVGPrinterName` - function to select a specific printer by name

- `fgl_report_setSVGSheetCollate` - function to control the collation of multiple copies

## Send Report Data to an XML File

The demo program **OrderReport.4gl** gives the user the option to output the report data to a file (`OrderReportData.xml`) in XML format.

The Reporting API function `fgl_report_create ProcessLevelData File()` instructs the Genero Report Engine to execute the report, outputting the data specified in the PRINT function of the REPORT program block to an XML file rather than to a Report Design Document. The output file is stored in your current project. The name of the data file is passed to the function as a String.

Changing the `SimpleReport.4gl` example program to use this technique, the MAIN program block becomes:

```

13 MAIN
14
15 DEFINE handler om.SaxDocumentHandler, -- return value
16 dataFile STRING -- name of XML file to be created
17
18 --configure the report engine to output the XML file
19 LET dataFile = "./OrderReportData.xml"
20 LET handler = fgl_report_createProcessLevelDataFile(dataFile)
21
22 -- run the report, creating the XML file
23 CALL runReportFromDatabase()
24
25 END MAIN

```

## Sending Report Data to an HTML File

HTML output can be selected by passing the string "HTML" to the function `fgl_report_selectDevice`. You can use this function to output any existing report (including ASCII) as an HTML document.

Graphical elements such as bar codes are implemented by rendering them as images. These images are included in the document. An option to generate the images on disk and to use external URLs instead is provided by the function `fgl_report_configureHTMLDevice`.

When elements overlay each other (or text strings intersect), the default behavior is to create one image from the two elements and to include the image. An exception to this occurs when the overlaid element is an image: the "background" image will be removed from the resulting document.

There is an option in the function `fgl_report_configureHTMLDevice` to ignore the horizontal or vertical alignment in the document. The default is FALSE for HTML document output.

## Sending Report Data to an Excel Spreadsheet

The report output can be sent to an Excel spreadsheet by passing the string "XLS" or "XLSX" to the function `fgl_report_selectDevice`. The output will be in the specified Excel format, with the XLS format limited to 65,536 rows. In contrast, for the XLSX format the document is generated with constant memory consumption, so that very large documents can be produced without exhausting heap space.



**Note:** The XLSX format can only be opened with newer versions of Microsoft Excel (2007 or later). A backward compatibility pack can be downloaded from Microsoft; however, the XLSX format will then be subject to the 65,536 limit of the earlier versions.

Any existing report, including ASCII, can be output to the spreadsheet.

The layout of the cells (size, font color, etc.) can be predictably controlled from the Report Designer. The goal is to put the report layout into the cells of the spreadsheet efficiently. There is an option in the functions `fgl_report_configureXLSDevice` and `fgl_report_configureXLSXDevice` to ignore the vertical or horizontal alignment. The default is TRUE for Excel document output, since the emphasis is on the ability to compute the values in the cells, rather than primarily on the appearance.

When items in the report design overlap, by default the placement is preserved in the spreadsheet, but not the alignment. To make the necessary decisions, Studio marks the column and row boundaries internally with tabs. If there are two consecutive tabs that have no element that bounds on them, the column or row will collapse. The implication of this is:

- Rivers of white space (empty columns, empty rows) are eliminated when possible.
- Elements that overlap may be placed in the same column, but they will still maintain their relative placement (above/below).
- Contiguous items will never be placed in the same column, they will maintain their relative placement and alignment.

The values in the cells are generated for the report, not the Excel formulas. Graphical elements such as bar codes are implemented by rendering them as images. Business charts are currently drawn as tables; we cannot create charts or pivot tables.

The following code fragment illustrates the functions enabling Excel output:

```
...
11 CALL fgl_report_selectLogicalPageMapping("multipage")
12 CALL fgl_report_configureMultipageOutput(2,4,TRUE)
13 CALL fgl_report_selectDevice("XLS")
...
```

## Creating a report with data but no titles

To obtain a spreadsheet containing only the data in a flat table:

- Create a new empty report (from the main menu, **File>>New, Reports, Empty report**).
- Configure a large custom page size, **300cm x 200cm**, for example (from the main menu, **File>>Report Properties, Paper Setting**).
- From the **Toolbox**, drag a Stripe container into the report design.
- In the **Data View**, select the button create a table column value object; then, double-click the fields to select them for the report.
- In the **Structure View**, associate a Trigger node with the stripe, to specify when the contents of the stripe should print (ON EVERY ROW, for example)

## Creating a report with tables that contain headings

To obtain a spreadsheet containing tables with column headings, use the standard list template (from the main menu, **File>>New, Reports, List report**). Use a large custom page size, **300 cm x 200cm**, for example. The output will contain a title for the document and headings for each column.

## Creating a single-sheet document

Currently each page produces a separate Sheet in Excel. If the page breaks are not desirable, you can change the page size to a larger custom value. However, creating a huge page can adversely affect memory reclamation and performance. To avoid this problem, we recommend that you use a standard page size and set the **mergepages** parameter of the function `fgl_report_configureXLSDevice` or `fgl_report_configureXLSXDevice`. This will merge the pages into a single-sheet result.

**Note:** If you have specified "XLS" in the call to `fgl_report_selectDevice` to specify the Excel format, and the resulting sheet has more than 65536 rows, the exceeding rows will spill over into extra sheets; if you have specified "XLSX" as the Excel format, the size of the sheet is not limited by available memory.

## Outputting Report Data in Microsoft RTF Format

You can use the API function `fgl_report_selectDevice("RTF")` to output the data in the RTF Format; configure the output using the function `fgl_report_configureRTFDevice`.

In order to prevent exhaustion of main memory when processing large documents, the processor can be instructed to swap parts of the document to a temporary disk file when the document size exceeds this threshold. Use the function `fgl_report_setRTFMemoryThreshold` to specify the threshold.

## Create Multi-Page ISO reports

For reports printed on ISO and JIS-sized pages, you can configure the output to print several logical pages per physical page. Use the following Reporting API functions to change the output options:

- `fgl_report_selectLogicalPageMapping` - use the parameter **multipage** to change the logical page mapping, allowing the printing of multiple pages per physical page.
- `fgl_report_configureMultipageOutput` - to configure the required number of pages. The parameters are:
  1. **pageExponent** String - Specifies the number of pages to print. The actual number of pages is calculated by multiplying the page exponent number that you specify by 2. In the example there will be  $2 * 2 = 4$  pages per physical page.
  2. **isoNumber** Integer - Specifies the ISO number. This parameter is optional (indicated by passing a null value). If no value is specified, the page size of the logical page is taken from the .4rp file (if specified). In the example, 4 refers to ISOA4.
  3. **portrait** boolean - TRUE specifies that the page is in portrait orientation; FALSE= landscape. This parameter is optional (indicated by passing a null value). If no value is specified, the orientation of the logical page is taken from the .4rp file.

Example function:



```

20 FUNCTION configure_report()
21 DEFINE handler om.SaxDocumentHandler -- configure report engine
22
23 IF fgl_report_loadCurrentSettings(NULL) THEN -- there is no .4rp file
24 -- change default output options
25 CALL fgl_report_selectLogicalPageMapping("multipage")
26 CALL fgl_report_configureMultipageOutput(2,4,TRUE)
27 LET handler = fgl_report_commitCurrentSettings() -- commit settings
28 ELSE
29 EXIT PROGRAM
30 END IF
31 END FUNCTION

```

## Using Localized Strings

### Topics:

- Localized Strings
  - The Source String File
  - Using the translate function
  - Entries in fglprofile
- Finding the Translated String

### Localized Strings

The **Localized Strings** feature allows:

- the **captions** (titles of report objects) to be customized.
- The translation of the text property for Word Boxes and WordWrap Boxes to be customized.

This feature can be used to implement internationalization in your application, or to use site-specific text (for example, when business terms change based on territory).

You could have multiple external string files, each containing the translated string in a different language. The DVM (runtime system) searches the files in order to assign text in the report definition. The localized text replaces the captions and other specified text at runtime, choosing the correct string file based on entries in the file fglprofile.

### The Source String file

A text file, with a .str extension, defines the string to be displayed for the report object captions or the specified Word Box or WordWrap Box text. You can check the Text property of the report object to determine the exact string to be replaced.

#### Syntax:

*"identifier" = "string"*

where:

- *identifier* is the value of the Text property of the caption or label that you want to localize
- *string* is the text that you want to be displayed instead

The backslash "\" is accepted as the escape character, to define non-printable characters.

Example file **captions\_en.str**:

```
"Userid" = "Client ID"
"Shipcity" = "City"
```

At runtime, the text 'Userid' or 'Shipcity' would be displayed in the report as 'Client ID' and 'City'.

This source string file should be added to the Studio Project Manager application node for your report program, so it can be compiled when the application is built. The compiled file (.42s) is not linked into the Genero executable.

**Note:** Although you can call the API function `fgl_report_setCallbackLocalization()` in your 4GL source code to specify that the caption of the report object be used instead, we recommend using localized strings.

### Using the translate function

Entries in the source string file can also be used by the **translate** PXML function to replace text in an expression that sets a property value. For example, if you set the text of a Word Box) to the following value:

```
"hello "+"world".translate()
```

and you have the following entry in the .str file:

```
"world" = "universe"
```

then the final text displayed on your WordBox will be “hello universe”.

### Entries in fglprofile

You can specify the compiled String File, or a list of these files, with entries in an **fglprofile** configuration file. For example:

```
fglrun.localization.warnKeyNotFound= false -- don't display warnings on errors
fglrun.localization.file.count = 1 -- number of string files
fglrun.localization.file.1.name = "captions_en.42s" -- name of the string file
```

See the FGLPROFILE topics in the *Genero Business Development Language User Guide* for complete information about the fglprofile file.

See the Localized Strings topics in the *Genero Business Development Language User Guide* for complete information and examples.

### Finding the translated string

The search behavior that Report Writer uses to find a string in a compiled string file (.42s) is similar to that of Genero BDL. Once the compiled string files are located, Report Writer uses the first occurrence of the string that it finds. The .42s files are sought in the following order:

1. Any localization (.42s) files specified in a BDL configuration file. First, the current directory, and then all directories in DBPATH or FGLRESOURCEPATH, are searched for the specified .42s files. The precedence of BDL configuration files is:
  - a. the BDL configuration file, **<FGLDIR>/etc/fglprofile**.
  - b. the BDL configuration file specified by the FGLPROFILE environment variable, if set.
  - c. any BDL configuration file having the same name as the report program, and stored in the directory **<FGLDIR>/defaults**.

2. Any localization (.42s) file that has the same name as the report definition file, minus the .4rp extension. First, the current directory, and then all directories listed in the DBPATH or FGLRESOURCEPATH environment variable, are searched for the *<report-name>.42s* file.
3. Any localization (.42s) file that has the same name as the report application, minus the .42r extension. First, the current directory, and then all directories in DBPATH or FGLRESOURCEPATH are searched for the *<application-name>.42s* file.

---

## Report Design Document

### Topics:

- "Overview of Genero Report Designer"
- "Designing a Report" on page 9-45
- "Design How-To" on page 9-62
- "Working with Business Graphs" on page 9-74
- "Expressions in Properties" on page 9-84

## Overview of Genero Report Designer

### Topics:

- Genero Reports
- Launching the Report Designer
- The Report Design Window
  - Work Area
  - Structure View
  - Data View
  - Properties View
  - Output View
- The Report Output

See also: Designing a Report, Design HowTo, Creating Business Graphs

## Genero Reports

Genero reports can be:

- **General Documents** - such as invoices, corporate documents, accounting reports, You define the contents, page size, page headers and footers, output format, and other attributes
- Pre-printed forms - you define the content for a pre-printed form
- Labels - you define the content and label size to be printed on pages of labels. Refer to the Genero Studio >> Report Writer documentation topic "Configuring for Labels" for more details.
- "Compat" Text - you output a legacy Genero (.4gl) report in ASCII text, using the Report Writer. Refer to the Genero Studio >> Report Writer documentation topic "Writing a BDL Report" for more details.
- Business Graphs - you specify the type of graph and the data items

A report can be displayed in various output formats.

## Launching the Report Designer

Choose **File>>New, Report** from the Studio main menu, or double-click an existing report design document (.4rp file) in the Project Manager, to open the Report Design Window.

## The Report Design Window

You create a **Report Design document** (.4rp) which defines the report page as consisting of box-shaped containers, to hold the data objects and text or image objects that make up the report.

### Work Area

- Report elements (containers such as Mini Page and Layout Nodes, and their child elements such as Word Box and Image Box) and data values from the Data View can be dragged and dropped onto the report page and rearranged.
- Page Headers and Footers can be defined.

### Structure View

The organization of the report can be seen in the Structure View. Containers and their contents can be moved around in the View to insure that they print in the correct order.

Triggers in the Structure View specify what should be printed when a trigger event (change in data) occurs.

### Data View

Your Report Design Document will use an .rdd file to determine the data schema for a report. The same .rdd file can be used for multiple report definitions.

The available data objects are shown in the Data View, containing the Data Schema (.rdd file). At runtime the data is sent to the report by a BDL report program. The BDL Program page explains the ways that report data can be fetched.

### Properties View

Each report element has properties, displayed in the Properties View, with values that you can change.

In addition to literal values, expressions can be used to change the value of report elements properties. For example, the appearance can be changed conditionally, by creating an expression to turn the background color red if the value for the form element is greater than 1000.

### Output View

This view is used by any module to display its output. The output of the various modules (messages, errors, results of commands, and so on) is displayed as tabs in the view.

The **Document Errors tab** displays any error messages for the report design document.

See: Designing a Report, Design HowTo

## The Report Output

A report can be output in various formats and different page sizes, to various output devices:

Default output and printer options can be set for the Report Design document using the File>>Report Properties menu options:

- Paper settings
  - You can select the report page size (standard or custom), and set the margins.
  - You can set the unit of measure for the page (centimeters or inches), and define the page orientation.
- Output configurations
  - You can create a report in **SVG format**, and display it in the Genero Report Viewer. The Genero Report Viewer is provided as part of the Genero Desktop Client. This provides a preview of the report, and allows the user to select a printer.
  - You can create a **PDF file** that can be previewed in Acrobat Reader, or an **image (.jpg)** file.

See Designing a Report, Design HowTo

## Designing a Report

Topics:

- The Report Design Window
- The Work Area
- The Toolbox
- Placing Elements on the Report Page
  - Specific Positioning or Automatic Layout
  - Dragging Multiple Objects
  - Choosing Containers
  - Choosing Other Report Elements
- “Changing a Report Element type” on page 9-51
- “Changing a Property Value (The Properties Panel)” on page 9-51
  - Using Expressions for Property Values
  - Adding Report Document Metadata
- Adding Report Data (Data View)
  - Adding Report Data and Captions
  - Adding Report Data and Captions as a Table
  - Support for Arbitrary XML Data Sources
- Organizing the Report (Report Structure View)
  - The Tree Structure
  - The Data Triggers
  - Page Headers and Footers
  - Example Report Structure
  - The Printed Page
- Using Page Numbers
- Report Design Metadata
- Configuring the Output

- Paper Settings
- Output Configuration

See also: Design HowTo

## The Report Design Window

Choose **File>>New, Report** from the Studio main menu, or double-click an existing report design document (**.4rp** file) in Project Manager, to open the Report Design Window. When creating a new report, you can choose to begin with a blank report, or with a list report template that has the basic structure already in place. The following views and windows provide the tools and work areas for the report. Use the **Window>>Views** main menu option to display and hide views.

- The Work Area - main window of the report
- Structure View - a tree of the report containers and their contents
- Properties View - a list of the properties for a selected report element
- Data View - a list of the data objects that are available for the report
- Toolbox - a list of the containers that are available
- **The Output View** - tabs display the output, document design errors, and the task manager.

**Note:** Metadata for the report design document can be stored in the properties of the

## The Work Area

The work area in the Main Window provides a GUI interface to the report.

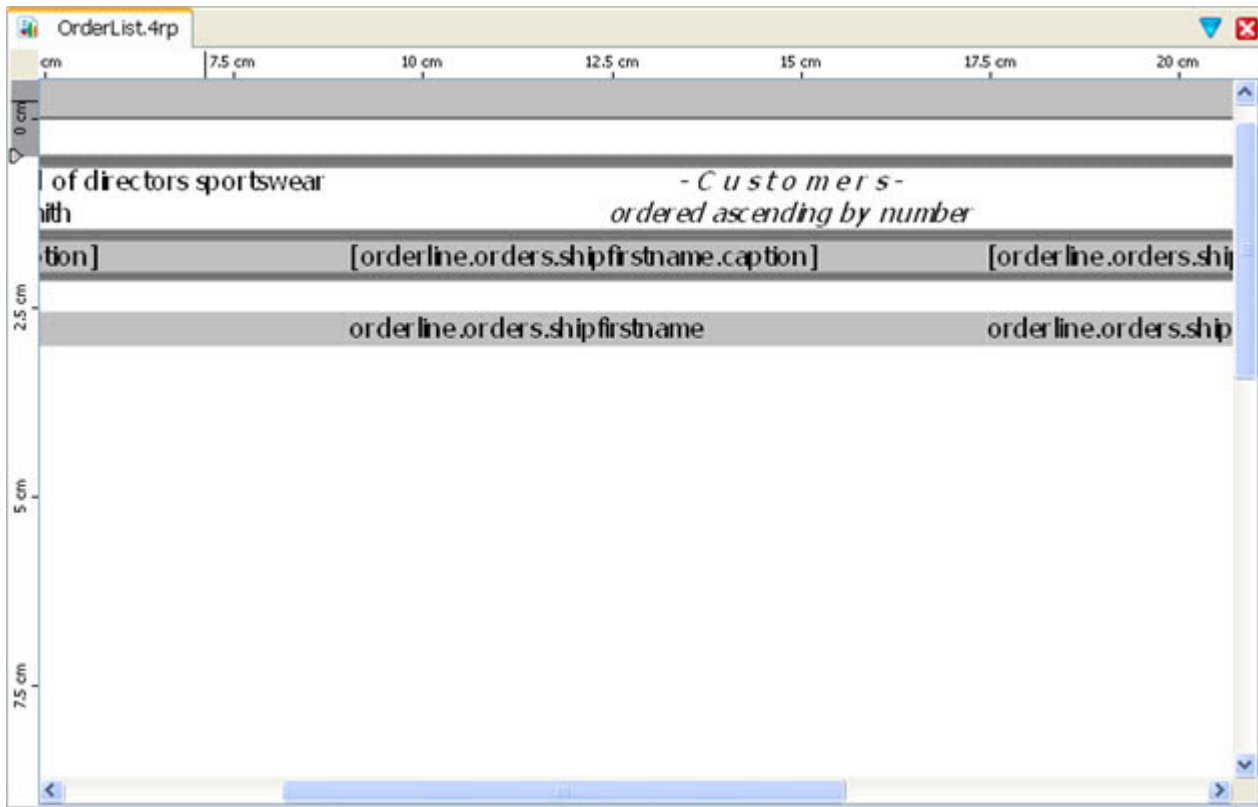


Figure 9-15. Report Designer work area

You design a report by dragging and dropping Containers from the ToolBox into the Work Area, stacking and arranging them to create the report page. Then, other report elements such as Word Boxes, Decimal Format Boxes, or Images are dragged and dropped into the containers.

From the Data View, you can drag and drop Data Items into a container, if you have specified the .rdd file containing the data schema.

When you select a report element in the work area, its properties are listed in the Properties View, where you can change a property value. For example, a WordBox has a text property where you can enter the text to be displayed in the report.

If you select multiple elements, all items in the current selection are affected by the current operation, such as moving, sizing, changing the type or text.

Use the **View>>Toggle View** menu option or the **Toggle View icon** in the Studio toolbar to toggle the work area between the report design and a preview of the report. When you preview a report, sample data is displayed on the page.



Figure 9-16. The Toggle View icon

**Zoom buttons** on the toolbar allow you to zoom in and out on the report design document:

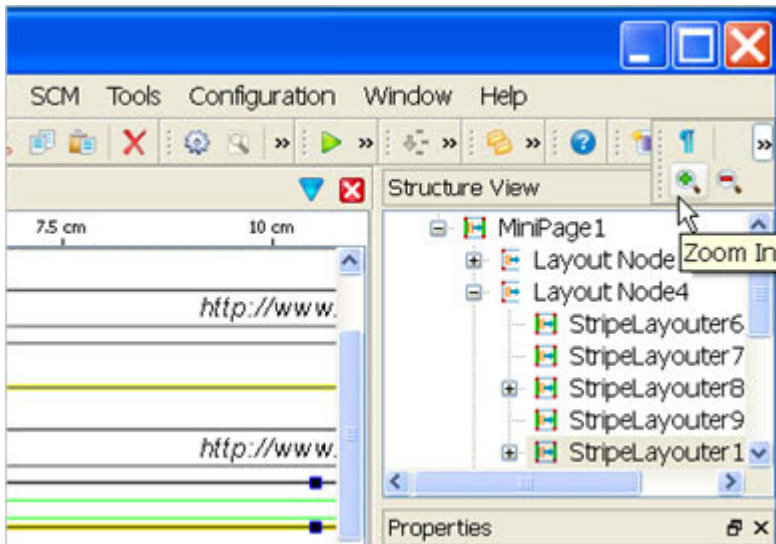


Figure 9-17. Zoom buttons

## The Toolbox

The Toolbox tab provides the following report elements that can be dragged and dropped into a report design document:

- **Containers** - for grouping other elements on the report page; see Choosing Containers
- **Drawables** - the report elements contained by a container; see Choosing Other Report Elements
- **References** - to define layout-specific elements; see Choosing Other Report Elements
- **Business Graphs** - the specific charts and items; see Working with Business Graphs
- **Bar Codes** - specific bar codes; see Bar Code Values

## Placing Elements on the Report Page

**Specific Positioning or Automatic Layout:** The position of the report element can be **specific**, to enable reports that use pre-printed forms, or **dynamic**, adjusting as needed based on the length of the report element:

- **Dynamic Layouting** - if you Ctrl-drag the report element onto the Design Window, the element is positioned relative to the existing elements. Because the design changes dynamically based on the actual size of the specific report items, this is the recommended method. When you drag the element, a moving red line indicates where the element will be located when you drop it. A colored dot on the element indicates its attachment point. If you select a container and then double-click an element in the ToolBox or Data View, the element will be automatically positioned after the last existing child object in the container.
- **Specific Positioning** - if you drag and drop report elements on the Design Window using the mouse, you can position the element at a specific spot on the virtual grid of a container. This is recommended when you need to match the report design to a preprinted form. When you drop the element, it will snap to the closest point on the grid. A **red** dot on the element indicates its attachment point.



As you drag an element, a moving thin black line helps you line it up with other elements on the report if desired. The **X** and **Y** properties of the element in the Properties View indicate its location relative to its parent. These are automatically calculated when you drop an element into a container, or move it around. When you move it inside a container, the lines of the container are highlighted in yellow :

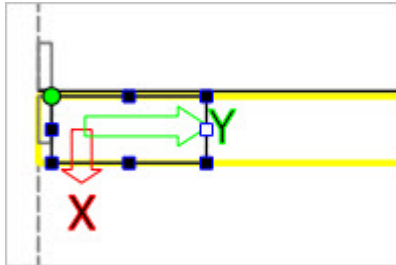


Figure 9-18. Highlighted container

All elements that are dragged from the Toolbox or Data View have the `floatingBehavior` property set to **enclosed** (the object will be enclosed in its parent.)

- **X-Axis and Y-Axis arrows** - These indicate the direction of the X-Axis and Y-Axis. On containers the Y-Axis arrow indicates the filling or `layoutDirection` of the container. For example, a `Stripe` lays out its children next to each other left-to-right within the container, and the Y-Axis arrow points to the left. Other containers have the Y-Axis arrow pointing down, as they lay out their children next to each other in a top-to-bottom direction.
- **Attachment point** - The attachment point at an intersection of the X-Axis and Y-Axis is indicated by a green dot (for dynamic layouting) or a red dot (for specific positioning.) If you drag the edge of the element to expand its size, the attachment point remains fixed. You can move the attachment point using the right-click context menu.

Elements on a report have a **right-click menu** of options that allow you to:

- Align elements within a vertical or horizontal container, and move the attachment point on the element
- Change the width and height of elements
- Change the focus to a different container or other element
- Change the object type

**Dragging Multiple Objects:** You can drag and drop a multiple selection of objects from the Data View to the Design Work area or Structure View. Use the Ctrl and Shift keys to select the objects, and drag them to the desired location. An object will be created for each element selected, following the order in which they were selected.

If you drag to the Design Work area using Specific Positioning (the Ctrl key is pressed when dropping), an additional container is created for each element object. If you have chosen to create a form field object, a horizontal container is used so the elements will appear in a line. When you create a column object, a vertical container is used so the elements will appear stacked.

If you use Dynamic Layouting, or drag to the Structure View, the behavior is the same as if each element has been dragged individually.

**Choosing Containers:** The template for new reports already has the Page Root in the report structure, which is a Mini Page container that is already expanded to its maximum width and length. The other containers used for the report are dropped within the Page Root. although you could drop all the elements directly on the Page Root, building up the report in blocks of containers allows you to group elements together, move the groups around, and align the children elements within a parent container.

- For the **main container of a report page** - a Mini Page (named PageRoot) is used. The default Layout Direction when you add elements to a Mini Page is top to bottom, down the length of the page. This container propagates - when a report is printed, if a Mini Page is full, a copy is made and the leftover material flows to the copy or copies as needed.
- **For Page headers and footers** - use Layout Node (Vertical Box) containers, which define a rectangular area in the report in which the elements are laid out top-to-bottom by default. A Layout Node does not propagate, so the contents of headers or footers can not spill over into another page. Within the header or footer, use Stripe (Mini Page) containers for elements that should be laid out left-to-right across the page. Adding a Stripe to a Layout Node automatically extends the Layout Node across the page.
- **For Table rows** - use a Stripe container- this is a Mini Page with the Y-Size set to **max**, so it stretches across the report page. Items added to a Stripe are laid out left-to-right. if the elements within a Stripe exceed the page width, the row is broken into the next line.
- For a **report page with a different layout** (a different first page, for example) - use a Mini Page for that layout.

**Choosing Other Report Elements:** From the Data View:

Use the icons at the top of the data view to specify whether you wish to drag and drop a reference to a data value, or a data item's caption (title). You can also choose to have the values and captions aligned in a table format. See Data Values and Captions.

- **For data values passed to the report** - when you place the data item on the design page, it is automatically enclosed in a Word Box/ Word Wrap Box or Decimal Format Box container, depending on the data type.
- **For captions (titles) for data items** - the caption is automatically enclosed in a Word Box.

From the **Toolbox**:

- **For additional text** - use a Word Box or Word Wrap Box. You can enter the text when you drag the report element onto the report design page, or you can set the value of the text property in the Properties View. You can also double-click on the text in the report design page to edit it.
- **For Numeric data** - use a Decimal Format Box, which makes it possible to parse and format numbers in any locale. The value property specifies the number. You can define the printed format, including decimal places, by setting the value of the format property in the Properties View. You can also double-click on the text in the report design page to edit it.
- **For Images** - use an Image Box, which allows you to specify the image to be printed by setting the location property.
- **For page numbers** - use a Page Number Box to automatically display the correct page number for each report page.

- For **business graphs** - choose the Chart element for the specific type of graph. See Working with Charts.
- For values like **"Total from previous pages"** or **"Totals until this point"** - use an InfoNode and Reference Box. These two elements work together to enable this type of content. See Design HowTo for additional information.

## Changing a Report Element type

To change a report element from one type to another, right-click the element in the report design document work area, and select **Convert To**; choose the new type from the list that displays. To convert the type of multiple report elements at once, Ctrl-click each element, and right-click to display the context menu.

The name property of the element will not change, unless the name of the old node is of the format [Type][Number]. A node named WordBox12 would be renamed, for example.

By default the new object type will have the properties set that it has in common with the old type. For some type conversions, additional properties may be set. For example, when converting a Decimal Format Box to a Word Box, the **value** property is converted to a string value and assigned to the **text** property.

## Changing a Property Value (The Properties Panel)

Select a report element in the Structure View or the Work Area to display the property values in the Properties View.

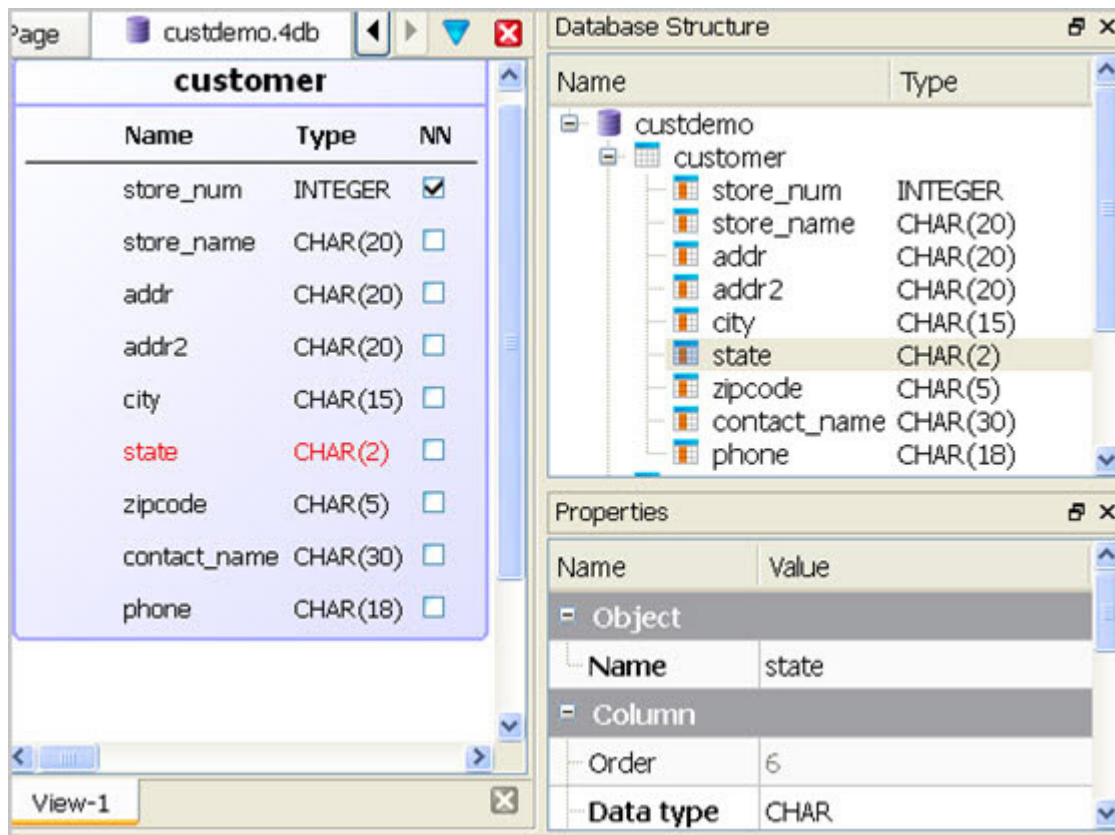


Figure 9-19. Properties View

The values for the properties of a report element can be changed by typing the new value in the Value column. The value may be a literal value, or it may be an expression written using the RTL Expression Language. All the properties are assigned a type, and the values entered must be valid for that type. The type of each property is listed in the Properties page.

**Note:** For WordBoxes, WordWrapBoxes, and DecimalFormatBoxes, if the text property is a literal value it may be edited directly in the report design document. Double-clicking on the element selects the text and places the input cursor in the document. The layout of the document is updated on each keystroke.

**Using Expressions for Property Values:** A valid expression for a property value is a sequence of operands, operators, and parentheses that the runtime system can evaluate as a single value. The RTL Expression language used in Report Writer closely follows the Java syntax for expressions and evaluation semantics.

- Arithmetic formulas can be used.
- Conditional expressions allow you to express IF/ELSE statements.
- Genero BDL variables can be used
- Functions from the Reporting API can be used.

Press the **fx** button to open the Expression Editing Window. See Using RTL Expressions for additional information.

**Adding Report Metadata:** Five new string properties for Report Metadata - title, author, creator, subject and keyword - can be specified for the document root in the Report Designer Properties View.

The meta data is inserted into the final document (e.g. PDF, SVG) if the format supports meta data. In the case of SVG, the **title** property is used as a document caption in GRV.

For reports running in compatibility mode (reports having no associated .Arp report design document), the values can be set by calls to the corresponding API functions. See the Genero Studio >> Report Writer documentation topic "Reporting API Functions".

The "Keywords" property is currently not working for "xlsx" format.

## Adding Report Data (Data View)

This view specifies the structure of the data record in the .rdd file created for the report, or from an .xsd file that defines an XML data source. The .rdd file is extracted from your BDL file. See the Genero Studio >> Report Writer documentation topic "Steps to a Report" for more details.

although the data for the report originally may have come from several different data tables, the PRINT statement in your BDL REPORT program block outputs the data as part of a single record. See the Genero Studio >> Report Writer documentation topic "Writing the BDL Program" for more information.

- The **arrows** icon allows you to sort the data items alphabetically.
- Values in the **sample data column** are displayed when you preview a report. Double-click a value to edit or replace it.
- The **Filter Fields By Name** edit box at the bottom of the Data View allows you to specify that only the fields containing the name that you enter in the box are displayed in the data items.

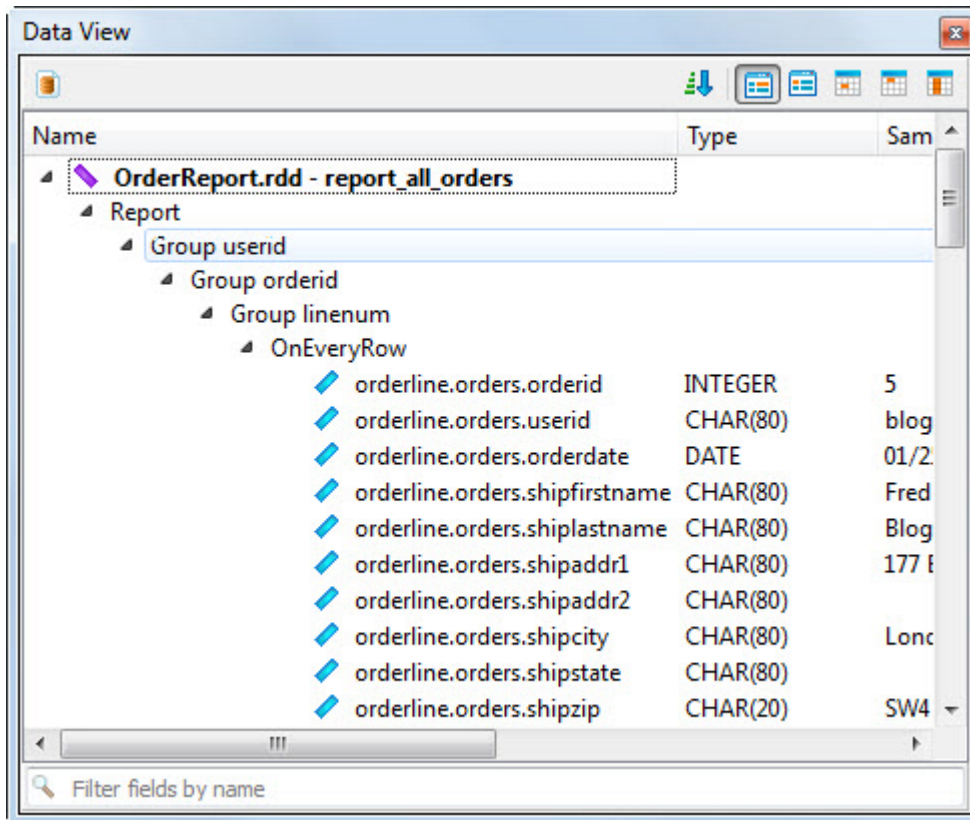


Figure 9-20. Data View

Click the **Open Data Report File** icon at the top of the Data View to specify the .rdd file or xsd file that is to be used to populate the Data View.

**Adding Data Values and Captions:** Before you place a data object onto the report design window, click one of the icons on the integrated toolbar to specify whether you wish to drop the item or its title, and whether the object is part of a table:

Dropping the object as a simple report field:

- A form field **value object**
- A form field **title (caption) object**

Dropping the object aligned as part of a table (the space allocated for the column will be the larger of the space required for the data or the title, helping to align the title and data in the columns of a table):

- A **table column value object**
- A **table column title (caption) object**
- A **table column value object for a column without a title** - if you are not going to have a column header, the space allocated for the column is set to the maximum required by the value only

The data objects are automatically contained in a Word Box if the data type is defined as less than 30 CHAR, and in Wordwrap Boxes if the data type is defined as larger than 30 CHAR. If the data type is Numeric, the data object is contained in a Decimal Format Box.

The text property of the Word Box or Word Wrap Box, or the value property of the Decimal Format Box, specify what will print in the report output. The value property of the Decimal Format Box can be calculated using an Expression. See Using RTL Expressions.

**Support for Arbitrary XML Data Sources:** Report writer can now produce reports from arbitrary xml input sources. The xml source is described by an xml schema. The Open schema file dialog proposes rdd and xsd file formats.

If an xsd file is selected, the designer interprets the file as follows:

- Any xml attribute is considered a variable
- Xml elements that do not repeat are also considered variables
- Xml elements that repeat are considered repetition and a trigger is created

Optional variables (defined in XSD through the attribute minOccurs="0") are currently not supported.

### Organizing the Report Structure (the Report Structure View)

How do you make sure that the various elements print in the correct place in the report? Use containers - by combining horizontal boxes, stripes, word boxes, etc., in the correct order in the parent container, you control the order in which the elements in that parent container print out. Then you can move the containers around in the Structure View to make sure they print out in the correct spot.

#### The Tree Structure:

- **object nodes** - the containers and objects that are to be printed on the report output
- **trigger nodes** - the triggers (event handlers) that specify when the object nodes are to be printed out

You can change the hierarchy of the objects and triggers in the Structure View by dragging and dropping them, to change where the data is printed:

- Drag a container or trigger and drop it on a different node; it will become a child of that node.
- Press the alt key and drag a container or trigger, dropping it on a different node; it will become the parent of that node.

**Triggers:** The data for your report is passed from your Genero BDL report program to the report one row at a time, sorted by the criteria in your BDL program's SQL statement. You may want to print some of the data or additional text for every row that is passed to the report; this is generally the report body. Some data or text should print when a change in a specific data item takes place (totals that print every time the customer\_id changes, for example.) The triggers in the structure of the report specify what should be printed when a change in data occurs. Arrange all the report elements for a single trigger in a parent container.

Trigger nodes are specified by the ORDER EXTERNAL statement in the REPORT program block in your BDL code, and indicate the data values by which the data is grouped. The final trigger node is ON EVERY ROW, specifying what is to be printed for each row of data passed to the report. The trigger nodes display in the Report Structure as red bullets:



## Genero BDL Code

## Report Structure

**ORDER EXTERNAL BY**

`orderline.orders.userid,`

`orderline.orders.orderid,`

`orderline.lineitem.linenum`



Figure 9-21. BDL Code and Report Structure

**Page Headers and Footers:** Arrange all the elements for a Page Header or Page Footer in a parent container that does not propagate, generally a Vertical Box (Layout Node). The container has a section property, allowing you to specify the header or footer you are defining (first Page Header, any Page Footer, etc. ) See "Print Headers and Footers" on page 9-63 in the Design HowTo page for additional information about creating Page Headers and Footers.

**Example Report Structure:**

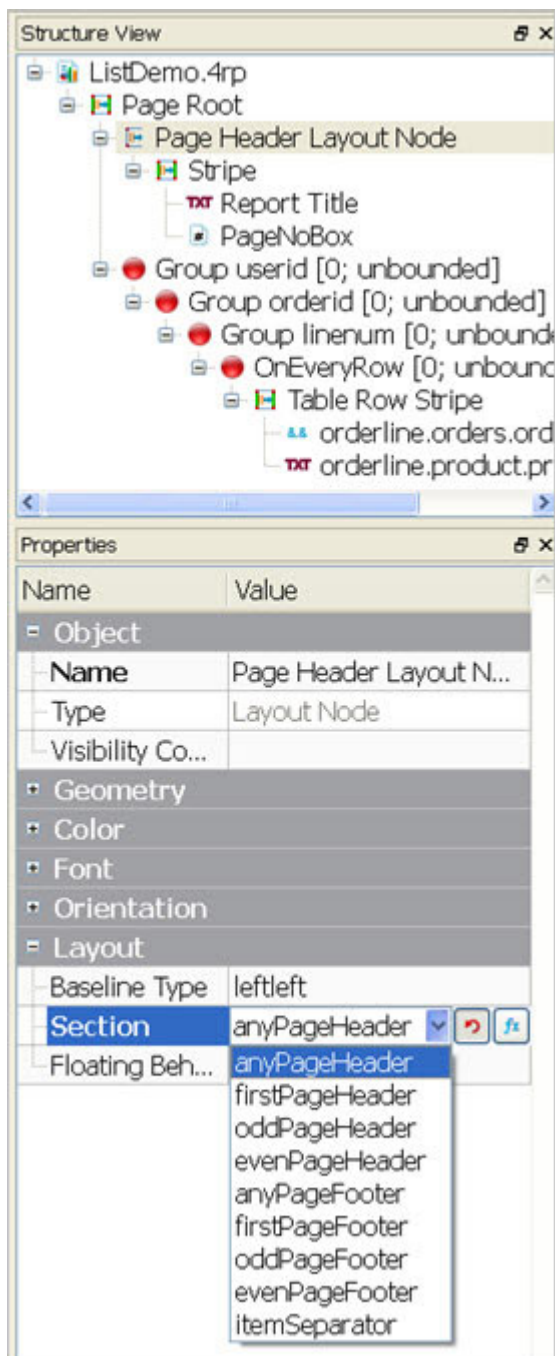


Figure 9-22. Structure View and Properties of a sample report

The Page ( Page Root container)

- Page Header: In the example shown, the Page Header is a Layout Node container that specifies what is to be printed as a Page Header. See Page Headers and Footers. The example Page Header contains:
  - a Stripe container
  - a Word Box named Report Title
  - a Page Number box
- The example data-related triggers Group userid, Group orderid, and Group linenum do not cause anything to print as they have no child containers.



- The example OnEveryRow trigger has a Stripe containing a Decimal Text Box and a Word Box. These are the data items that will be printed for every data row that was passed to the report.

**The Printed Page:** When a report page is printed, any Page Header (if defined) will print at the top of the page, followed by the content of the containers associated with trigger nodes, followed by the Page Footer (if defined) at the bottom of page, in accordance with the Page Size set for the report. This is the basic design of the report page, which will be repeated when the report is run for as many pages as are required, based on the data passed to the Report Writer.

## Using Page Numbers

The layout container PageNumber (PageNoBox) prints the page number of the physical page, and is frequently part of a Page Footer. Like BarCodeBox and ReferenceBox, the pageNoBox may contain a string that is dependent on the page breaks in the report for its value. See Calculating the Page Number String .

The following properties of the PageNumber container allow you to format the page number:

pageNoOffset, pageNoFormat, textAlignment,

**Calculating the Page Number String:** When the string is an expression that uses some of the functions, the maximum length required for a page number can be dependent on the page breaks, and vice versa. To make sure that there is enough space available to display any possible page number string, use the properties provided, such as X-size, text, textExpression, or CodeValueExpression, to indicate the maximum length. For example, setting the value of the text property to "000" specifies that the maximum length is 3 digits. The priority for selecting the property to be used is:

1. X-size
2. string text
3. expression evaluation.

If none of these properties are set, a default value is used to calculate the maximum length of the string.

**Using "Page N of M":** For PageNumber Boxes, you can use the textExpression property to calculate a page number string, for example to print "Page N of M". If you set the textExpression property, the values of the **pageName**, **pageNoOffset**, and **pageNoFormat** properties are irrelevant; however, the text property is still used to indicate the maximum length. Beginning with version 2.4x, pages using "Page N of M" no longer causes latency; they will be streamed,

This same functionality is available for BarCode Boxes, using the property codeValueExpression.

The following functions can be used to format and access specific page numbers and totals.

- Class String: format(Numeric *number*, Enum *format*) - formats the number as specified. The value for the format parameter can be ARABIC, LOWERROMAN or UPPERROMAN.
- Class Numeric: getPhysicalPageNumber() - gets the current page number of the physical page.

- Class Numeric: `getTotalNumberOfPhysicalPages()` - gets the total number of physical pages.
- Class Numeric: `getPageNumber(String pageName)`- gets the page number of the specified page
- Class Numeric: `getTotalNumberOfPages(String pageName)` - gets the total number of pages for the specified page.

### Effect on latency

Using the function `getTotalNumberOfPhysicalPages` causes the output of the report to be delayed until the entire input has been read. The use of the function `getTotalNumberOfPages` delays the output until all material for the node named by "`nodeName`" has been produced. Using any of the other functions has no impact on the streaming (causes no latency).

### Examples:

The following expression computes the string "Page n of m" for the physical pages. The equivalent of the "Offset" property can be achieved by doing arithmetic with the results from the page number functions. In this case the numbering will start at page 11 since the example formula adds 10 to the value returned from the function `getPhysicalPageNumber`.

```
"Page "+format(getPhysicalPageNumber(),ARABIC)+" of
"+format(getTotalNumberOfPhysicalPages(),ARABIC)
```

The following expression computes the string "Page n of m" for logical pages, providing page numbers for each order within a batch of several orders.

```
"Page "+format(getPageNumber("pageRoot"),ARABIC)+" of
"+format(getTotalNumberOfPages("pageRoot"),ARABIC)
```

### Report Design Document Metadata

The title, author, creator, subject, and keyword properties of the document root allow you to add metadata for a report. Select the document root in the Structure View and enter your values for the properties:

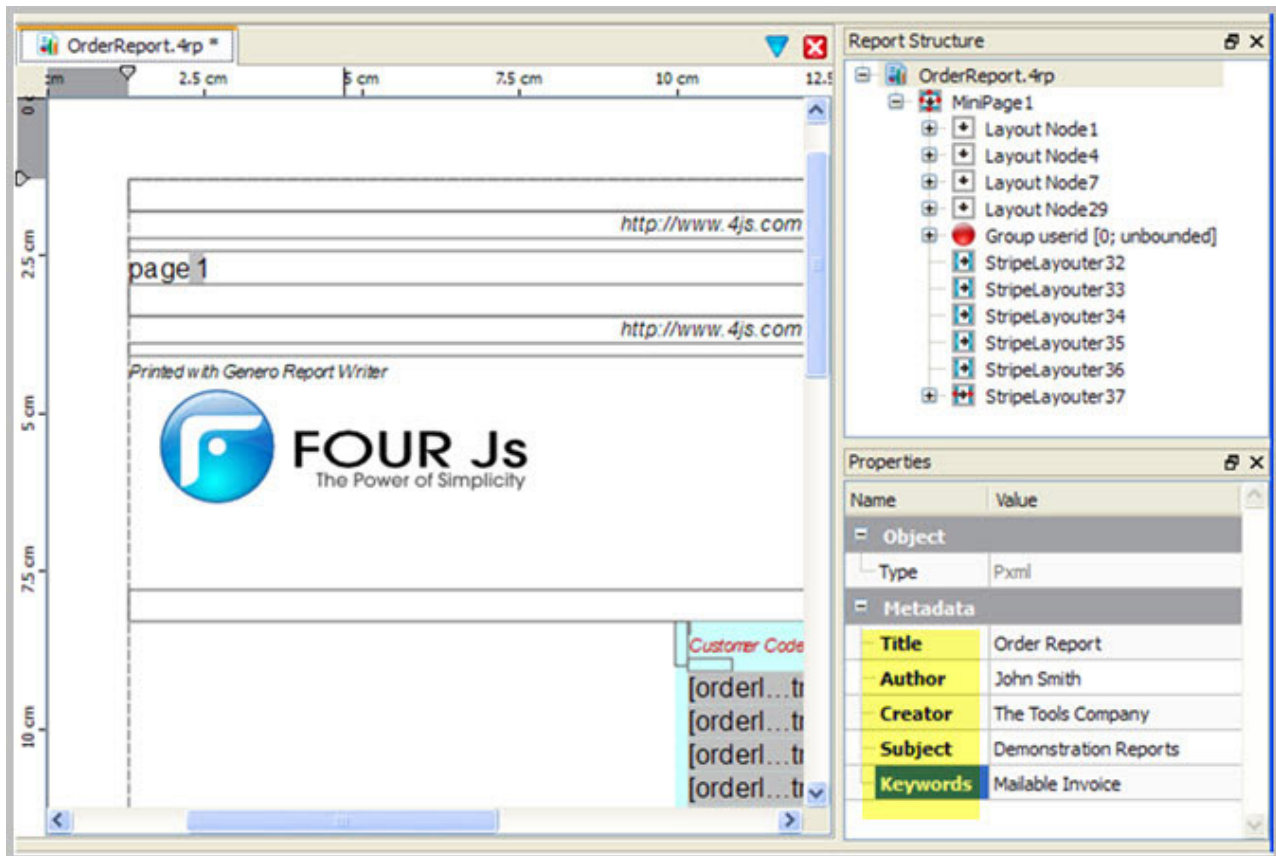


Figure 9-23. Report Design metadata

The meta data is inserted into the final document (e.g. PDF, SVG) if the format supports meta data. In the case of SVG, the **title** property is used as a document caption in GRV.

If your report will run in compatibility mode (having no .4rp design document), the values can be set by calls to the corresponding Report API functions: `fgl_report_setTitle()`, `fgl_report_setAuthor()`, `fgl_report_setSubject()` and `fgl_report_setKeywords()`. See the Genero Studio >> Report Writer documentation topic "Report API Functions".

## Configuring the Output

The same report can be output in different formats and different page sizes, and to different output devices.

In **Preferences** you can change the default paper settings for all reports: Tools>>Preferences, Report Writer, Paper Settings. See the Genero Studio >> Report Writer documentation topic "Report Writer Preferences".

The **File>>Report Properties** main menu option allows you to change the default report options for the currently open report design document.

- **Paper Settings** - select the page size and other paper settings for a report

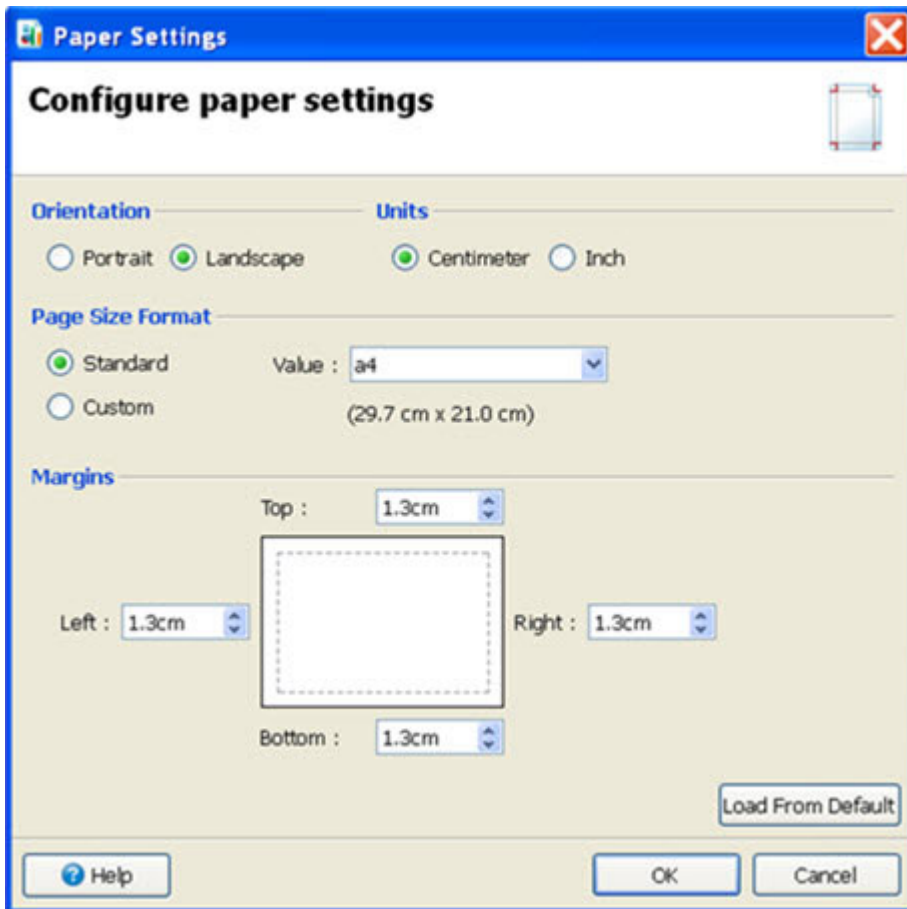


Figure 9-24. Paper Settings dialog

- **Orientation** - portrait or landscape
- **Units** - centimeter or inch
- **Page Size Format** - select from a list of common formats, or enter a custom height and width
- **Margins** - set **left**, **right**, **top**, and **bottom** margins

The **Load from Default button** restores the default values for paper settings as set in the Genero Report Writer Preferences. See the Genero Studio >> Report Writer documentation topic "Report Writer Preferences".

- **Output Configuration** - select the output format and other options

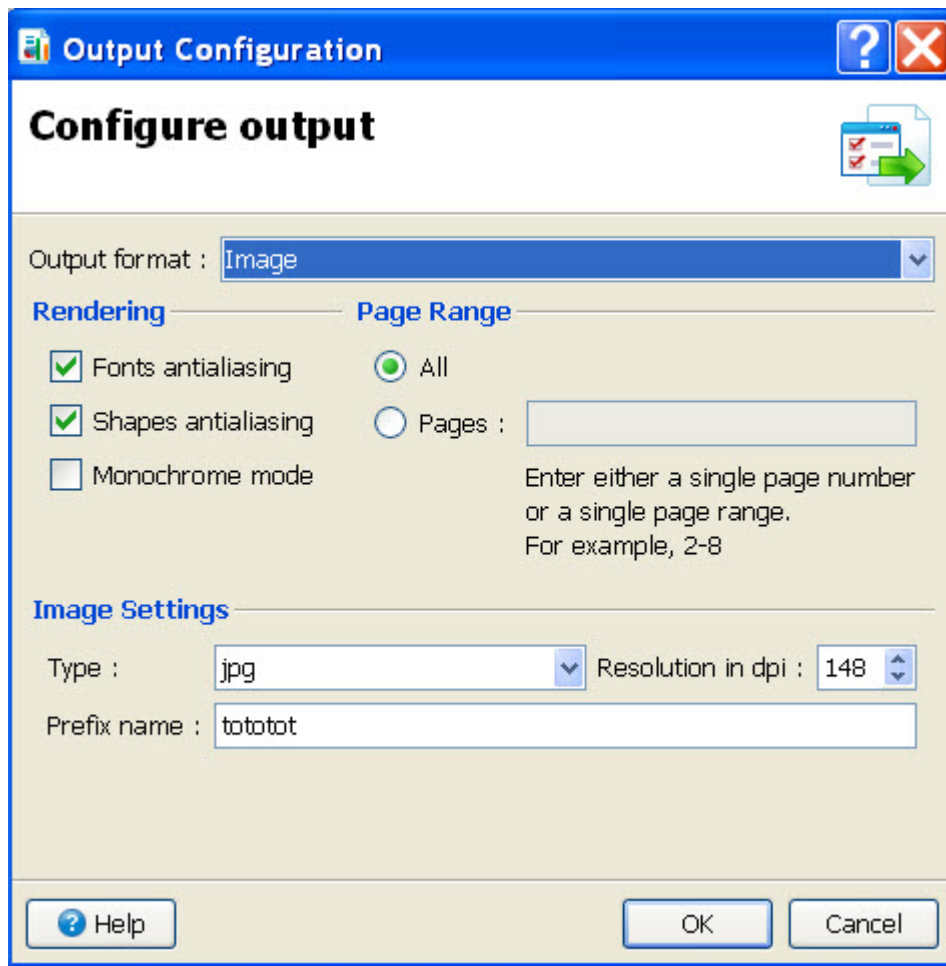


Figure 9-25. Output Configuration dialog

Options for **Output format**:

- **SVG** (scalable vector graphics - can be displayed using the Report Viewer feature of GDC)
  - **Rendering** - Select options to minimize the aliasing distortion
- **PDF** (Acrobat PDF format, can be displayed using PDF viewer)
  - **Rendering** - Select options to minimize the aliasing distortion or set monochrome mode
  - **Page Range** - output all pages or enter a range
- **Image** (creates an image, such as .jpg. You can select the image type.)
  - **Rendering** - Select options to minimize the aliasing distortion or set monochrome mode
  - **Page Range** - output all pages or enter a range
  - **Image settings** - Select image **Type**, **Resolution**, **prefix for the image filename**

**Note:** The Rendering options for font antialiasing (SVG or PDF documents) will only take effect if the fidelity property of report text elements is set to "True".

Functions from the Reporting API can be used in your BDL report program (.4gl) to further override the default options at runtime. For example, the function

fgl\_report\_selectdevice provides additional output formats. See the Genero Studio >> Report Writer documentation topic "Report API Functions".

## Design How-To

### Topics:

- General Design
  - Align Numbers (format)
  - Center Elements
  - Configure the Page size
  - Force a Page Break
- Print Headers and Footers
- Print Group Totals and Report Totals
- Print Totals at the beginning of the report
- Have Different first and last Pages
- Print an Invoice Page Number instead of the physical page number
- Print a layout-dependent Reference (InfoNodes)
- "Design Documents for Preprinted Forms" on page 9-67
- Design Labels
- Modify an object's Borders, Margins, and Padding
  - Size Expressions for Bordered Boxes
- Using Hyperlinks in a Report
- Some Tips for Legacy Report Designers

See Also: Report Designer

### General Design

The basics of report design is discussed in Designing a Report. These topics describe some common features of reports and how to do them.

**Align and Format Numbers:** Use the ToolBox object Decimal Format Box as the report element for numbers. It supports different locales and kinds of numbers, including integers, fixed-point numbers, and currency amounts (\$123).

The text property specifies the value for the number to be printed. The value of this property may also be edited directly in the report design document by double-clicking the Decimal Format Box. The input cursor will be placed in the document, and the layout of the document is updated on each keystroke.

The format property specifies how the number will print out. The default value for this property is "---,---,---&.&&". You can change this string, using the specified symbols. The - (minus) symbol represent digits (if the number is negative, it will print with a leading - ), and the period represents the decimal point. The & symbol fills with zeros any position that would otherwise be blank. If the actual number displayed requires fewer characters than the format string specifies, numbers are right-aligned and padded on the left with blanks.

*Table 9-2. Formatting examples*

| Value of text property | Appearance in report |
|------------------------|----------------------|
| 123456.1               | 123,456.10           |
| 15.24                  | 15.24                |

Table 9-2. Formatting examples (continued)

| Value of text property | Appearance in report |
|------------------------|----------------------|
| -1600                  | -1,600.00            |

See the format property for a complete list of symbols.

**Center Elements:** To center an element in its parent container you can set its properties as described in Table 9-3

Table 9-3. Centering elements

| Property | Value |
|----------|-------|
| x        | max/2 |
| y        | max/2 |
| anchorx  | 0.5   |
| anchory  | 0.5   |

An **x** and **y** value of **max/2** sets the x and y coordinates of the element to the maximum of its parent container divided by 2. An **anchorx** and **anchory** value of **0.5** sets the attachment point to the center of the element.

**Configure the Page Size:** You can include the page size and margin settings in the Report Design Document using the menu options File>>Report Properties, Paper Settings. This will override the default settings. In addition, the Reporting API contains functions that allow you to configure the page size programmatically.

**Force a Page Break:** To force a page break, you can insert a container in the page, using the *rest* variable in the X-size or Y-size property of the container; this makes the container consume the remainder of its parent container. The property to be set is determined by the default layout direction of the container. For example, to force a page break after a MiniPage or Stripe container, set the height ( X-size) to **rest**. To force a page break after a Layout Node, set the Y-size to **rest**.

Table 9-4. Forcing a Page Break

| Property | Value |
|----------|-------|
| Y-size   | rest  |

## Print Headers and Footers

A MiniPage or PageRoot in your Report Design document defines how the content is to be laid-out on the report page. Headers and Footers can be defined for the report pages, which can depend on the relative position of the page in the overall report.

You can use any Simple Container or Drawable as a container for a header or footer. Generally, a Vertical Box (LayoutNode) is used.

Add the header or footer container to a MiniPage or PageRoot. To identify the containers as headers or footers, and to specify where on the report page a container should print, set the container's section property to one of the following ports:

- **firstPageHeader** - Page Header, to print on the first page only; if this section is defined, subsequent Page Headers begin printing on the second page.



- **oddPageHeader, evenPageHeader** - Page Header, to print on odd or even pages; has precedence over anyPageHeader.
- **anyPageHeader** - Page Header is for every page, unless separate Odd and Even Page Headers are defined.
- **firstPageFooter** - Page Footer, to print on the first page only; if this section is defined, subsequent page footers begin printing on the second page.
- **oddPageFooter, evenPageFooter** - Page Footer, to print on odd or even pages; has precedence over anyPageFooter.
- **anyPageFooter** - Page Footer, to print for every page, unless separate Odd and Even Page Footers are defined.

For example, a Vertical Box with the section property set to the firstPageHeader section will print as the header on the first page of the report. In the parent Container, you cannot have multiple header or footer containers set to the same section.

If you use a Vertical Box for the header or footer container, set the Layout Node's X-size property to *max*, and its Y-Size property to *min*. Within the container you can build up the header or footer using various containers and report elements. The Stripe container is useful for report elements that are to be laid out left to right across the page. Use Stripes, WordBoxes, etc. as needed, to arrange the contents of the header or footer within the container in the order in which it should be printed.

The properties HidePageHeaderOnLastPage and HidePageFooterOnLastPage provide flexibility in the printout.

**Important:** It is an error if any element having the section set is preceded in the same sibling list by one that doesn't. In other words, any sections for the MinPage need to be specified first. Verify in the Structure View that the report structure is correct.

## Print Group Totals and Report Totals

In your BDL report application, define variables for the totals, calculate the values, and output them to the report engine.

- in the REPORT program block:
  - Define variables
 

```
DEFINE
 data store_order_data,
 --Add variables for totals
 store_total, order_total, item_total, report_total DECIMAL(10,2)
```
  - Identify the data columns by which the data is grouped in the **ORDER EXTERNAL command**.
 

```
ORDER EXTERNAL BY
 data.orders.store_num,
 data.orders.order_num
```
  - Calculate the group and report totals, and output them, in the **FORMAT** section.
 

```
FORMAT

 --Add FIRST PAGE HEADER, BEFORE GROUP OFs
 FIRST PAGE HEADER
 LET report_total = 0

 BEFORE GROUP OF data.orders.store_num
 LET store_total = 0
```



```

BEFORE GROUP OF data.orders.order_num
 LET order_total = 0

ON EVERY ROW
 --Add statements to calculate totals
 LET item_total = data.items.price*data.items.quantity
 LET order_total = order_total + item_total
 LET store_total = store_total + item_total
 LET report_total = report_total + item_total
 PRINT data.*, order_total, store_total, report_total

```

- In the Report Design window, define Stripes in the design document, one for each total. Use Decimal Format Boxes to hold the values.
- In the Structure View, drag each Stripe and drop it onto the trigger node for the corresponding group trigger. This will position the Stripe as a child of the trigger node. It is important that the total stripes are the last child element of a report trigger node.

### Example Report Structure for Totals

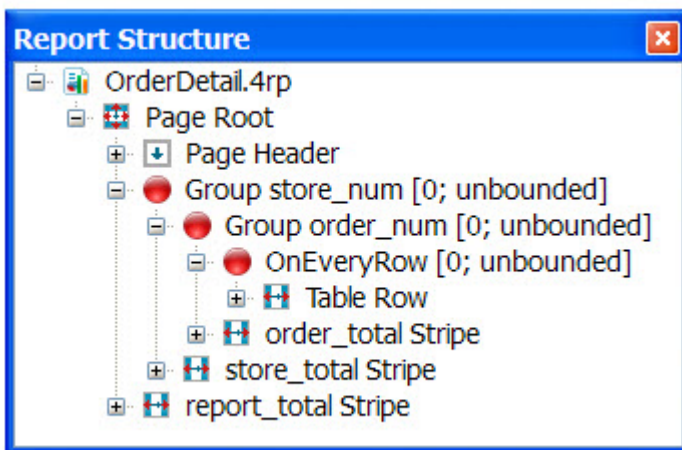


Figure 9-26. Example Report Structure

In this example, the **order\_total** Stripe is the last child of the Group **order\_num**, the **store\_total** Stripe is the last child of the Group **store\_num**. The **report\_total Stripe** was dropped onto the **Page Root**, the main page for the report. This will position the Stripe as a child of the Page Root, and will place it at the very bottom of the child list for that trigger.

On each change of data, the specified data for every corresponding row will print out, and the appropriate Stripe will print out after each change of Group. The report\_total Stripe will be the last thing to print out on the report. The values of any data objects in the Stripe will be taken from the immediately preceding row of data (the last report line of the container for OnEveryRow trigger.)

### Print totals at the beginning of a report

it is now possible to print aggregate values before the first detail row is printed. As a result, totals can be printed at the beginning for a report, for example.

While this feature adds flexibility to the designs, the output has to be delayed until the input has been processed to the point where the variable value is shipped. In the case of a *grand total* which is shipped at the end of the report, the entire input would be consumed before the document fragment containing the total would be output. If, on the other hand, the lookahead distance is small (a few hundred rows such as printing the total of an order in advance), the delay will hardly be noticeable.

## Print a Layout-dependent reference (InfoNodes)

InfoNodes allow you to print a value on the report that depends on the paged stream resulting from the report layout. For example, a value for "total from previous page" can vary depending on how the page options for a report are set. In order to have a report layout that will work with various page sizes, you can use an InfoNode and a Reference Box.

The following example illustrates how to print the total price (**overalltotal**) from a previous page:

- First, the **REPORT** block of the BDL file associated with the report must calculate the desired value and output it to the report. The following example is from the OrderReport.4gl file in the demo sample programs:

```
ON EVERY ROW
 LET lineitemprice = orderline.lineitem.unitprice * orderline.lineitem.quantity
 LET overalltotal = overalltotal + lineitemprice
 LET ordertotal = ordertotal + lineitemprice
 PRINT orderline.*, lineitemprice, overalltotal, ordertotal
```

The variable **overalltotal** contains the running total price of the lineitems on the report.

- You will use the following objects from the Toolbox in your report design:
  - InfoNode - place this object in the container for the ON EVERY ROW trigger of your Structure View. This will create an invisible column in your report line containing the value of the InfoNode.

The Value property of the InfoNode must be a String. You can use the fglValue member of the FGLNumericVariable class to convert **overalltotal**:

```
overalltotal.fglValue
```

This will format the value of **overalltotal** as a String based on the default format set in the Genero DVM. Or, you can use the format method of the Numeric class to convert to a string and also specify the format, as in the following example:

```
overalltotal.format("-",---,---,--&.&&")
```

- Reference Box - place this object in the Page Header at the top of the report structure.
  - For the **InfoNode name** property, enter the name of the InfoNode that you created.
  - For the **text** property, enter a string that will only be used to determine the maximum length of the value in the InfoNode, since the value will not be known at the time the ReferenceBox is positioned. Examples: Enter "000,000.00" as the maximum length for a value that is from a numeric data type, or "MMMM" as the maximum length for a value that is from a CHAR(4) data type.
- WordBox - optionally use this object to add some text next to the Reference Box.

A Reference Box points to the immediately previous occurrence of the InfoNode value in the paged stream. Because you placed the Reference Box in a Page Header, it will point to the last occurrence of the **overalltotal** value on the previous page.

## Have Different First and Last Pages

How do you have a different first or last page in a report?

- Add a separate MiniPage container for each page variation (first-page, main-report-page, last-page, for example), as children of the Page Root container.
- Add the report elements that are specific to each container.

In the Structure View the MiniPage containers should be listed in the order in which you want them to appear in the report. For example:

- Page Root
  - first-page - this is a "before" page, to print before the main content
  - main-report-page - this would contain all the triggers and containers that make up the body of the report
  - last-page - this is an "after" page, to print after the main content

## Print an Invoice Page Number instead of the Physical Page Number

- Use the name property of the MiniPage container of the invoice to assign a name to the page.
- Add a Page Number report object to the page header of this MiniPage container.
- In the **Page Number** section of the properties for the Page Number object, set the **name** property to the name of the MiniPage Container.

The page number of the MiniPage will be printed on the report.

## Design Documents for Preprinted Forms

A simple drag and drop of the report elements will invoke the Positioning method, which allows you to place an element in the desired location on the form. To make this easier, add an Image box to the report, containing an image of the form to serve as a background.

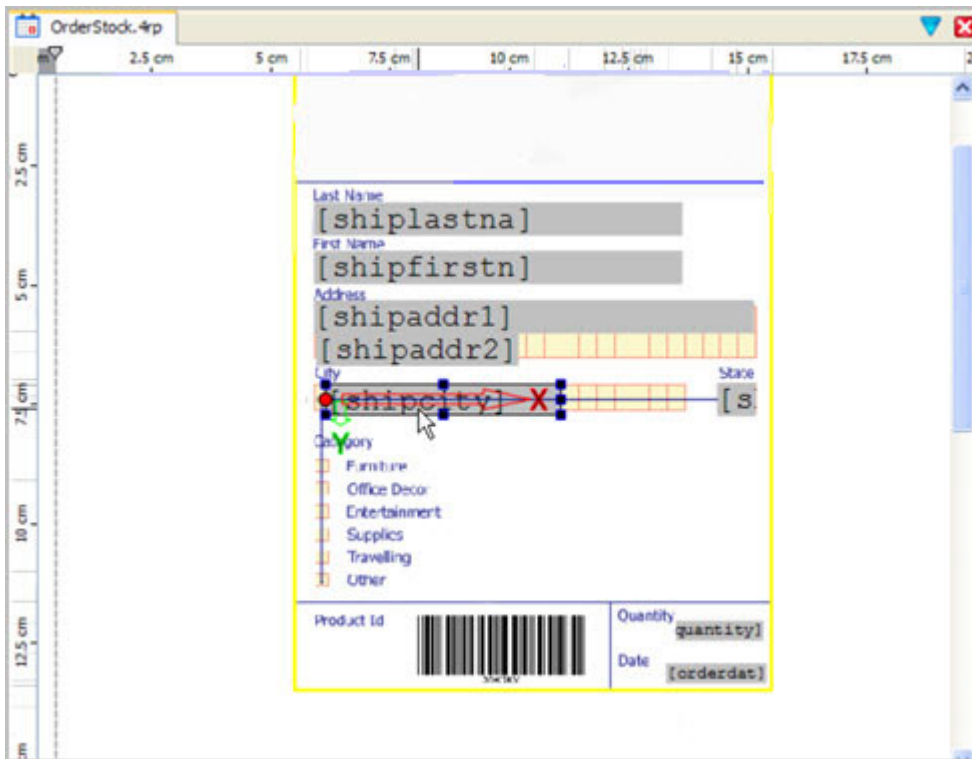


Figure 9-27. Report with Image

The report design window contains a global grid; as you drag a report element onto the design window, black grid lines help you align it with the other elements on the form. The red attachment point indicates the mapping of the element to the global grid.

Once you have dropped an report element, you can refine its location by dragging, or you can select an element and use the keyboard arrow keys:

- Pressing an arrow key moves the element incrementally in the corresponding direction
- Ctrl-arrow key moves the element along the global grid.

Right-click an element to display a menu of additional options.

## Design Labels

In your report design document, set the page size format (**File>>Report Properties>>Paper Settings**) to custom, and make the page the size of one label.

Draw just one label inside the small page you've created, by dragging and dropping the desired data fields from the Data View.

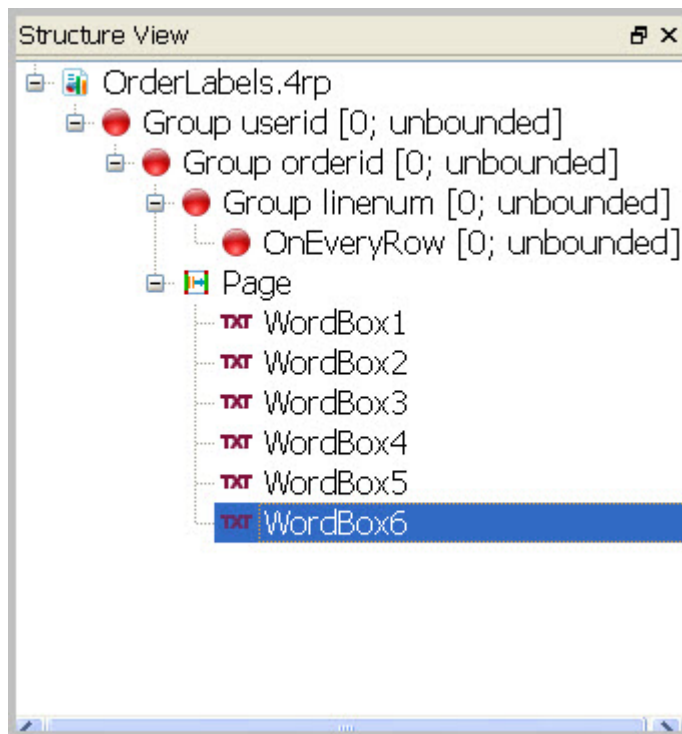


Figure 9-28. Report Structure

Next, in your BDL file, you'll use functions from the Reporting API to describe the paper and specify how many labels should print on the physical page.

Refer to the topic "Configuring for Labels" in the Genero Studio >> Report Writer documentation for more information.

### **Modify an Object's Borders, Margins, or Padding**

Any box object on a report design document can have margins, borders, and padding:

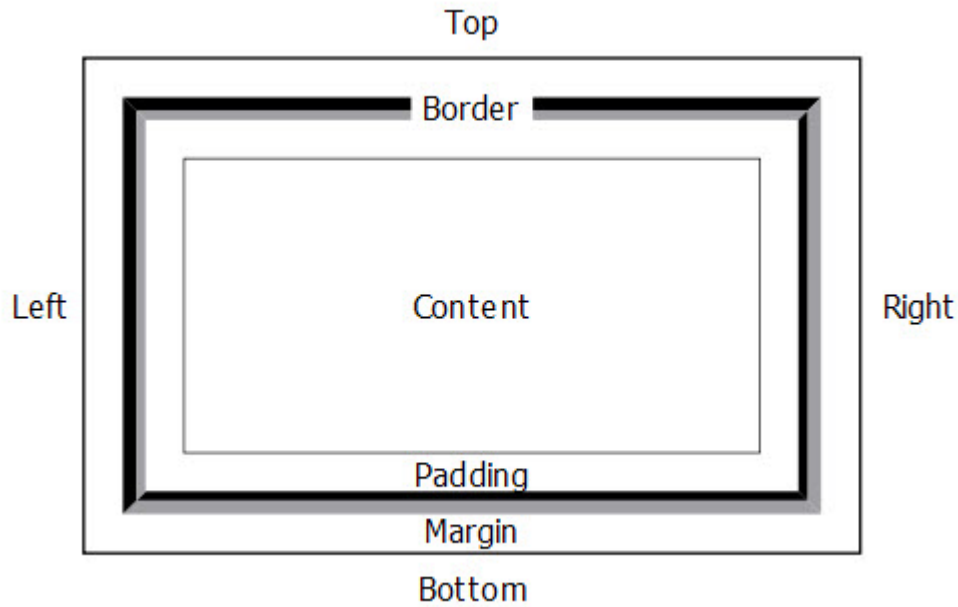


Figure 9-29. Border, Padding, and Margin

Set an object's specific properties in the Properties View to change:

- the width of a margin, border, or padding - `marginWidth`, `marginRightWidth`, `marginBottomWidth`, `marginLeftWidth`, `marginTopWidth`, `borderWidth`, `borderRightWidth`, `borderBottomWidth`, `borderLeftWidth`, `borderTopWidth`, `paddingWidth`, `paddingRightWidth`, `paddingBottomWidth`, `paddingLeftWidth`, `paddingTopWidth`
- the style of a border: solid, dashed, double, dotted, groove, ridge, inset, outset - `borderStyle`, `borderRightStyle`, `borderBottomStyle`, `borderLeftStyle`, `borderTopStyle`
- the color of a border - `borderColor`, `borderRightColor`, `borderBottomColor`, `borderLeftColor`, `borderTopColor`
- whether the box will have rounded corners (limited to the border styles solid, dashed, and double) - `roundedCorners`

Borders are **drawn outside the box** and will increase the actual size of the box beyond the value specified in `x-Size` and `y-Size`.

When a bordered item is positioned it behaves like a regular element, so that the attachment point appears at the specified position.

### Illustrations

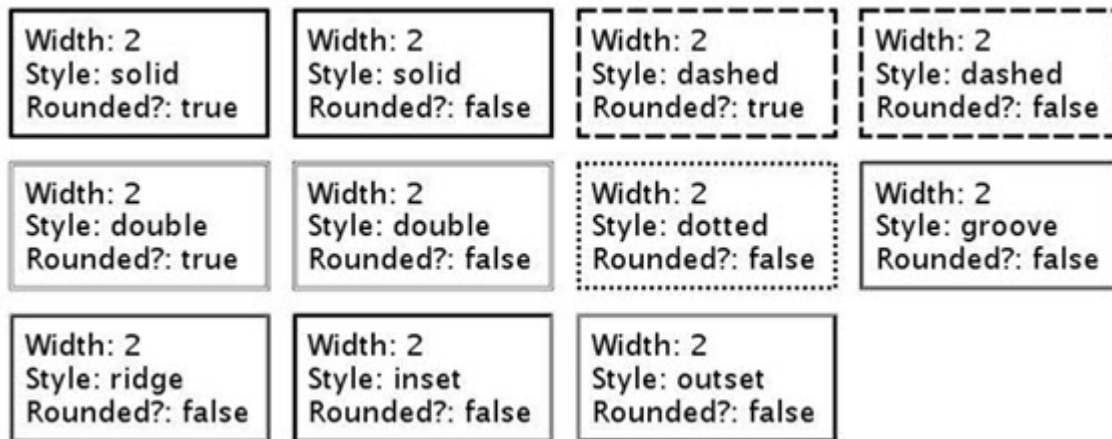


Figure 9-30. Examples of borders

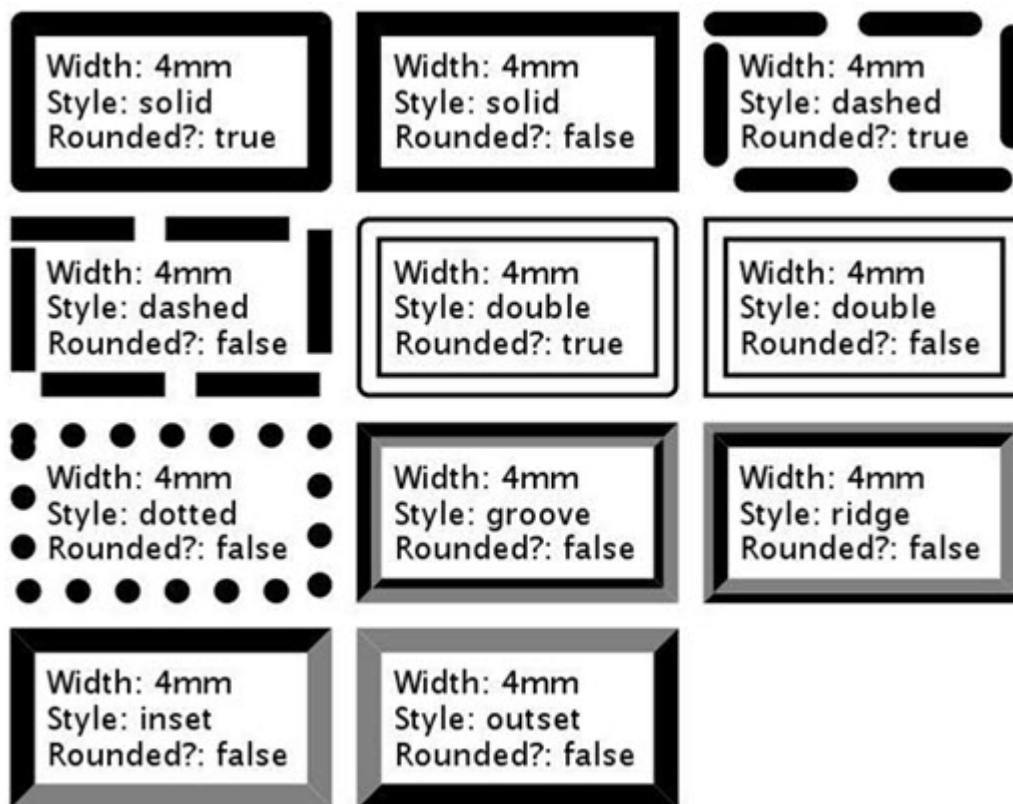


Figure 9-31. Examples of borders

**Size expressions for bordered boxes:** The x-Size and y-Size properties specify the inner size of the box; if we specify a box to be 3cm wide and have a 1mm thick border on all sides, for example, the box's outer bounds will appear to be 3.2cm wide. This conforms to the CSS specification.

You can define the outer bounds of a box instead:



- Determine the x-Size and y-Size values by subtracting the width of the borders from the desired height and width. For example, if you want a box to be 3cm wide on the outside while having 1mm borders on all sides, calculate the width to be 3cm-2mm=2.8cm wide.
- If you want a box to have the same size as its parent, however, set both the x-Size and y-Size properties to the value **max**. You do not have to subtract the borders, since the system automatically adjusts the value of **max** in cases where the box has borders. For example, if the box has a 1mm border and is contained in a box that is 3cm high and wide, the outer bounds of the contained box will also be 3cm.

**Note:** Do not use expressions that contain **max** as only one of its components, such as  $\text{max}/2$ , to specify the height and width of bordered boxes. Doing so can have unexpected results.

**Size expressions that contain the variable max with other components:** As explained in “Size expressions for bordered boxes” on page 9-71, the value of **max** is automatically adjusted by the border values. This causes unexpected results in expressions such as  $\text{max}-2\text{cm}$  or  $\text{max}/2$ , for example, where **max** is only a component of a more complex expression. This will be fixed in the next release. For the time being, modify such expressions as follows:

1. Take the original formula and replace any occurrence of "max" with "(max+borders+padding+margin)" where "borders", "padding" and "margin" denote the width values for each on both sides of the box.
2. Take the resulting formula (which we'll call "f") from step 1, and create the final formula as "f-borders-padding-margin".

### Example

LAYOUTNODE: x-Size="max/2", leftBorderWidth="2mm", leftMargin="1mm", rightBorderWidth="1.5mm", rightPadding="3mm"

Changing the expression for x-Size:

- Before Step 1: x-Size="max/2"
- After Step 1: x-Size="(max+2mm+1mm+1.5mm+3mm)/2"; this is "f" in the explanation.
- After Step 2: x-Size="(max+2mm+1mm+1.5mm+3mm)/2-2mm-1mm-1.5mm-3mm", which can be consolidated to "(max-7.5mm)/2-7.5mm".

The final property values for the box are:

LAYOUTNODE: x-Size="(max-7.5mm)/2-7.5mm", leftBorderWidth="2mm", leftMargin="1mm", rightBorderWidth="1.5mm", rightPadding="3mm"

### Using Hyperlinks in a Report

Two new properties - id and href - can be specified for text and images ( wordBox, wordWrapBox, decimalFormatBox, pageNoBox, referenceBox, imageBox, htmlBox). The runtime system can insert hyperlinks and ids in any output file that is in SVG, PDF, HTML or XLS format. The Report Viewer has been extended to support hyperlinks.

**id** - can be used to create an anchor in the document. Nodes can be identified with a unique id and then used as the target of a hyperlink.



**href** - can be used to define a hyperlink pointing to any resource on the Internet, local disk, or any anchor inside the document. The href should be defined using the URI syntax. For example:

```
http://www.google.com
mailto:santa.clauss@northpole.com
#ref
```

The runtime inserts hyperlinks and ids in the output file for SVG, PDF, HTML and XLS. The report viewer has been extended to support hyperlinks.

**Note:** XLSX format is not working for the moment. This should be fixed when the POI library version 3.8 is published and integrated in GRE.

## Some Tips for Legacy Report Designers

If you've designed legacy Genero reports, the following may answer some of your questions regarding the correlation between Report Designer and traditional 4GL commands in reports:

*Table 9-5. Legacy Report to Genero Report Writer*

| Legacy Report command              | Using Genero Report Writer                                                                                                                                                                                                                           |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SKIP TO TOP OF PAGE                | In the report design document, drop a container that will consume the remainder of the page. See Forcing a page break.                                                                                                                               |
| BEFORE GROUP OF,<br>AFTER GROUP OF | There is a GROUP trigger for data control breaks in the report structure. The position and contents of the child containers of the trigger determine what is printed out and when. See Group and Report totals.                                      |
| ON EVERY ROW                       | In the ON EVERY ROW statement of the BDL file, the PRINT statement just sends the data items to the report engine. The report design document specifies what is to be printed out for every row of data passed to the report.                        |
| SPACES, format strings             | All of the formatting for the report line is done in the report design document. These keywords are no longer used in a PRINT statement in the BDL file.                                                                                             |
| PRINT                              | In the report design document, the Data View tab displays the list of data items in the order in which they are specified in the PRINT statement.                                                                                                    |
| ON LAST ROW                        | Drop a container positioned as the last child of the page root. The contents of the container will print out after the last report row. See Report Total.                                                                                            |
| NEED <i>n</i> LINES                | Put all the report elements that need to be kept together in a Vertical Box Layout Node container. If there is not sufficient room on the page to print all of the elements in the container, the entire container will be printed on the next page. |

Table 9-5. Legacy Report to Genero Report Writer (continued)

| Legacy Report command    | Using Genero Report Writer                                                                                                                                                                                                                                     |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PAGE HEADER, PAGE FOOTER | Avoid using these control breaks, which are triggered by the line count of the BDL report, which does not correspond with the actual page breaks in the report output by Report Writer. Create page headers and footers in the report design document instead. |

## Working with Business Graphs

### Topics:

- Report Writer Business Graphs
  - Map Chart
  - Category Chart
  - XY Chart
- Creating a Graph
  - The Data
  - The Design
  - A Custom Key
- Output Charts as Tables
  - Table
  - Sorted Table
  - Aggregated Table

### Report Writer Business Graphs

All of the Business Graphs in Genero Report Writer map Numeric values, grouping the data as required for the desired result.

**Map Chart:** The MapChart layout object allows you to create a graph that has one set for values to be mapped, grouped together by a specific key.

**Draw As property:** This kind of graph can be drawn as a **Pie**, **Pie3D**, **Bar**, **Bar3D** or **Ring**, Table, Sorted Table, Aggregated Table. For example, the pie chart in Figure 9-32 on page 9-75 presents the total revenue (the value) for each customer (the key), based on the **OrderReport.rdd** file in the Report Writer sample programs.

**Draw Legend and Draw Labels property:** These properties have been added to customize the appearance of the plots. In particular the option to remove the legend is useful when more than several charts are drawn next to each other in a document; the option can be used to make the charts share a single legend by specifying the legend only on one of the charts.

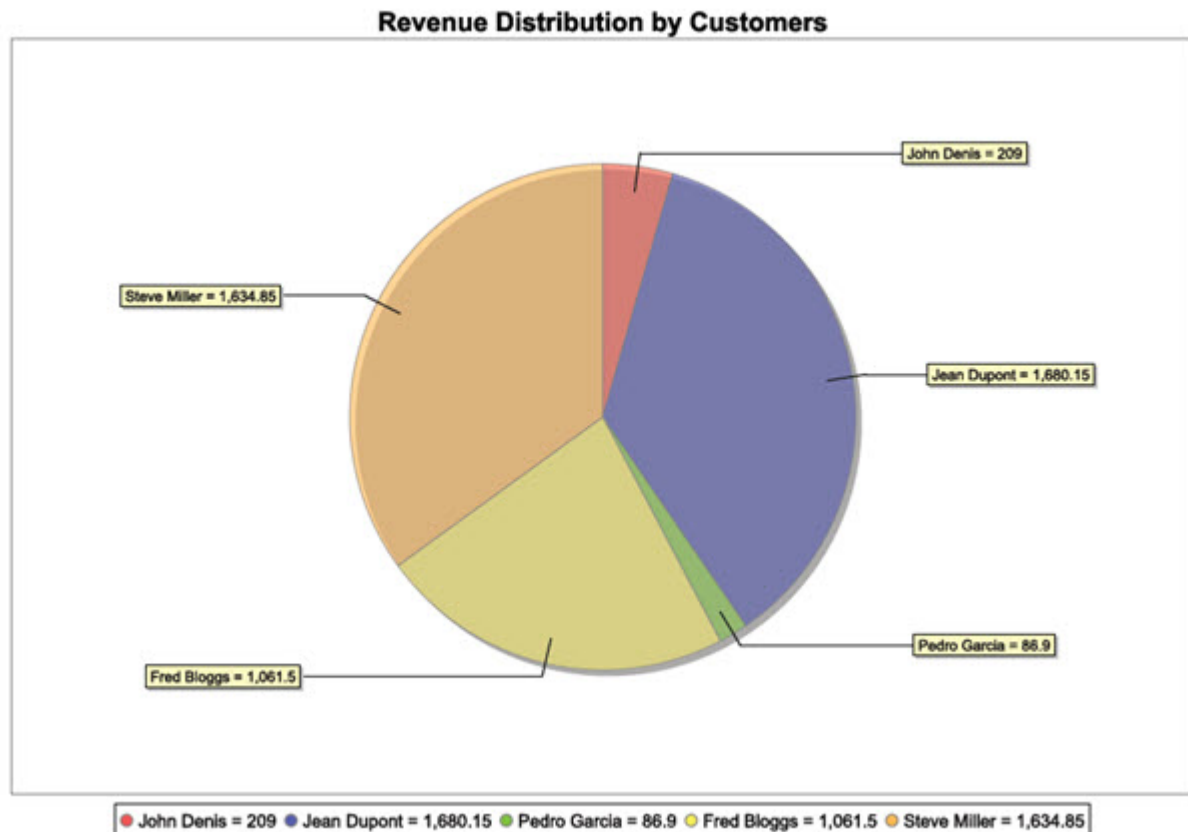


Figure 9-32. Pie Chart example

The **Map Chart Item** layout object specifies the key data item and the values data item in its properties:

- **Key** - the data is to be grouped by this property (customer name in the example chart); must be a String.
- **Value** - the chart will display the total of this property (total unitprice in the example chart) for each key (customer name); must be Numeric.
- **Name** - name of this report item in the Structure view; must be a String.
- **Color** - gives each slice a specific color. When a color is specified for a particular key in one chart, then the same color will be used for that key in other charts too, unless specified otherwise. If different colors are specified for the same key, the most recent value is used. If the same color is specified for a number of different keys, only one of these keys will be painted with the specified value; the other slices will be painted with interpolated values. Charts may use gradients, shading, or translucency with the colors specified.

**Category Chart:** This layout object allows you to create a graph that has two keys for each set of values.

**Draw As property:** This kind of graph can be drawn as a **Bar**, **Bar3D**, **Stacked Bar**, **Line**, **Line3D**, **Area**, **Stacked Area**, or **Waterfall**, Table, Sorted Table, Aggregated Table

In a waterfall graph, the value in the last category of the data set should be (redundantly) specified as the sum of the items in the preceding categories -

otherwise, the final bar in the chart will be incorrectly plotted. At the present time, the waterfall graph can only have one category.

The following Bar chart presents total revenue (the value) by the Customer Name and Product Category (the keys):

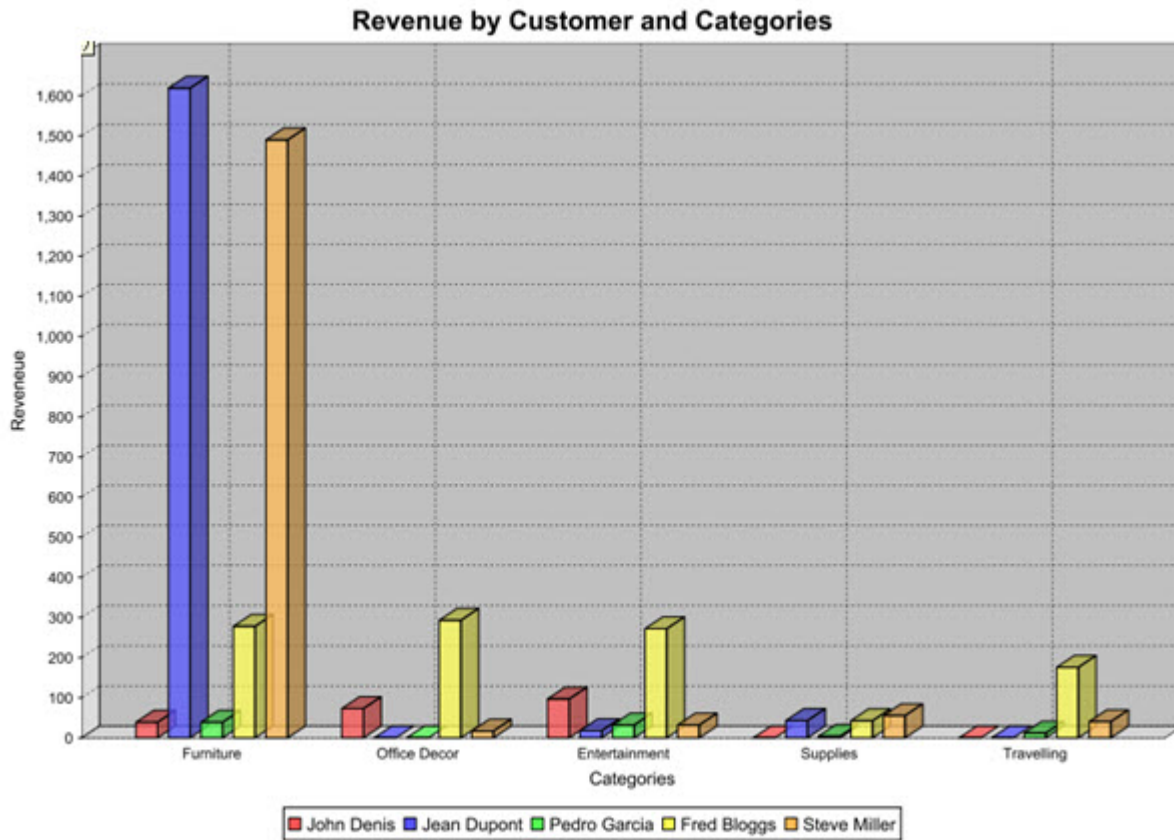


Figure 9-33. 3-D Bar Chart example: Revenue by Customer and Categories

The **Category Chart Item** layout object specifies the key data items and the values data item in its properties:

- **Category Key** - the categories for the data (the type of product purchased in the example chart); must be a String.
- **Key** - the data in each category is grouped by this property (the customer name); must be a String.
- **Value** - the chart will display the total of this property (the unitprice of order line items) for each category (product) divided into groups by Key (customer name); must be Numeric.
- **Name** - name of this report item in the Structure view; must be a String.

**XY Chart:** This element allows you to create a graph that maps a series that has two values, as an XY-plot. it is used most often for scientific data.

**Draw As property:** This type of graph can be drawn as a **Polar, Scatter, Area, Stacked Area, Line, Step, Step Area**, Table, Sorted Table, or **Time Series** (property.) The following graph presents the x and y values of three series of trigonometric functions.

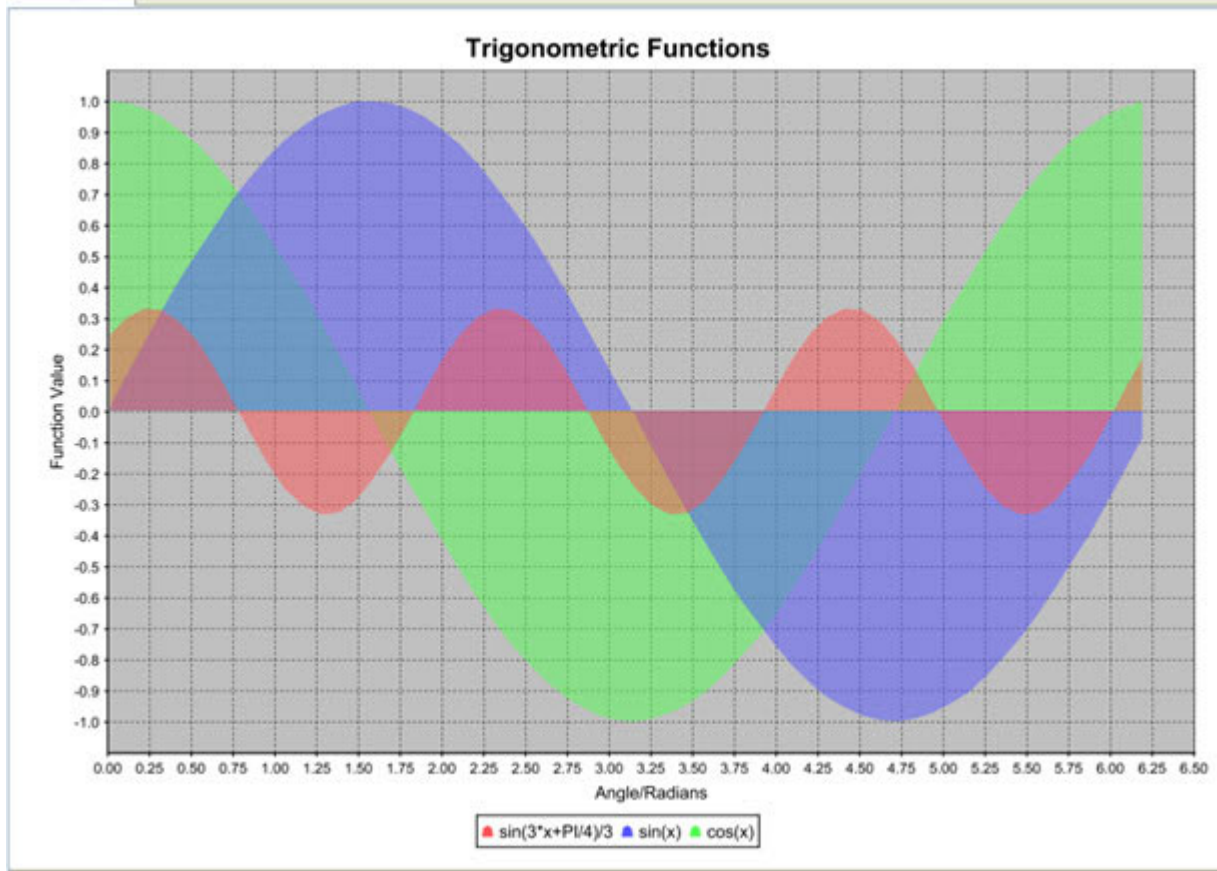


Figure 9-34. XY Chart example

The **XY Chart Item** element specifies the value data items in its properties:

- **Series Title** - the caption for the series of values being charted; there can be more than one series (three in Figure 9-34); must be a String.
- **X** - the values for the x axis; must be Numeric.
- **Y** - the values for the y axis; must be Numeric.
- **Name** - name of this report item in the Structure view; must be a String.

### Creating a Graph

A Business Graph uses the same process as any other type of report. See Steps to a Report for a list of the steps required to create a report.

**The Data:** As in other types of reports, the PRINT statement in the BDL file specifies the data structure of the information that is available for the graph:

```
ON EVERY ROW
 LET lineitemprice = orderline.lineitem.unitprice *
 orderline.lineitem.quantity
 LET overalltotal = overalltotal + lineitemprice
 LET ordertotal = ordertotal + lineitemprice
 PRINTX orderline.*, lineitemprice, overalltotal, ordertotal
```

See Creating the BDL File for additional information.

You extract the data structure as an .rdd file that you reference in the Data View page of the Report Designer:

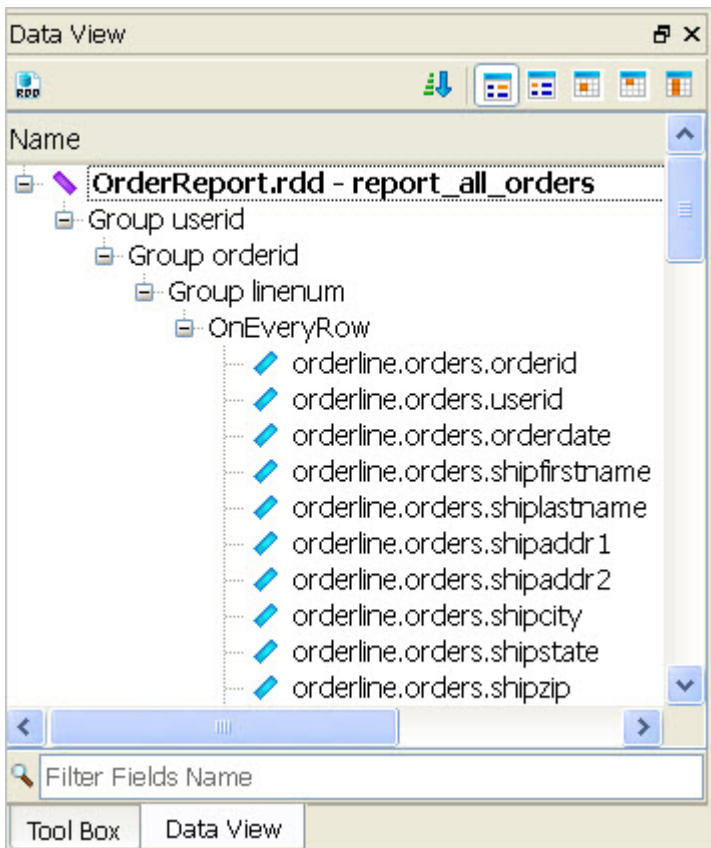


Figure 9-35. Report Designer Data View

A single .rdd file can be used to design multiple reports containing the values of various data items from the file. The same data items can also be displayed as different types of charts.

#### The Design:

1. Open a new or existing report in Report Designer.  
If you are creating a new report, specify the .rdd file to be used for the report.
2. From the ToolBox, drag and drop the desired Graph ( Map Chart, Category Chart, XY Chart) into a container on the Report Design Window.

The Design Window will contain a Chart object and its related Item object (shown as a price tag). If you cannot view the entire chart in the Report Design window, you can re-size the display of the report in the window: point the mouse at the report and scroll the mouse scroll button while holding down the keyboard Control key.



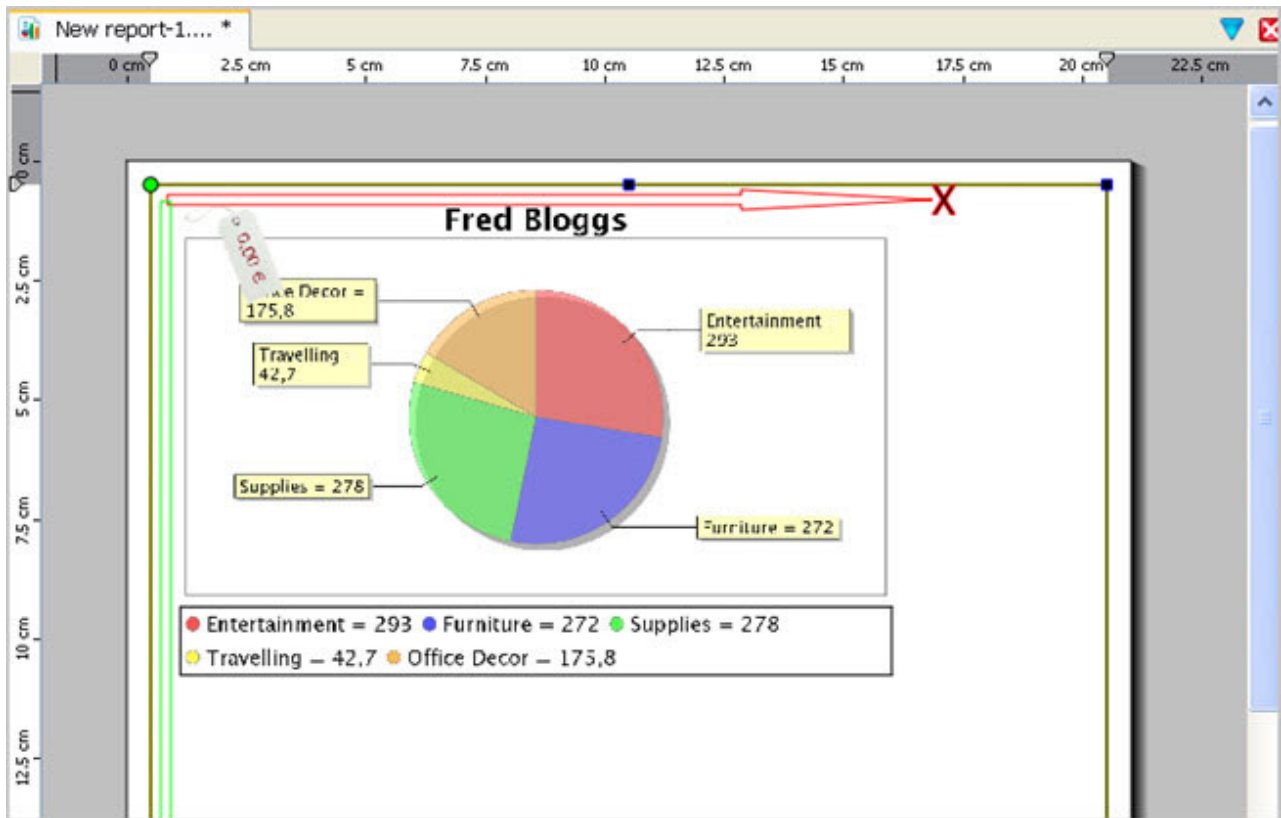


Figure 9-36. Chart object in the work area

3. Select the chart object ( Map Chart, Category Chart, XY Chart), and enter the values for its properties in the Properties View. The properties vary depending on the graph chosen:
  - Title - caption at the top of the graph
  - Values Title - caption for the values
  - Keys Title - caption for the keys
  - Category Title - caption for the categories (of a Category Chart)
  - X Axis Title - caption for the X axis (of an XY Chart)
  - Y Axis Title - caption for the Y axis (of an XY Chart)
  - Draw As - the type of chart
4. Select the chart's price tag, which represents the item object ( Map Chart Item, Category Chart item, XY Chart item), and enter values for its properties. The specific properties vary depending on the graph type.
  - **keys** - the data items that are used to group values, These data items must be Strings. You are not limited to existing String data items, however. You can define a custom string for the key, using the data items in a expression.
  - **values** - the data item that contains the numbers to be charted. These data items must be Numeric.
5. Modify the tree in the Structure View as needed:
  - The item object for the chart ( Map Chart Item, Category Chart item, XY Chart item) must be a child of the OnEveryRow trigger node.
  - The location of the chart object ( Map Chart, Category Chart, XY Chart) specifies how many different charts will be created, based on the resulting subsets of the data.

Drag and drop the nodes in the Structure View to create the correct hierarchy. If you drop a container or trigger on a different node, it will be a child of that node. Use alt-drag and it will become the parent of that node.

### Examples

With Figure 9-37, the Structure View that resulted from the creation of a new report containing a chart. It will create a new page and chart for each data row passed to the report, and therefore needs to be modified:

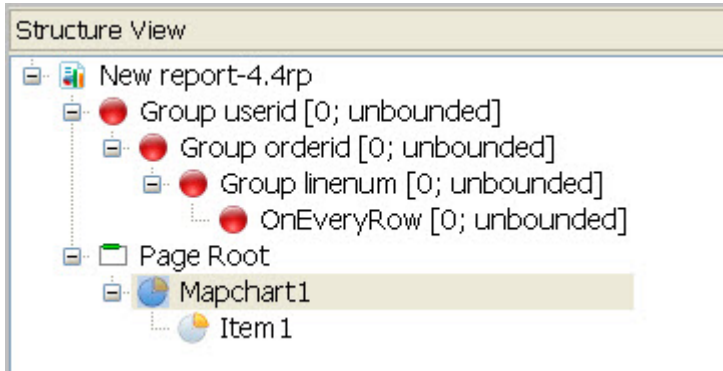


Figure 9-37. One chart per each row

With Figure 9-38, the Structure View will result in a page containing one chart; the item node is a child of OnEveryRow:

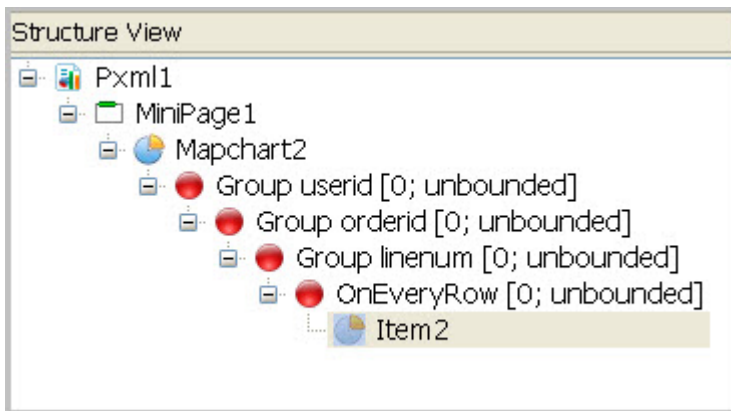


Figure 9-38. One chart, one page

With Figure 9-39 on page 9-81, the Structure View will result in one chart for every unique userid, since the Map Chart is a child of the userid trigger node; the item node is a child of OnEveryRow:



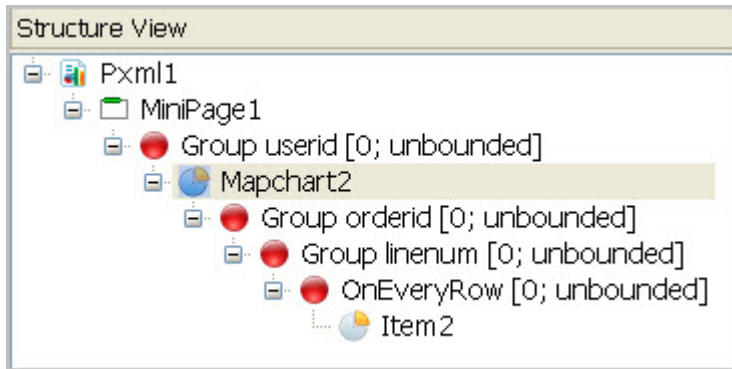


Figure 9-39. One chart for each unique userid

**Custom Keys:** You can enter any valid expression for the String value of a key property. This could be a substring or a concatenation of existing Strings. For example, the following expression would group the data values based on the first letter of shiplastname:

```
orderline.orders.shiplastname.substring(0,1)
```

### Examples

This chart uses the last name as the key (trimmed of trailing blanks), as shown in the Properties View. The unit price on each order for each unique last name is rolled up to a total as shown on the chart. There are five unique names:

| Properties |                                        |
|------------|----------------------------------------|
| Name       | Value                                  |
| Items      |                                        |
| Key        | {orderline.orders.shiplastname.trim()} |
| Value      | {orderline.lineitem.unitprice}         |
| Name       | Item2                                  |

Figure 9-40. Properties panel for the chart Items (where key uses trimmed last name)

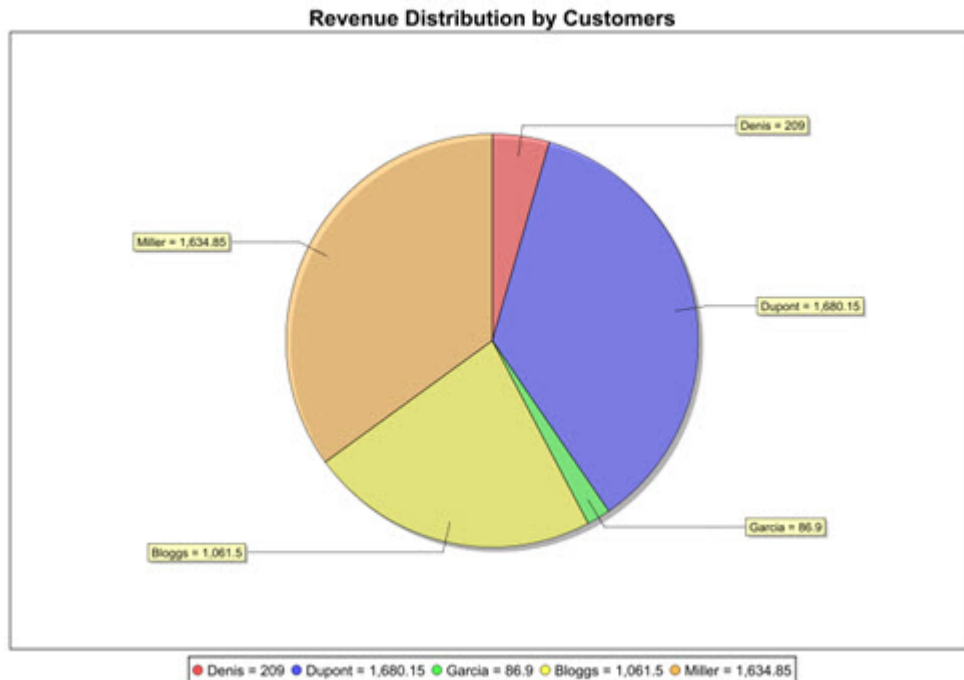


Figure 9-41. Chart showing Revenue Distribution by Customers (where key uses trimmed last name)

And this chart uses the first letter of the last name as the key, as shown in the Properties View. The unit price on each order for each unique first letter is rolled up to a total as shown on the chart. In this chart, there are only four unique first letters, as two customers have last names beginning with D:

| Properties |                                              |
|------------|----------------------------------------------|
| Name       | Value                                        |
| Items      |                                              |
| Key        | orderline.orders.shiplastname.substring(0,1) |
| Value      | {orderline.lineitem.unitprice}               |
| Name       | Item2                                        |

Figure 9-42. Properties panel for the chart Items (where key uses substring)

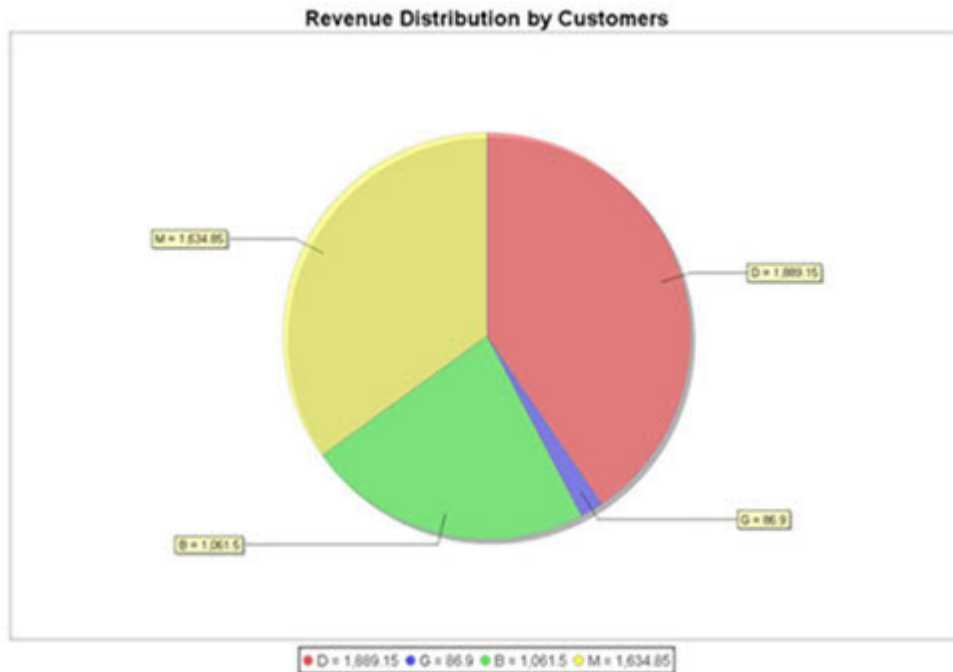


Figure 9-43. Revenue Distribution by Customers (where key uses substring)

## Output Charts as Tables

All chart types now support the drawing of the data as tables via the Draw As property. The tables drawn are "pivot" tables that may contain subtotals. The following terminology and rules are applied:

- The columns of the tables are either of type **Dimension** (the data items that are used to group values ) or of type **Value** (see Mapping of Chart properties).
- **Values** are aggregated or totaled by the **Dimension**.
- **Dimension** columns precede **Value** columns in the table.
- The order of the **Dimensions** specifies the order of the data.
- Subtotals are generated for each **Dimension** except the rightmost.

## Mapping of chart properties

- Map chart - the Key property is mapped to a **Dimension** column, and the Value property to a **Value** column.
- Category chart - the Category Key property and the key property are mapped to **Dimension** columns, and the **value** property to a **Value** column.
- XY chart - the Series Title property is mapped to a Dimension column and the X and the Y properties are mapped to Value columns.

There are three types of tables: Table, SortedTable, and AggregatedTable.

**Table:** This option lists all data items in a table. The data needs to be presorted in the order of the dimensions; if this is not the case, the table will contain useless subtotal rows.

If the number of rows in the table is large, then **Table** is the preferred choice since it produces the tabular output row by row while reading the input and does not

keep a copy of the table data in memory. In other words, this option does not delay the output until the end of the input has been read. as the Sorted Table and Aggregated Table options do.

**Sorted Table:** This option produces the same output as the Table option, but the data does not need to be presorted. The output is delayed until the last row of table data has been read, and the entire table data is stored in main memory.

**Aggregated Table:** This option draws the same table as the previous two options, but subsequent rows with identical dimensions are drawn only once and the total values are computed. This option always sorts the data, and delays the output until the last row of data has been read. This option is not available for XY chart types.

## Expressions in Properties

### Topics:

- Overview
- Using the RTL Expression Language
  - Operators
  - Conditional Expressions
  - Operands
  - 4GL Variables
  - Examples
- Using RTL Classes
  - Classes
  - Members
  - Examples
- Using the PXML Expression Language
  - Units of Measure
  - Variables
  - Functions
- Substituting 4GL Variables for Constants

See also: Dimension Resolver

### Overview

The value for a property of a report item can be a literal value, or it can be derived from an expression that is written using the RTL Expressions language. RTL Expressions allow you to define runtime values for any property of a report item, except for those properties that display a specific set of valid values in a dropdown listbox.

Enter an expression in a field of the Properties View, by clicking the Value field. To enter longer values, or obtain hints while typing the entry, click the **f(x)** button of the desired property to open the RTL Expression Editor:

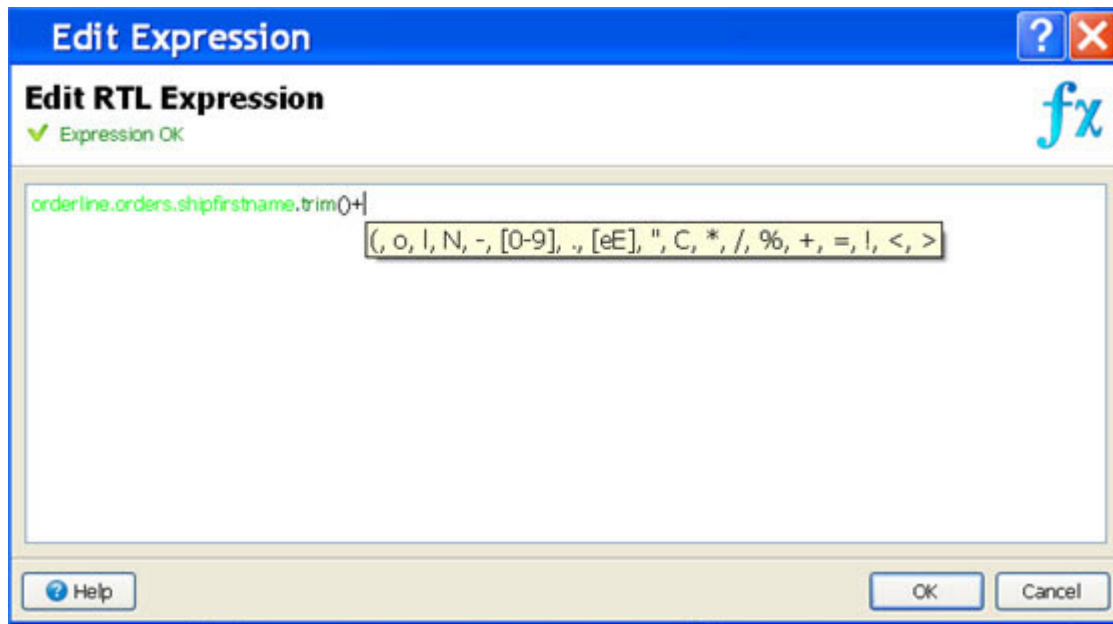


Figure 9-44. Edit Expression dialog

The acceptable choices for the next character are displayed as you type. If you press Ctrl Space while entering an expression, a Code Completion box appears containing a list of the valid choices based on the context.

RTL Expressions:

- **are typed** - an expression is composed of items of different types and the expression returns a particular type. The properties in the Properties View are also typed. *Any expression entered as a value for a property must return the specified type.* For example, the text property of a WordBox has a type of String. So, any RTL expression for the text property is expected to yield a String. See Report Element Properties for a list of each report property and its type.
- **closely follow the Java syntax** for expressions. The main difference is that the "new" keyword is not supported, so that it is not possible to create and subclass objects. RTL Expressions loosely follow the Java evaluation semantics (operator precedence, evaluation order, etc.).
- **can use 4GL variables.** The 4GL variable is converted to a specific type within the expression (See Conversion table) .

RTL Classes provide member functions (methods) and member variables that you can use in your expressions. There is a class for each type of a form property.

**Important:** Numeric data types are limited to 15 significant digits. To avoid problems with totals in reports, do not calculate the value of an item that is part of a total using an RTL expression. Perform any calculations in the BDL program.

## Using the Language

An Expression is a sequence of operands, operators, and parentheses that the runtime system can evaluate as a single value.

The RTL Expression language follows the Java syntax for expressions and evaluation semantics. Expressions can include the following components:

- "Operators" on page 9-86

- “Conditional Expressions”
- “Operands”
- “4GL Variables” on page 9-87

## Operators

Table 9-6. RTL Operators

| Operator | Description                      | Example         | Precedence |
|----------|----------------------------------|-----------------|------------|
| %        | Arithmetic: Modulus              | x % 2           | 8          |
| *        | Multiplication                   | x * y           | 7          |
| /        | Division                         | x / y           | 7          |
| +        | Addition                         | x + y           | 7          |
| -        | Subtraction                      | x - y           | 6          |
| +        | Concatenation                    | string + string | 5          |
| <        | Relational/Boolean:<br>Less than | numeric < 100   | 4          |
| <=       | Less then or equal to            | numeric <= 100  | 4          |
| >        | Greater than                     | numeric > 100   | 4          |
| >=       | Greater than or equal<br>to      | numeric >= 100  | 4          |
| ==       | Equal to                         | numeric == 100  | 4          |
| !=       | Not equal to                     | numeric <> 100  | 4          |
| !        | Logical inverse<br>(NOT)         | !(x = y )       | 3          |
| &&       | Logical intersection<br>(AND)    | expr1 && expr2  | 2          |
|          | Logical union (OR)               | expr1    expr2  | 1          |

The first column in the table describes the precedence order of the operators, listed highest to lowest. For example, the % modulus operator has a higher precedence than the \* operator. Parentheses can be used to overwrite the precedence of operators.

## Conditional Expressions

Conditional expressions allow you to express IF/ELSE statements.

Syntax:

```
Boolean-expression? expression-1: expression-2
```

The ? operator indicates that this expression is conditional; the return value is dependent on the result of the Boolean expression. If the Boolean expression is TRUE, the first expression is the return value; otherwise, the second expression is the return value.

## Operands

Operands include the following:

- Literal values

- Other expressions
- 4GL Variables
- RTL Class Members
  - Objects
  - Methods (returning a single value)

A literal value for a string in an expression should be delimited by double quotes: "Test".

## 4GL Variables

The data types of 4GL variables are taken into account when constructing expressions. For every 4GL variable an object is created that is either an instance of a FGLNumericVariable or an FGLString Variable. These objects hold the value of the 4GL variable, and at the same time they contain a member variable **value** which also contains the value. For this reason, it is legal to write "order\_line.itemprice" in your expression as a shortcut for "order\_line.itemprice.value". Both types of objects have the following specific member variables defined:

- **value**- value of the 4GL variable
- **caption**- the title of the field
- **name**- the name of the variable
- **type** - the RTL type of the variable
- **isoValue**- the locale and formatting-independent representation of the value of the variable

The following conversion table lists 4GL data types and the type into which they are converted within an RTL expression:

*Table 9-7. 4GL data types and the type into which they are converted within an RTL expression*

| 4GL type                                                | Corresponding RTL type                                                                                                                                                                                                               |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHAR, VARCHAR, STRING and TEXT                          | FGLStringVariable                                                                                                                                                                                                                    |
| DATE, DATETIME and INTERVAL                             | FGLNumericVariable                                                                                                                                                                                                                   |
| INTEGER, SMALLINT, FLOAT, SMALLFLOAT, DECIMAL and MONEY | FGLNumericVariable, <b>limited to 15 significant digits</b> . The value of a number larger than 15 digits will be truncated, and the resulting number is rounded. For example, 12345678901234567 will be rounded to 123456789012346. |

**Important:** To avoid problems with inaccurate totals on a report due to rounding, do not perform RTL arithmetic on either the individual values or the total value; calculate the value of the item in the BDL program instead, before passing the value to the report.

## Examples

For the purpose of these examples, order\_line has been replaced with order.

1. To add 10% to the itemprice: order.itemprice\*1.10

The data item `order_line.itemprice` is converted to a Numeric type, so we can use the Numeric operators. In order to display the result of a Numeric expression in a WordBox, we must convert the result to a String. See Example 1 in the Using RTL Classes section.

2. Let's add 10% to the item price conditionally, depending on the value:  
`order.itemprice<100?order.itemprice*1.10:order.itemprice`

The condition in this Boolean expression tests whether the `itemprice` is greater than 100; if so, the value returned is 110% of the `itemprice`; otherwise, the value returned is simply the `itemprice`.

3. To set the font of a report item to italic when the 4GL variable **`order_line.lineitemprice`** exceeds \$20, we must create an expression for the `fontItalic` property: `order.lineitemprice>20`

The property `fontItalic` is of type boolean, so any RTL expression that we use for that property must return a boolean value (TRUE/FALSE). Any of the relational operators yields a boolean, so the type of the returned value of this expression is a boolean (The expression will return TRUE if the `lineitemprice` exceeds 20).

**Note:** A numeric value by itself is not a boolean value as it is in some programming languages.

## Using RTL Classes

RTL expressions do not contain the primitive datatypes "byte", "short", "int", "long", "float", "double", "boolean" and "char". Instead, everything is expressed as objects. All methods are member functions. There are no global functions.

## Basic Object Classes

There are object classes for each type of the form item properties. The type of each property is indicated in the Properties page.

- String - contains methods used for all string operations
- Numeric - contains methods used for all numeric operations; the class has the precision of a double and the arithmetic operators are defined for objects of this type.
- Boolean - contains methods used for all logical operations
- Color - contains methods and static member variables related to color.
- Enum - a set consisting of a class for each property of this type; each class contains static member variables that provide a list of valid values for the corresponding property.
- Date Class - a class representing a Date value; contains methods for parsing and formatting strings.

## Members

*Instance Member Methods:*

These methods are called on an object instance. You can get an object instance by referencing a 4GL variable or by calling a method on another object. You can also use a literal value as an object instance.

When you invoke the method, it is prefixed with the object instance name and the "." character.



Examples of instance methods are expressions like "order\_line.customer\_name.trim()". This is valid because the 4GL variable **order\_line.customer\_name** has a CHAR data type, which is converted within the RTL Expression to an object of the type String. And, the method **trim** is a member function of a String object.

Methods always yield objects, so it is also legal to call methods on the return value of a method.

#### *Static Member Methods:*

These methods do not require an object instance. When you invoke the method, it is prefixed with the classname and the '.' character.

#### *Static Member Objects:*

These objects are member variables that do not require an object instance. The objects are prefixed with the classname and the '.' character. Examples of static member objects are expressions like "Color.RED" or "Numeric.PI".

### Examples

1. This example concatenates the first and last names of a customer, using the trim method of the String class and the + operator:

```
order_line.shipfirstname.trim()+" "+order_line.shiplastname.trim()
```

This expression prints the first name (trimmed of trailing blanks), a string consisting of a single space, and the last name (trimmed). Use double quotes instead of single quotes to delimit strings.

2. Parenthesis can be used to change the order of operations. For example:

```
(order_line.unitprice+10).toString()
```

Parentheses are used to force the addition to be done prior to the conversion to a String.

3. The following conditional expression used in the color property of the **order\_line.unitprice** WordBox will change the color to red if the value is less than 20:

```
order_line.unitprice<20?Color.RED:Color.BLACK
```

This expression specifies that the return value when the boolean expression is TRUE is the static member variable RED of the Color class, otherwise the return value is the static member variable BLACK.

- 4.

It is legal to call methods on the return value of a method. For example, the following is a valid expression:

```
order_line.customer_name.trim().toUpperCase().substring(1)
```



Figure 9-45. `order_line.customer_name.trim().toUpperCase().substring(1)`

In this example, the object **order\_line.customer\_name** is a BDL CHAR variable; CHARs are assigned to the String type. The String method **trim()** is called first, returning the String object a. The method **toUpperCase()** is called for the object a, returning object b which will be in upper case. Finally, the method

**substring()** is called for the object *b*, returning object *c*. If the *customer\_name* is "Springs", the resulting object *c* is the string "PRINGS".

There are many additional examples of expressions in the properties of report elements defined in the .4rp programs that are part of the GRWDemo project.

## Using the PXML Expression Language

Genero Report Writer provides the PXML Expression language to define the value of a property that is of the **PXML** (dimension) type. The type of each property is specified in the Properties page of the Report Writer documentation.

A PXML expression always yields a Numeric value. The value is expressed in units of measurement. It is legal, for example, for the value to be **10in**. If no unit is specified, the unit is presumed to be points. When you specify a value in units, it is converted internally to its equivalent value in points.

## Units of Measure

The most commonly used units are:

- **point** | **pt**
- **pica** | **pc**
- **inch** | **in**
- **cm**
- **mm**

See Dimension Resolver for additional explanations and examples of the units that can be used.

## Variables

The following variables can be used in any PXML expression to define the layout dynamically:

- **max** - the maximum extent of the current parent box
- **min** - the minimum extent of the current parent box
- **rest** - the remainder of the current parent box

For example, to center an element in its parent container you can use the **max** variable for the following properties:

*Table 9-8. Centering an element*

| Property | Value |
|----------|-------|
| x        | max/2 |
| y        | max/2 |
| anchorx  | 0.5   |
| anchory  | 0.5   |

You can use the **rest** variable in the Y-Size property of a MiniPage container to force a page break by consuming the remainder of the container:

*Table 9-9. Using rest*

| Property | Value |
|----------|-------|
| Y-Size   | rest  |

## Functions

The PXML Expression language has a 4GL-like syntax. The most commonly used functions are:

- **max** ( valueA, valueB) - this is a function, not the variable listed in “Variables” on page 9-90
- **min** (valueA, valueB)
- **length** (value)
- **width** (value)

For example, the following expression uses the functions **max** and **width** :

```
max(10cm,width("HELLO"))
```

It first calculates the width of the string "HELLO", taking the current font metrics into account. Next, it compares the value "10cm" and the width of "HELLO", and returns the larger value.

## Substituting 4GL variables for constants

You may want your expression to depend on a 4GL data variable rather than on the constant string "HELLO". For this, we use RTL embedded in curly braces (Note that we are now mixing two languages. Within the braces it is RTL, outside it is a PXML expression). The rules for embedding RTL in PXML are:

- wherever a numeric constant is allowed in PXML, you can insert an RTL numeric expression (enclosed in curly braces) instead.
- wherever a string constant is allowed in PXML, you can insert a RTL string expression (enclosed in curly braces) instead.

The expression "max(10cm,width("HELLO"))" contains one numeric constant "10" and one string constant "HELLO" which can be replaced so that we can write:

```
max({order_line.titlewidth}cm,width({order_line.title}))
```

For this expression to be legal, the 4GL variable **order\_line.titlewidth** has to be of type Numeric, and the 4GL variable **order\_line.title** has to be of type String.

**Note:** You cannot construct a dynamic expression where the function names, such as max or width, or the unit names, such as cm, are dynamic.

---

## Reference

### Topics:

- “Report Design Elements (The Toolbox)”
- “Report Element Properties” on page 9-102
- “Reporting API Functions” on page 9-119
- “Bar Codes” on page 9-155
- “RTL Classes Overview” on page 9-185
- “Dimension Resolver” on page 9-186

## Report Design Elements (The Toolbox)

### Topics:

- Simple Containers
  - Horizontal Box (Layout Node)

- Vertical Box (Layout Node)
- Propagating Containers
  - Horizontal Box (Mini Page)
  - Vertical Box (Mini Page)
  - Page Root (MiniPage)
  - Stripe (MiniPage))
- Drawables
  - Word Box
  - Word Wrap Box
  - Decimal Format Box
  - Page Number Box
  - Image Box
  - HTML Box
- Business Graphs
  - Map Chart
  - Map Chart Item
  - Category Chart
  - Category Chart Item
  - XY Chart
  - XY Chart Item
- References
  - Reference Box
  - Info Node
- Bar Codes
  - Bar Code Boxes

## Simple Containers

### Horizontal Box, Vertical Box (Layout Node)

**Container:** A Layout Node is a rectangular area in the Report Design page. A Layout Node does not propagate; the content is not allowed to overflow the container. As a result, a Layout Node can be used for content that should be kept together on the page.

This **Horizontal Box** is a Layout Node with the Layout Direction property set **lefttoright**; elements in the box are laid out across the page. The **Vertical Box** is a Layout Node with the layout direction property set **toptobottom**; elements in the box are laid out down the page.

### Properties

Select the object on the Report Design page to display its properties in the Properties View.

You can change the object's default appearance by setting the values of the following properties:

name, color, bgColor, fontName, fontSize, X-SizeAdjustment, Y-SizeAdjustment, fontBold, fontItalic

Some specific properties allow you to define borders, margins, and padding for the boxes:

marginWidth, marginRightWidth, marginBottomWidth, marginLeftWidth, marginTopWidth, borderWidth, borderRightWidth, borderBottomWidth, borderLeftWidth, borderTopWidth, paddingWidth, paddingRightWidth, paddingBottomWidth, paddingLeftWidth, paddingTopWidth, borderStyle, borderRightStyle, borderBottomStyle, borderLeftStyle, borderTopStyle, borderColor, borderRightColor, borderBottomColor, borderLeftColor, borderTopColor, roundedCorners

The x-Size and y-Size properties specify the INNER size of the box. Adding borders, for example, will increase the overall size.

Layout Nodes can be used as a container for content that must print at a specific part of the MiniPage, a page header, for example, by setting its section property.

The clip property can be used to clip the object box and its content along the sides. This property applies to all layout nodes.

Other properties have values that are derived from the type and position on the page, adjust automatically if the object is moved or re-sized, and need not be changed manually:

type, baselineType, layoutDirection, swapX, alignment, scaleX, scaleY, x, y, anchorX, anchorY, floatingBehavior

## **Propagating Containers (Mini Pages)**

### **Mini Page**

**Container:** Mini Page. This container formats the report page into lines and columns.

A Mini Page is a propagating box. The boxes can handle unknown amounts of material; if the box is full, a copy is made and the leftover material flows to the copy or copies, as needed. A MiniPage cannot be used as a container for a page header, page footer, or separator.

### **Page Root (Mini Page)**

**Container:** Mini Page. Page Root is the recommended base container when you start creating a report. It is a Mini Page with the height and width properties set to maximum. The container propagates; if it is full, a copy is generated and the extra content overflows to the copy.

### **Stripe (MiniPage)**

**Container:** Mini Page. This container has the y-size property set to "max". This container is recommended for content that stretches horizontally across the report page (lines in a report, for example.) The container propagates; if it is full, a copy is generated and the extra content overflows to the copy.

### **Horizontal Box, Vertical Box (Mini Page)**

**Container:** Mini Page. These containers have their orientation ( layoutDirection "swapX" on page 9-111property) set to display the box content horizontally

(lefttoright) or vertically (toptobottom). The containers propagate; if a container is full, a copy is generated and the extra content overflows to the copy.

## Properties

The following properties are specific to Mini Page:

Hide PageHeader OnLastPage, Hide PageFooter OnLastPage

Additional properties are inherited from Propagating Box and Layout Node. The property floatingBehavior allows you to specify whether the parent container will resize itself so that this Mini Page object is enclosed in the parent.

## Drawables

### Word Box and Word Wrap Box:

#### Word Box

**Drawable:** Use this object for a specified chunk of text, which will use the current font.

The following properties are specific to Word Box:

trimText, underline, strikethrough, fidelity, localizeText

#### Word Wrap Box

**Drawable:** This object is like a WordBox with paragraphs of uniform text.

The following properties are specific to Word Wrap Box:

trimText, indent, fidelity, localizeText

## Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

The text property specifies the string to be displayed in the WordBox or Word Wrap Box.

You can set the textAlignment property for a Word Box or Word Wrap Box to **left**, **right**, or **center**. The alignment does not influence page break positions even if the indent property is set to some value. For Word Wrap Boxes, the textAlignment property can also be set to **justified**.

The localizeText property enables the localization of text content in Word Boxes and Word Wrap Boxes.

Additional properties are inherited from Layout Node.

### Tip:

- Don't set the Y-Size (height) property on a Word Wrap box, because the element should typically grow in accordance with its content. If you set a fixed Y-Size, you'll prevent that automatic enlargement.

- The text value of these boxes can be edited directly in the report design document; double-click the object and the input cursor will be placed in the text. The layout will be updated on each keystroke.
- To force a line break, use the newline character "\n".
- Beginning in version 2.4x, a Word Wrap Box can span pages; when the available vertical space for WORDWRAPBOX is not sufficient to display the entire text, the box now propagates the exceeding content to additional boxes, with the same behavior as a propagating MINIPAGE.

**Decimal Format Box: Drawable:** Use a Decimal Format Box for decimal numbers. It has a variety of features designed to make it possible to parse and format numbers in any locale, including support for Western, Arabic, and Indic digits. It also supports different kinds of numbers, including integers (123), fixed-point numbers (123.4), scientific notation (1.23E4), percentages (12%), and currency amounts (\$123). All of these can be localized. The value of the number is limited to 15 significant digits.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

format

The value of the text property can also be edited directly in the report design document; double-click the object and the input cursor will be placed in the text. The layout will be updated on each keystroke.

**Page Number (PageNo Box): Drawable:** Use this object for page numbers. In order to provide for virtual pages (multiple logical pages on one physical page) a pageName property can be specified to identify the logical page.

Specific functions are available to allow you to calculate an expression for the page number such as Page N of M, using the property textExpression. If a value is provided for textExpression, the pageName, the pageNoOffset, and pageNoFormat properties are irrelevant.

If values for either the textExpression property or the text property are not set, a default length is calculated. See Page Numbers.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

The following properties are specific to PageNo Box:

pageName, pageNoOffset, pageNoFormat, textAlignment, textExpression

Additional properties are inherited from Word Box.

**Image Box: Drawable:** Use this object for images. On all platforms GIF, JPEG, PNG, BMP, WBMP, and SVG formats are supported. The images are cached on a per document basis. The default resolution is the current screen resolution.

Use the Location property to specify the file name and path of the image to be rendered. Press the ... button to open a dialog and select the image located in your file structure.

If you want to preserve the aspect ratio of an image, set the value of either length (y-size) or width (x-size), allowing Report Writer to calculate the corresponding value. If you set both properties, the resulting image will appear distorted.

**SVG images** - Using SVG images instead of bitmap images can substantially reduce the document size, which is particularly useful when PDF documents are produced. When providing the SVG content from a 4GL variable, use the mime type "image/svg" so that the url looks something like "data:image/svg,..." when read from a string variable and "data:image/svg;base64,..." when read from a BLOB. The currently supported SVG version is "1.2 Tiny" (See <http://www.w3.org/TR/SVGTiny12>).

### Embedding images

To imbed an image, we use a URL type that allows encoding the data in the body of the URL text. The syntax is:

data:image/png;base64, followed by the base 64 encoded bitmap data of the image

To automatically construct this URL for an image, press the ... **button** for the Location property. When the Open dialog displays, select the image file located in your file structure, and check the **Embed in document** box at the bottom of the dialog:

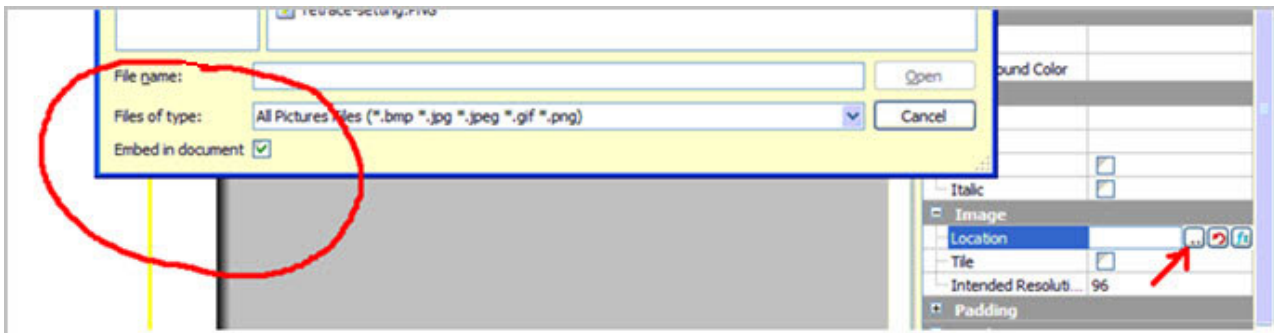


Figure 9-46. Embed in document checkbox

### Populating images from blob variables

If you drag a BYTE variable from the Data View onto the Report Design, an Image Box object will be created. If the variable was named "imageblob" then the Location property would automatically be filled with the following formula:

```
data:image/jpg;base64,"+imageblob
```

where **imageblob** is the name of the blob

This has the same effect as pressing the **formula button** for the Location property, and entering the formula including the blob variable name.



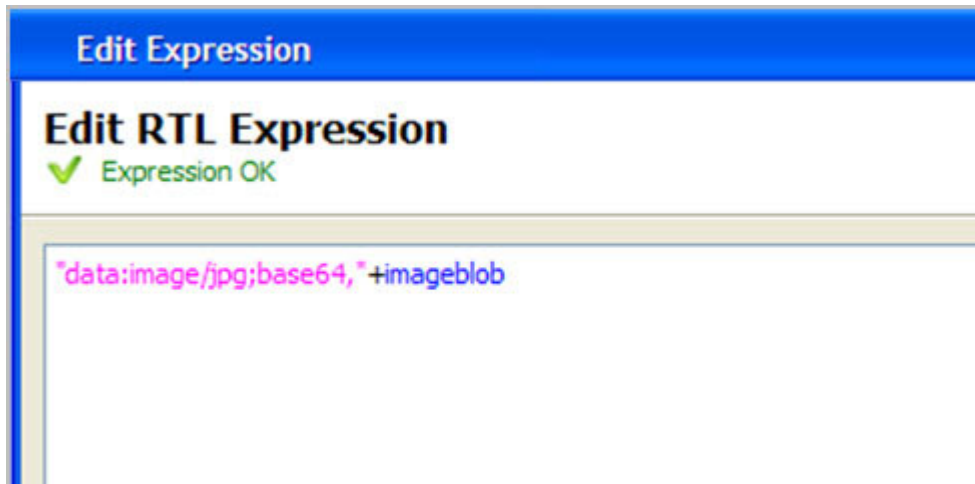


Figure 9-47. Entering blob variable name in RTL Expression editor

**Note:** Since a blob variable may contain images of various types, the implementation ignores the image type declared in the formula, and looks at the encoded data itself to determine the image type. This formula would work for blobs that were of **png** type also.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

The Location property is used to specify the image.

The following properties are specific to Image Box:

Intended Resolution, URL, fill

**Note:** Beginning with version 2.4x, the **tile** property is replaced by a new property, **fill**. If you open a document containing the **tile** property, it will be automatically replaced by **fill**.

**HTML Box: Drawable:** This element displays an image of an HTML document in the report.

Use the Location property to specify the file name and path of the document whose image is to be displayed. Press the ... button to open a dialog and select the html file.

**Note:** The content of an HTML box cannot span pages.

### Embedding HTML

To embed HTML, we use the data protocol syntax in the Location property of the HTMLBOX element. The syntax is:

data:text/html, followed by the percent encoded data of the html document

To automatically construct this URL for HTML documents, press the **...** button for the Location property. When the Open dialog displays, select the HTML file located in your file structure, and check the **Embed in document** checkbox at the bottom of the dialog:

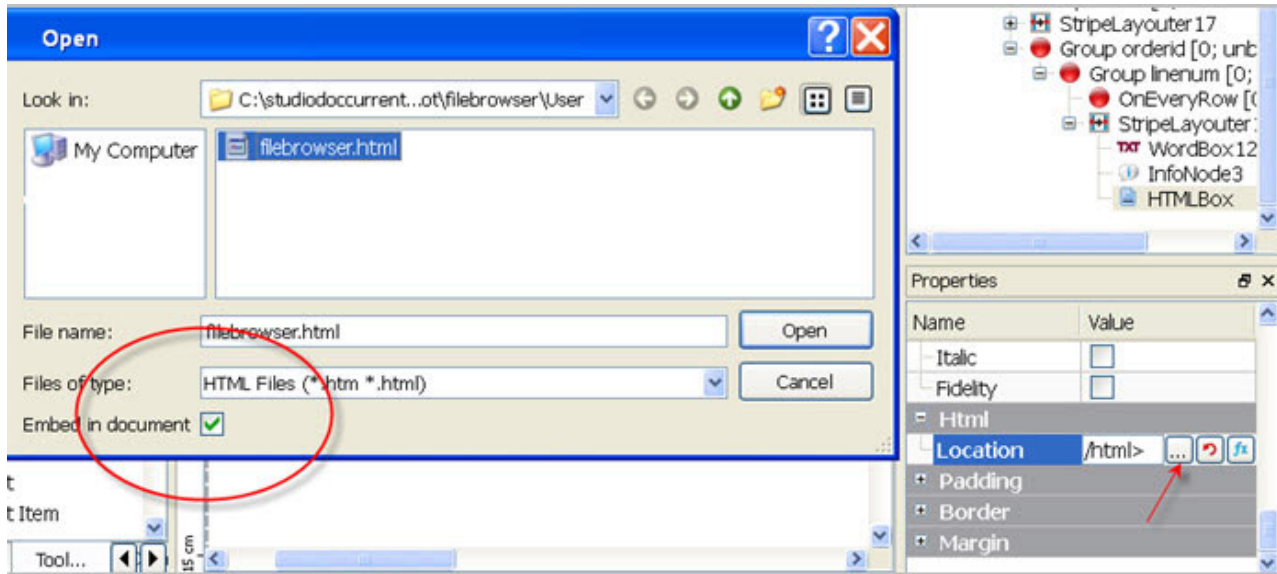


Figure 9-48. Embed in document checkbox

### Populating HTML content from text variables

Data in text variables was typically input into a database via a TextEdit form field, with the textFormat style attribute set to **html**. Refer to the Genero Business Development Language manual topic "Presentation Styles" for more details.

Press the **formula** button for the Location property of the HTML Box, and enter the expression, including the name of the text variable.



Figure 9-49. RTL Expression

The function `String.urlEncode()` is used to encode the data using percent encoding.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

The Location property is used to specify the html content.

## Business Graphs

**Map Chart: Business Graph:** the MAPCHART element defines the header for an abstract map dataset that can be used for creating a variety of one dimensional graphs such as pie charts. The map items are defined using the ITEM element. The resulting chart is drawn automatically. See Working with Business Graphs for additional information.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

title, valuesTitle, keysTitle, drawAs

The valid values of **drawAs** for this object are: Pie | Pie3D | Ring | Bar | Bar3D. The default is a Pie.

**Map Chart Item: Business Graph:** defines the data value items for a Map Chart.

### Properties

Select the object on the Report Design page to display its properties in the Properties View.

key, value

**Category Chart: Business Graph:** the CATEGORYCHART element defines the header for an abstract category dataset that can be used for creating a variety of two dimensional charts such as category charts. The categories are defined by the CATEGORY element and its "key" attribute, which has to be unique within a CATEGORYCHART. Within a CATEGORY, CATEGORYITEMS define the values within the category. Within one category, the "key" values of individual CATEGORYITEM elements has to be unique. The resulting chart is drawn automatically. See Working with Business Graphs for additional information.

### Properties

Select the object on the Report Design page to display to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

title, valuesTitle, keysTitle, drawAs

The valid values of **drawAs** for this object are:  
Bar | Bar3D | Area | StackedBar | StackedArea | Line | Line3D | Waterfall. The default is a Bar.

**Category Chart Item: Business Graph:** defines the data value items for a Category chart.

### Properties

Select the object on the Report Design page to display to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

category Key, key, value

**XY Chart: Business Graph:** The XYCHART element defines the header for an abstract XY dataset that can be used for creating a variety of XY-Plots such as line or scatter plots. The XY data is defined by XYITEM elements. The resulting plot is drawn automatically. See Working with Business Graphs for additional information.

### Properties

title, xAxisTitle, yAxisTitle, drawAs

Value values of **drawAs** for this object are:  
Polar | Scatter | Area | StackedArea | Line | Step | StepArea | TimeSeries. The default is a Scatter.

**XY Chart Item: Business Graph:** defines the data value items for an XY Chart.

### Properties

Select the object on the Report Design page to display to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

seriesTitle, x, y

## References

**Reference Box: Reference:** This object allows you to create layout-dependent text output like "Total from previous pages: num".

Since the space to be allocated may not be known until the report is run, make sure that there is enough space available to display any possible text. Use the **text** property to provide an example, based on the underlying data type of the InfoNode object. This is only used to determine the maximum space to set aside. For example:

- Data types that are numeric - "000,000,000.00"
- Data types that are strings, for example CHAR(8) - "MMMMMMMM"

## Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

This object works in conjunction with an Info Node object, and its specific properties reference the Info Node:

- InfoNode Name - the name of the InfoNode to be referenced
- default - text to be displayed if the reference cannot be resolved. The default string is "-".

Additional properties are inherited from PageNo Box, Word Box.

See DesignHowTo for an example, and the demo programs provided with the product.

**Info Node: Reference:** This object helps resolve some layout-dependent problems by enabling the use of references. This node is invisible and does not consume space in the layout. This is the object referenced by a Reference Box, to print layout specific information, such as a total from a previous page, for example.

## Properties

The value is stored for querying by a Reference Box. The value is of type String. If the data type of the field being referenced does not correspond to a String, the value must be converted.

See DesignHowTo for an example of this use, and the demo programs provided with the product.

## Bar Codes

**Barcode Box: Bar Code:** Use this object for bar codes. Example:



Figure 9-50. Sample Barcode

The currently supported types are: U PC-A, UPC-E, UPC Supplementals 2 and 5, EAN-13, EAN-8, EAN Supplementals 2 and 5, Code 128 (EAN 128), Code 2/5 Industrial, Code 2/5 Inverted, Code 2/5 IATA , Code 2/5 Interleaved, Code 2/5 Matrix, Code 2/5 Datalogic, Code BCD Matrix, Code 11 Matrix, Code 39, Code 39 extended, Code 32, Code 93, Code 93 extended, Codabar 18, Codabar 2, and

data-matrix. For licensing reasons, it may be necessary for the user to supply the fonts required to draw the text for some types of bar code.

Bar codes are drawn in nominal sizes. By setting the `scaleX` and `scaleY` properties it is possible to draw larger or smaller versions. It is further possible to force a particular width and/or length but specifying the desired extend value.

Specific functions are available to allow you to calculate an expression for the page number such as Page N of M, using the property `codeValueExpression`.

See Bar Codes for a complete description of bar codes.

### Properties

Select the object on the Report Design page to display its properties in the Properties View. You can change the object's default appearance by setting the values of its properties.

The following properties are specific to Barcode Box:

`codeType`, `fidelity`, `noText`, `codeValue`, `check`, `noDigits`, `noCheckDigits`, `thinToThickRelation`, `thinToGapRelation`, `controlCharacters`, `codeValueExpression`

Additional properties are inherited from Layout Node, and specific bar codes may have unique properties.

See Bar Codes for a description of all the possible bar code types.

## Report Element Properties

Values for these properties may be literal, or they may be RTL expressions. If expressions are used, the resulting value must be of the specified data type of the property.

See Using Expressions in Properties.

- “General Properties”
- “Properties related to margins and borders” on page 9-115
- “Properties for Report Document Metadata” on page 9-118

### General Properties

- `alignment`
- `anchorX`
- `anchorY`
- `baselineType`
- `bgColor`
- `categoryKey`
- `categoriesTitle`
- `check`
- `clip`
- `codeType`
- `codeValue`
- `codeValueExpression`
- `color`

- controlCharacters
- data
- drawAs
- drawLegend
- drawLabels
- fidelity
- fill
- floatingBehavior
- fontBold
- fontItalic
- fontName
- fontSize
- format
- href
- id
- intendedResolution
- key
- keysTitle
- layoutDirection
- localizeText
- location
- name
- noCheckDigits
- noDigits
- HidePageHeader OnLastPage
- HidePageFooterOnLastPage
- noText (Hide Text)
- pageName
- pageNoOffset
- pageNoFormat
- preferRectangularSymbols
- section
- referenceDefault
- referenceName
- scaleX
- scaleY
- seriesTitle
- smartParse
- strikethrough
- swapX
- text
- textAlignment
- textExpression
- thinToThickRelation
- thinToGapRelation

- tile - replaced by fill
- title
- trimText
- type (ObjectType)
- underline
- URL
- value
- valuesTitle
- visibilityCondition
- x
- xAxisTitle
- X-Size
- x-SizeAdjustment
- y
- yAxisTitle
- Y-Size
- y-SizeAdjustment

**alignment:** Controls the x position of this report element in its parent container, unless you have set the x property explicitly.

Type: Enum, the alignment choices are:

- **none** - there is no adjustment.
- **near** - shortcut for  $x = 0$ ; that is, aligns closest to the origin of x within the parent container
- **far** - shortcut for  $x = \text{max}$ ,  $\text{anchorX} = 1$ ; that is, aligns the most remotely from the origin of x within the parent container
- **center** - shortcut for  $x = \text{max}/2$ ,  $\text{anchorX} = 0.5$ , centered in the parent container
- **baseline** - uses baseline alignment

The default value is **none**.

See: Placing Elements on the Report Page, Align Numbers, Center Elements

**anchorX:** This property is relevant only if the property x is set. Shifts the attachment point for self-adjusting nodes between the point nearest to the parent's coordinate system's origin ( $\text{value}=0.0$ ) and the most remote point ( $\text{value}=1.0$ ). For nodes that are adjusted by their parent this attribute has no effect. A value of 0.5, for example, sets the attachment point to the center of the node.

Type: PXML, point value. The default value is 0.

**anchorY:** This property is relevant only if the property y is set. Shifts the attachment point for self-adjusting nodes between the point nearest to the parent's coordinate system's origin ( $\text{value}=0.0$ ) and the most remote point ( $\text{value}=1.0$ ). For nodes that are placed by their parent this attribute has no effect. A value of 0.5, for example, sets the attachment point to the center of the node.

Type: PXML, point value. The default value is 0.



**baselineType:** Provides additional information for **baseline** alignment, to specify which baseline of this report element should be linked to which baseline of a preceding element that also has the property alignment set to **baseline**. This property is relevant only if alignment for the report element is set to **baseline**.

Type Enum, choices are:

- **leftleft** - the report element and its preceding element will be aligned along their left baselines
- **leftright** - the left baseline of the report element will be aligned with the right baseline of the preceding element
- **rightleft** - the right baseline of the report element will be aligned with the left baseline of the preceding element
- **rightright** - the report element and its preceding element will be aligned along their right baselines

**bgColor:** Sets the background color for this node. The value is not inherited from the parent node.

Type: **Color**; valid colors are selected from the Edit Expression color palette. The default value is 'no background color': transparent.

**categoryKey:** Specifies the key of a category, in a Category Chart; must be unique within a chart.

Type: String. The default is a blank String.

**categoryTitle:** Specifies the title for the categories axis in a Category Chart.

Type: String. The default is a blank String.

**check:** When set, the checksum character of the specified code value is checked for correctness.

Type Boolean. The default value is **true**.

**clip:** Option to clip the report object box and its content along the sides. This applies to all layout nodes.

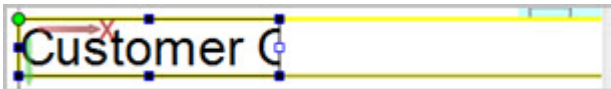


Figure 9-51. Clipped field

Type **Boolean**: The default value is **false**.

**codeType:** Specifies the type of Bar code. This is mandatory, a default value is not set. Not all codeTypes are relevant to all Bar Codes.

The currently supported types are: U PC-A, UPC-E, UPC Supplementals 2 and 5, EAN-13, EAN-8, EAN Supplementals 2 and 5, Code 128 (EAN 128), Code 2/5 Industrial, Code 2/5 Inverted, Code 2/5 IATA , Code 2/5 Interleaved, Code 2/5 Matrix, Code 2/5 Datalogic, Code BCD Matrix, Code 11 Matrix, Code 39, Code 39 extended, Code 32, Code 93, Code 93 extended, Codabar 18 and Codabar 2.

See Bar Codes for additional information.

Type: Enum.

**color:** Sets the paint color for this node and for all children of this node. On Map Charts this property is used to assign each slice of the chart a specific color.

Type: Color, valid colors are selected from the Edit Expressions color palette. The default value is inherited from the parent node. The root node has the color **black**.

**data:** Specifies the data of the image in **Base64** encoding.

There is no default value.

**drawAs:** Specifies the type of chart that is rendered from the data. This property also allows you to specify that the chart displays as a table.

Type: Enum.

Valid values for Category Chart: **Bar**, **Bar3D**, **Stacked Bar**, **Line**, **Line3D**, **Area**, **Stacked Area**, or **Waterfall**, Table, Sorted Table, Aggregated Table

**Note:** If you select **Waterfall** as the chart type, the value in the last category of the data set should be (redundantly) specified as the sum of the items in the preceding categories - otherwise, the final bar in the chart will be incorrectly plotted. At the present time, the chart can only have one category.

Valid values for Map Chart: **Pie**, **Pie3D**, **Bar**, **Bar3D** or **Ring**, Table, Sorted Table, Aggregated Table

Valid values for XY Chart: **Polar**, **Scatter**, **Area**, **Stacked Area**, **Line**, **Step**, **Step Area**, or **Time Series**, Table, Sorted Table

See also: Output Charts as Tables. See CategoryChart, MapChart, and XYChart for additional information about charts.

**drawLegend:** Controls whether the Legend is drawn for a MapChart. The option to remove the legend is useful when more than several charts are drawn next to each other in a document. You can make the charts share a single legend by specifying the legend only on one of the charts.

Type: Boolean. The default value is **true**.

**drawLabels:** Controls whether the Labels are drawn for a MapChart.

Type: Boolean. The default value is **true**.

**fidelity:** Controls the way text is output. When set, this property ensures that the preview and printout are 100% the same. In some cases this is necessary when the operating system font definitions deviate from the built-in font definitions in the printer. This flag then instructs the output routine not to use the printer font.

Type: Boolean. The default value is **false**.

**fill:** Replaces the previous property **tile**. When a document containing the tile property is opened in version 2.4x or higher, it is automatically replaced with this property.

This property is relevant when both x-Size and y-Size are set so that the outer image bounds are defined.

Type: Enum. The value can be one of:

- **completely** - The images is resized to fill the specified area. The aspect ratio is not respected.
- **preserveAspectRatio** - The images is resized to fit the specified area. The aspect ratio is respected. Setting the attributes X-Size\_Adjustment and Y-Size\_Adjustment to a value of "shrinkToChildren" will shrink the final bounds of the box so that it has the same size as the image.
- **tile** - The image is painted as tiles into the specified area.
- **clip** - The image is painted without scaling, and it is clipped at the edges of the specified area if it is too large to fit.

**floatingBehavior:** Controls the sizing behavior of the parent box when this node floats (sets x or y). This property is not relevant if x or y are not set.

The valid values are;

- **enclosed** - will make the parent size itself so that this object will be enclosed in the parent. All objects dragged from the toolbox or Data View have this property set to **enclosed**. Note that the node can still float outside of the parent using negative values for x or y. A
- **free** - will make the parent ignore this node during sizing, allowing it to float outside of its bounds.

Type: Enum. Valid values are **free** or **enclosed**.

**fontBold:** Sets a **bold** font style for this node and for all children of this node. The root node font style is **plain**.

Type: Boolean. The default value is inherited from the parent.

**fontItalic:** Sets an **italic** font style for this node and for all children of this node. The root node font style is **plain**.

Type: Boolean. The default value is inherited from parent.

**fontName:** Sets the font face name for this node and for all children of this node.

Type: Enum, platform-dependent. The default value is inherited from the parent. The root node has a platform-dependent Sans Serif' font.

See the Genero Studio >> Report Writer documentation topic "Configuring Fonts and Printers" for additional information about fonts.

**fontSize:** Sets the font size in points for this node and for all children of this node. During expression evaluation the variable **fontSize** is available, which contains the inherited font size. This variable can be used for relative font sizing. For example, the expression `fontSize="fontSize*1.2"` makes the current font 20% larger than the parent font.

Type: Numeric. The default value (not the expression) is inherited from the parent. The root node has a font size of 12 point.

**format:** Used to control the output of Numeric displays. For DECIMAL data types, *format-string* consists of pound signs (#) that represent digits and a decimal point. For example, "###.##" produces three places to the left of the decimal point and exactly two to the right.

See Align and Format Numbers. See the BDL **format** attribute for additional information and format characters.

Type: String.

**href:** This property can be used to define a hyperlink pointing to any resource on the Internet, local disk, or to any anchor inside the report document (see **id** for additional information about creating anchors). The **href** should be defined using the URI syntax. See Using Hyperlinks in a Report.

Type: String.

**id:** This property can be used to create an anchor in the report document. Nodes can be identified with a unique **id** and then used as the target of an href hyperlink. See Using Hyperlinks in a Report.

Type: String.

**indent:** Specifies the indentation value for the paragraph. The value may be negative.

Type: PXML. The default value is 0.

**intendedResolution:** Controls the mapping of pixels to device pixels. This property is relevant only if neither X-Size nor Y-Size are set, or if **tile** is set.

The default is the current screen resolution.

**key:** Specifies the key of the item in a chart. Within a category chart, specifies the key of an item within a category; must be unique.

Type: String.

**keysTitle:** Specifies the title of the keys Axis (usually the y Axis) of a Business Chart.

Type: String. The default is a blank String.

**layoutDirection:** Controls the direction in which child elements are laid out, which is also the direction of the Y-axis. Choices are:

- topToBottom
- leftToRight
- bottomToTop
- rightToLeft
- unturned
- turnRight
- upsideDown
- turnLeft
- inherit - the element inherits the orientation of its parent

- **swapped** - if the `swapX` property is also set to **swapped**, the element inherits the swapped orientation of its parent.

The following values depends on the language of the system where GRE is running; this allows you to have the `layoutDirection` follow the custom of the language.

- `horizontalNatural` - follows the custom of the system language
- `horizontalUnnatural` - reverses the natural order of the system language
- `verticalNatural` - follows the custom of the system language
- `verticalUnnatural` - reverses the natural order of the system language\*

For example, to have a horizontal layout that is `leftToRight` when the language is European, but `rightToLeft` for Arabic, select **horizontalNatural**. Select **horizontalUnnatural** to reverse the natural order of the language in the horizontal layout.

Type: Enum. The default value is **topToBottom**. By default the positive X-axis extends 90 degrees right (clockwise) of the positive Y-axis.

**localizeText:** Check the box to indicate that there is a localized string for this value, enabling the localization of text contents in Word Boxes and WordWrap Boxes.

Type: Boolean. Valid choices are True, False. The default value is False.

See the Genero Studio >> Report Writer documentation topic "Use Localized Strings" for more information.

**location:** Location of the file whose image is to be displayed: filename and path, or URL.

Type: String.

**name:** Assigns the specified name to the node for debugging purposes. The name must be unique in the document.

Type: String. The default value is "".

**HidePageHeaderOnLastPage:** Suppresses the drawing of `beforeFirst`, `firstPageHeader`, `evenPageHeader` and `oddPageHeader` material on the last page. Child elements querying for available space will, however, be given space values that are reduced by the size of the header as if it were drawn. Type: Boolean. The default value is **false**.

See also: Page Headers and Footers

**HidePageFooterOnLastPage:** Suppresses the drawing of `afterLast`, `firstPageTail`, `evenPageTail` and `oddPageTail` material on the last page. Child elements querying for available space will, however, be given space values that are reduced by the size of the footer as if it were drawn. Type: Boolean. The default value is **false**.

See also: Page Headers and Footers

**noText (Hide Text):** Suppresses text output if set.

Type: Boolean. The default value is **false**.

**pageName:** Name of a parent node.

Type: String. No default value is set.

**pageNoOffset:** An offset that is added to the current page number. When set to 100, for example, the first page will be number 101.

Type: Numeric. The default value is **0**.

**pageNoFormat:** Sets the number format type.

Type: Enum. Valid values are Arabic, Lowerroman, Upperroman. The default value is **Arabic**.

**referenceName:** The name of the InfoNode that is referenced.

Type: String. Mandatory; there is no default value.

**referenceDefault:** The text value to be displayed, if the reference cannot be resolved.

Type: String. The default value is **"-"**.

**(section):** This attribute of a Layout Node object specifies that the content of the node should print on its parent MiniPage at the location this property.

Type: Enum.

When you select a MiniPage for a page header or footer, for example, you specify where the container should be printed on its parent MiniPage by setting the section property. The report output prints any headers and footers that you have set, based on priorities of each of these values:

- **firstPageHeader** - the Page Header to be printed on the first page only; if this section is defined, subsequent Page Headers begin printing on the second page.
- **anyPageHeader** - the Page Header for every page, unless separate Odd and Even Page Headers are defined.
- **oddPageHeader, evenPageHeader** - specific Page Header to be printed on odd or even pages.
- **firstPageFooter** - a Page Footer that prints on the first page only; if this section is defined, subsequent page footers begin printing on the second page.
- **anyPageFooter** - the Page Footer for every page, unless separate Odd and Even Page Footers are defined.
- **oddPageFooter, evenPageFooter** - specific Page Footer to be printed on odd or even pages.
- **itemSeparator** - prints between each sibling element, as long as there is more than one element

If you have an anyPageHeader and a firstPageHeader, for example, the anyPageHeader content will print only on the second and subsequent pages.

**Important:** If you set the section property for any node, the node may not be preceded in its sibling list in the Report Structure tree by any nodes that do not have this property set.

See Print Headers and Footers.

**scaleX:** Applies the specified scale in this x-direction. The scale affects everything contained in this node (children, children-children, etc.). Scales are cumulative.

Type: PXML. The default value is **1.0**.

**scaleY:** Applies the specified scale in this y-direction. The scale affects everything contained in this node (children, children-children, etc.). Scales are cumulative.

Type: PXML. The default value is **1.0**.

**seriesTitle:** Title of the series in an XY Chart.

Type: String .

**smartParse:** Specifies that the codeValue property for Bar Code Boxes is parsed in "smart" mode. By default it is parsed in raw mode.

In "smart" mode the codeValue is interpreted literally. An attempt is made to map the characters contained in the string to one or more codes choosing the shortest possible representation. Control characters cannot be displayed in this mode. Currently the functionality is only available for "code-39-extended".

Type: Boolean. The default value is **false**.

**striketrough:** Specifies strikethrough for the text.

Type: Boolean. The default value is **false**.

**swapX:** By default the positive X-axis extends 90 degrees right (clockwise) of the positive Y-axis; if, for example, the Y-Axis points to north, the X axis will point eastward. Setting this value reverses the direction so that (in the example) the X-Axis would point to the west which is 90 degrees to the left (counter clockwise) of the Y-Axis.

Type: Boolean. The default value is **false**.

**text:** Specifies the text to be drawn. Occurrences of the newline character "\n" within the string cause line breaks.

Type: String. The default value is "".

For Word Boxes, WordWrap Boxes, and Decimal Format Boxes, the value of this property may be edited directly in the report design document instead. Double-click the object and the input cursor will be placed in the text. The layout of the document is updated on each keystroke.

For PageNo Boxes, BarCode Boxes, and Reference Boxes, the value of the text property specifies the maximum width.

**textAlignment:** Controls the horizontal alignment of text.

Type: Enum. Values are left, center, right, justified. The default value is **left**.

**textExpression:** Property of the PAGENOBOX, which expects a PXML string expression value to calculate a page number string. When set, the value of the

property overrides all other formatting relevant properties of the PAGENOBOX, although the text property is still used to set the width.

Type: PXML.

If a value for this property, or for the X-Size or text properties, is not set, a default length is used in calculating the page number string from these functions:

The following functions can be used to format and access specific page numbers and totals.

- Class String: `format(Numeric number, Enum format)` - formats the number as specified. The value for the format parameter can be ARABIC, LOWERROMAN or UPPERROMAN.
- Class Numeric: `getPhysicalPageNumber()` - gets the current page number of the physical page.
- Class Numeric: `getTotalNumberOfPhysicalPages()` - gets the total number of physical pages.
- Class Numeric: `getPageNumber(String pageName)` - gets the page number of the specified page
- Class Numeric: `getTotalNumberOfPages(String pageName)` - gets the total number of pages for the specified page.

See " Page N of M" examples.

**trimText:** Controls the trimming of spaces of the value of the text attribute.

Type: Enum. Values are both, compress, left, right. The default value is not set.

**title:** Specifies the metadata for the title of the report. In the case of SVG, the title property is used as a document caption in Genero Report Viewer.

Type: String.

**type (objectType):** The type of the report element; this is automatically set when you drop a specific element on the page.

**underline:** Specifies that the text is underlined.

Type: Boolean. The default value is **false**.

**value:** Specifies the value of the item.

Type: Numeric

**valuesTitle:** Specifies the Title of the values axis of a Business chart.

Type: String.

**URL:** Specifies the loading location or the name of the image to display.

Type: String. A value is mandatory; there is no default value.

**x:** Specifies the x-value of a X/Y coordinate pair defined by the element. The x value is an offset in the X-Size direction of the parent.



Type: Numeric. The default value is calculated during placing by the parent.

**xAxisTitle:** Specifies the title of the x Axis of a Business Chart.

Type: String.

**X-Size:** Gives the box a fixed dimension.

**Note:** If you want to preserve the aspect ratio of an image, set the value of either Y-Size or X-Size only, and allow Report Writer to calculate the corresponding value. If you set both properties, the resulting image will appear distorted.

Type: Numeric. The default value is calculated after the node has completed child alignment. The value is set to the smallest possible value that encloses all children without clipping any of them.

**X-Size Adjustment:** Specifies how the adjustment is to be made. A value of **shrinktoChildren** shrinks the X-Size as much as possible without clipping any of the children. A value of **expandToParent** causes the box to stretch as much as possible.

For objects that draw output with a defined size (Word Boxes, BarCode Boxes, Image Boxes where the tile property is set to FALSE), the value **shrinkToChildren** will not shrink the object below this size even if all children are smaller than this size or there are no children at all. Note that self-placing children are not considered.

Type: Enum. Choices are **shrinkToChildren**, **expandToParent**.

**y:** Specifies the y-value of a X/Y coordinate pair defined by the element. Changing the value adjusts the node at the specified coordinate. The coordinate value is an offset in the Y-Size direction of the parent.

Type: Numeric. The default value is calculated during placing by the parent.

**yAxisTitle:** Specifies the title of the y Axis of a Business Chart.

Type: String.

**Y-Size:** Gives the box a fixed dimension.

**Note:**

If you want to preserve the aspect ratio of an image, set the value of either Y-Size or X-Size, allowing Report Writer to calculate the corresponding value. If you set both properties, the resulting image will appear distorted.

Do not set a value for this property in WordWrapBoxes, because the element should typically grow based on its content.

Type: Numeric. The default value is calculated after the node has completed its child alignment. The value is set to the smallest possible value that encloses all children without clipping any of them.

**Y-Size Adjustment:** Specifies how the adjustment is to be made. A value of **shrinktoChildren** shrinks the length of y as much as possible without clipping any of the children. A value of **expandToParent** causes it to stretch as much as possible.

For objects that draw output with a defined size (Word Boxes, BarCode Boxes, Image Boxes where the tile property is set to FALSE), the value **shrinkToChildren** will not shrink the object below this size even if all children are smaller than this size or there are no children at all. Note that self placing-children are not considered.

Type: Enum. Choices are **shrinkToChildren**, **expandToParent**.

**codeValue:** Specifies the code value of a bar code. The character set and semantic constraints depend on the code type selected.

Type: String. See Bar Codes for complete information.

**codeValueExpression:** Property of the BARCODEBOX, which expects a PXML string expression value to calculate a page number string. When set, the value of the property overrides the codeValue property. The value of the **codeValue** property is still used for measuring the required space, but if it is not set the default size which is computed from the expression should be sufficiently exact.

Type: PXML.

If a value for this property, or for the X-Size property, is not set, a default length is used in calculating the page number string from these functions.

The following functions can be used to format and access specific page numbers and totals.

- Class String: `format(Numeric number, Enum format)` - formats the number as specified. The value for the format parameter can be ARABIC, LOWERROMAN or UPPERROMAN.
- Class Numeric: `getPhysicalPageNumber()` - gets the current page number of the physical page.
- Class Numeric: `getTotalNumberOfPhysicalPages()` - gets the total number of physical pages.
- Class Numeric: `getPageNumber(String pageName)`- gets the page number of the specified page
- Class Numeric: `getTotalNumberOfPages(String pageName)` - gets the total number of pages for the specified page.

See " Page N of M" examples.

**noDigits:** Controls the expected number of digits for codeValue for a code type that allows a variable number of digits. Specifically these are the code types "code-2-5-industrial", "code-2-5-inverted", "code-2-5-IATA", "code-2-5-interleaved", "code-2-5-matrix", "code-2-5-datalogic", "code-BCD-matrix", "code-11-matrix", "code-39", "code-39-extended", "codabar-18" and "codabar-2".

Type: Numeric.

**noCheckDigits:** Controls the expected number of check digits for codeValue for codes of type "code-11-matrix" and "code-93".

Type: Numeric.

**thinToThickRelation:** Controls the ratio of thin bars to thick bars. The value of a thick bar is calculated using the formula  $THICKBARWIDTH=THINBARWIDTH/$

**thinToThickRelation.** This parameter applies only to the code types "code-2-5-industrial", "code-2-5-inverted", "code-2-5-IATA", "code-2-5-interleaved", "code-2-5-matrix", "code-2-5-datalogic", "code-BCD-matrix", "code-11-matrix", "code-39", "code-39-extended", "code-32". and "codabar 2".

Type: Numeric. Default values: 1/3

**thinToGapRelation:** Controls the ratio of thin bars to the gaps between individual digits. The value of a gap is calculated by the formula  $GAPWIDTH = THINBARWIDTH / thinToGapRelation$ . This parameter applies only to the code types "code-2-5-industrial", "code-2-5-inverted", "code-2-5-IATA", "code-2-5-interleaved", "code-2-5-matrix", "code-2-5-datalogic", "code-BCD-matrix", "code-11-matrix", "code-39", "code-39-extended", "code-32", "codabar-18" and "codabar-2".

Type: Numeric. Default values: 0.5 for the types "code-2-5-industrial", "code-2-5-inverted" and "code-2-5-IATA"; 1 for the types "code-2-5-interleaved", "code-2-5-matrix", "code-2-5-datalogic" "code-BCD-matrix", "code-11-matrix", "code-39", "code-39-extended", "code-32", "code-93" and "code-93-extended".

**controlCharacters:** Configures which characters to use for textual printout of control characters in Code 93 and Code 93 extended. Code 93 defines four control characters "!? \ |" which are represented per default by the unicode characters "circled dash" '⊖' (229d), "circled asterisk operator" '⊛' (229b), "circled division slash" '⊘' (2298) and "circled plus" '⊕' (2295). Depending on the font used it might be desirable to used different characters than the default characters.

Type: String. Default value: "&#x229d;&#x229b;&#x2298;&#x2295;"

**visibilityCondition:** Boolean value (TRUE/FALSE) indicating whether the object is visible (not hidden).

Type: Boolean. The default is TRUE.

**preferRectangularSymbols:** The data-matrix bar code is usually quadratic, and any code value can be represented by a quadratic symbol. if you are concerned about running out of space in the vertical of the page, you might prefer a symbol that is wider than it is high. Check the box to enable rectangular symbols.

Type: Boolean. Valid choices are True, False. The default value is False.

## Properties related to margins and borders

- marginWidth
- marginRightWidth
- marginBottomWidth
- marginLeftWidth
- marginTopWidth
- borderWidth
- borderRightWidth
- borderBottomWidth
- borderLeftWidth
- borderTopWidth
- borderStyle
- borderRightStyle

- `borderBottomStyle`
- `borderLeftStyle`
- `borderTopStyle`
- `borderColor`
- `borderRightColor`
- `borderBottomColor`
- `borderLeftColor`
- `borderTopColor`
- `roundedCorners`
- `paddingWidth`
- `paddingRightWidth`
- `paddingBottomWidth`
- `paddingLeftWidth`
- `paddingTopWidth`

**marginWidth:** Sets the thickness of all an object's margins; can be overridden by specific Margin properties. See Design HowTo for illustrations.

Type: PXML, point value.

**marginRightWidth:** Sets the thickness of an object's right margin; overrides the property `marginWidth`. See Design HowTo for illustrations.

Type: PXML, point value.

**marginBottomWidth:** Sets the thickness of an object's bottom margin; overrides the property `marginWidth`. See Design HowTo for illustrations.

Type: PXML, point value.

**marginLeftWidth:** Sets the thickness of an object's left margin; overrides the property `marginWidth`. See Design HowTo for illustrations.

Type: PXML, point value.

**marginTopWidth:** Sets the thickness of an object's top margin; overrides the property `marginWidth`. See Design HowTo for illustrations.

Type: PXML, point value.

**borderWidth:** Sets the thickness of an object's borders; can be overridden by specific border properties. See Design HowTo for illustrations.

Type: PXML, point value. Default value is 2.

**borderRightWidth:** Sets the thickness of an object's right border; overrides the `borderWidth` property. See Design HowTo for illustrations.

Type: PXML, point value.

**borderBottomWidth:** Sets the thickness of an object's bottom border; overrides the `borderWidth` property. See Design HowTo for illustrations.

Type: PXML, point value.

**borderLeftWidth:** Sets the thickness of an object's left border; overrides the `borderWidth` property. See Design HowTo for illustrations.

Type: PXML, point value.

**borderTopWidth:** Sets the thickness of an object's top border; overrides the `borderWidth` property. See Design HowTo for illustrations.

Type: PXML, point value.

**borderStyle:** Sets the style for all an object's borders. Can be overridden by specific `borderStyle` properties. See Design HowTo for illustrations.

Type: Enum, choices are: none, solid, dotted, dashed, groove, ridge, inset, outset, double. Default is **none**.

**borderRightStyle:** Sets the style for an object's right border. Can override the `borderStyle` property. See Design HowTo for illustrations.

Type: Enum, choices are: solid, dotted, dashed, groove, ridge, inset, outset, double.

**borderBottomStyle:** Sets the style for an object's bottom border. Can override the `borderStyle` property. See Design HowTo for illustrations.

Type: Enum, choices are: solid, dotted, dashed, groove, ridge, inset, outset, double.

**borderLeftStyle:** Sets the style for an object's left border. Can override the `borderStyle` property. See Design HowTo for illustrations.

Type: Enum, choices are: solid, dotted, dashed, groove, ridge, inset, outset, double.

**borderTopStyle:** Sets the style for an object's top border. Can override the `borderStyle` property. See Design HowTo for illustrations.

Type: Enum, choices are: solid, dotted, dashed, groove, ridge, inset, outset, double.

**borderColor:** Sets the color of all an object's borders. Can be overridden by specific `borderColor` properties. See Design HowTo for illustrations.

Type: **Color**, valid colors are selected from the Edit Expression color palette.

**borderRightColor:** Sets the color of an object's right border. Can override the `borderColor` property. See Design HowTo for illustrations.

Type: **Color**, valid colors are selected from the Edit Expression color palette.

**borderBottomColor:** Sets the color of an object's bottom border. Can override the `borderColor` property. See Design HowTo for illustrations.

Type: **Color**, valid colors are selected from the Edit Expression color palette.

**borderLeftColor:** Sets the color of an object's left border. Can override the `borderColor` property. See Design HowTo for illustrations.

Type: **Color**, valid colors are selected from the Edit Expression color palette.

**borderTopColor:** Sets the color of an object's top border. Can override the **borderColor** property. See Design HowTo for illustrations.

Type: **Color**, valid colors are selected from the Edit Expression color palette.

**roundedCorners:** Specifies that the object's border corners will be round. This applies to the border styles **solid**, **dashed**, and **double** only. See Design HowTo for illustrations.

Type: Boolean. Valid choices are True, False. The default value is False.

**paddingWidth:** Sets the width of all of an object's padding. Can be overridden by specific padding properties. See Design HowTo for illustrations.

Type: PXML, point value.

**paddingRightWidth:** Sets the width of an object's right padding. Can override the **paddingWidth** property. See Design HowTo for illustrations.

Type: PXML, point value.

**paddingBottomWidth:** Sets the width of an object's bottom padding. Can override the **paddingWidth** property. See Design HowTo for illustrations.

Type: PXML, point value.

**paddingLeftWidth:** Sets the width of an object's left padding. Can override the **paddingWidth** property. See Design HowTo for illustrations.

Type: PXML, point value.

**paddingTopWidth:** Sets the width of an object's top padding. Can override the **paddingWidth** property. See Design HowTo for illustrations.

Type: PXML, point value.

## Properties for Report Document Metadata

Report Metadata properties can be set in the .Arp report design document. For compatibility reports, which have no .Arp document, API functions are provided. See Adding Report Document Metadata.

The metadata is inserted into the final document (e.g. PDF, SVG) if the format supports metadata.

- title
- author
- creator
- subject
- keywords

**title:** Specifies the title of the object.

Type: String.

**author:** Specifies the metadata for the author of the report.

Type: String.

**creator:** Specifies the metadata for the creator of the report.

Type: String.

**subject:** Specifies the metadata for the subject of the report.

Type: String.

**keywords:** Specifies the metadata for the keyword of the report.

Type: String.

## Reporting API Functions

Topics:

- General
- Mandatory report functions
  - fgl\_report\_loadCurrentSettings()
  - fgl\_report\_commitCurrentSettings()
  - fgl\_report\_loadAndCommit()
  - Usage
- Additional Functions to Change Default Report and Output options
  - Data
  - Form
  - Page
  - Output
  - Printer
- Additional functions to set Report Metadata for compatibility reports
- Additional functions to introspect reports at runtime (librdd)

### General

The Reporting API consists of the following files:

1. The **helpers.42m** file contains BDL functions that handle Genero Report Writer output and other features. The mandatory functions must be called in your report application. Other functions allow you to change the output format and output and printer features.
2. The following files contain some functions that are used internally by Genero Report Writer:
  - CaptionCustom.42m
  - CompatCustom.42m
  - EncodingCustom.42m
  - isotools.42m

The compiled versions of these files are provided in **<GRE-install-dir>\lib**.

**Important:** These compiled files are linked in a **libgre.42x** Genero library, which must be listed in the **external dependencies property** of any Genero Studio Project Manager application node that uses Genero Report Writer. This file should be listed by name only, without a path. Click the ... button in the Value column of the Properties View to open the Edit List window for that property.

## Error Handling

In case of an error, functions from this library will write error messages to stdout using the DISPLAY statement. Additionally the error condition is indicated by a return value.

## Verbosity level

Depending on the value of the environment variable "DEBUGLEVEL" Some of the functions issue warnings and other useful debugging information. Currently any value greater than zero will cause debug output. Debug information is also output using the DISPLAY statement.

## Mandatory functions - helpers.42m file

The following functions are required in the Report Driver section of the BDL file associated with a report. See Usage.

Table 9-10. Mandatory functions

| Function                                                                                                            | Description                                                                                                                                                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fgl_report_loadCurrentSettings</code><br>( <i>reportFileName</i> STRING) RETURNING <i>ok</i> INTEGER          | Function that sets the settings for the report to those specified in the current 4rp file (Configuration menu in the Report Design page); mandatory.<br><b>Note:</b> The <code>fgl_report_loadAndCommit</code> function may be used in place of the other mandatory functions if no changes to the default settings are required. |
| <code>fgl_report_commitCurrentSettings</code> ()<br>RETURNING <i>driver</i> om.SaxDocumentHandler                   | Function that configures the report engine; mandatory.<br><b>Note:</b> The <code>fgl_report_loadAndCommit</code> function may be used in place of the other mandatory functions if no changes to the default settings are required.                                                                                               |
| <code>fgl_report_loadAndCommit</code> ( <i>reportFileName</i> STRING) RETURNING <i>driver</i> om.SaxDocumentHandler | Function that loads a 4rp file and configures the report engine to execute this report.                                                                                                                                                                                                                                           |

### `fgl_report_loadCurrentSettings:`

FUNCTION `fgl_report_loadCurrentSettings` ( *reportFileName* STRING)  
RETURNING *ok* INTEGER

This function loads the specified 4rp file and configures the current in memory settings accordingly. Calling this function is a required prerequisite to calling `fgl_report_commitCurrentSettings()`.

### Parameters

- *reportFileName* STRING - Name of the 4rp report file to extract the settings information from. (The settings are made using the Configuration menu of the Report Design page. )

### Returns

- *ok* INTEGER - "true" if no error occurred.

See Usage.



### **fgl\_report\_commitCurrentSettings:**

FUNCTION fgl\_report\_commitCurrentSettings()  
RETURNING *driver* om.SaxDocumentHandler

This function configures the report engine based on the current settings in the .4rp file that have previously been loaded by a call to fgl\_report\_loadCurrentSettings(), and have possibly been modified.

It is mandatory to call fgl\_report\_commitCurrentSettings to set up the report engine, unless there are no changes to the default settings; in that case, fgl\_report\_loadAndCommit can be substituted for both of the functions.

### **Returns**

- *driver* om.SaxDocumentHandler - the created XMLhandler.

See Usage.

### **fgl\_report\_loadAndCommit:**

FUNCTION fgl\_report\_loadAndCommit ( *reportFileName* STRING)  
RETURNING *handler* om.SaxDocumentHandler

Function that loads a 4rp file and configures the report engine to execute this report. This function is a convenience function for:

```
call fgl_report_loadCurrentSettings(reportFileName)
call fgl_report_commitCurrentSettings()
```

Use this function instead when no change of settings is required.

### **Parameters**

- *reportFileName* STRING - The name of the 4rp report file to load. When running a legacy BDL report using Report Writer, the reportFileName should be Null. This indicates the ASCII output from the BDL file is to be written to the "SVG" device in preview mode.

### **Returns**

- *handler* om.SaxDocumentHandler - Null if an error occurred.

See Usage.

**Usage: load and commit:** The functions fgl\_report\_loadCurrentSettings and fgl\_report\_commitCurrentSettings work together to configure the report.

The call sequence is the following:

```
IF fgl_report_loadCurrentSettings(filename) THEN -- function returns TRUE
 -- if settings loaded OK
 -- optional functions to change settings would go here
 LET rephandler = fgl_report_commitCurrentSettings() -- function returns the
 -- SAXDocumentHandler
END IF
```

See The BDL file for complete examples.

These functions must appear in the Report Driver section of the BDL file associated with the report. If no settings are to be changed, the function fgl\_report\_loadAndCommit may be substituted for these two functions.

## Additional Functions to change default output options - helpers.42m file

These optional functions may be used to change the default settings for a report, and to configure output and printers.

Table 9-11. Data Functions

| Function                                                                                                                                          | Description                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fgl_report_createProcessLevelDataFile</code> ( <i>dataFileName</i> STRING) RETURNING <i>driver</i> om.SaxDocumentHandler                    | Function that configures the report execution to output data to an XML datafile.                                                                      |
| <code>fgl_report_setProcessLevelDataFile</code> ( <i>dataFileName</i> STRING)                                                                     | Function that configures the report execution to output an XML datafile in addition to the regular processing. New in 2.40                            |
| <code>fgl_report_runReportFromProcessLevelDataFile</code> ( <i>out</i> om.SaxDocumentHandler, <i>fileName</i> STRING) RETURNING <i>ok</i> INTEGER | Function that runs a report from an XML data file, previously created by a call to <code>fgl_report_createProcessLevelDataFile</code> .               |
| <code>fgl_report_runFromXML</code> ( <i>dataFileName</i> STRING) RETURNING <i>ok</i> INTEGER                                                      | Function that can run a report from an arbitrary XML data file that was not created by a call to <code>fgl_report_createProcessLevelDataFile</code> . |

Table 9-12. Form Functions

| Function                                                             | Description                                                                                                                 |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>fgl_report_findResourcePath</code> ( <i>reportname</i> STRING) | Function that facilitate loading of form files (.4rp) by searching directory lists specified in path environment variables. |

Table 9-13. Page Functions

| Function                                                                                                                                                                                                              | Description                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>fgl_report_setPageMargins</code> ( <i>topMargin</i> STRING, <i>bottomMargin</i> STRING, <i>leftMargin</i> STRING, <i>rightMargin</i> STRING)                                                                    | Function that configures the logical margins of a report page.                                                                          |
| <code>fgl_report_setPaperMargins</code> ( <i>topMargin</i> STRING, <i>bottomMargin</i> STRING, <i>leftMargin</i> STRING, <i>rightMargin</i> STRING)                                                                   | Function that configures the physical margins of a report page.                                                                         |
| <code>fgl_report_configureLabelOutput</code> ( <i>paperWidth</i> STRING, <i>paperHeight</i> STRING, <i>labelWidth</i> STRING, <i>labelHeight</i> STRING, <i>labelsPerRow</i> INTEGER, <i>labelsPerColumn</i> INTEGER) | Function that configures the physical layout of a label page.                                                                           |
| <code>fgl_report_configureMultipageOutput</code> ( <i>pageExponent</i> STRING, <i>isoNumber</i> INTEGER, <i>portrait</i> BOOLEAN)                                                                                     | Function to configure the multi page output for ISO or JIS formats.                                                                     |
| <code>fgl_report_selectLogicalPageMapping</code> ( <i>mapping</i> STRING)                                                                                                                                             | Function that configures the mapping from logical pages to physical pages.                                                              |
| <code>f gl_report_configurePageSize</code> ( <i>pageWidth</i> STRING, <i>pageHeight</i> STRING)                                                                                                                       | Function to configure the page size.                                                                                                    |
| <code>fgl_report_setPageSwappingThreshold</code> ( <i>value</i> INTEGER)                                                                                                                                              | Function that sets the threshold for page-to-disk swapping to prevent memory exhaustion on very large documents that use "Page N of M". |

Table 9-14. Output Functions

| Function                                                     | Description                           |
|--------------------------------------------------------------|---------------------------------------|
| <code>fgl_report_selectDevice</code> ( <i>device</i> STRING) | Function to select the output device. |

Table 9-14. Output Functions (continued)

| Function                                                                                                                                                                                                                                                                         | Description                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <code>fgl_report_selectPreview (preview INTEGER)</code>                                                                                                                                                                                                                          | Function to select preview mode.                                                                                     |
| <code>fgl_report_configurePDFDevice (fontDirectory STRING, antialiasFonts BOOLEAN, antialiasShapes BOOLEAN, monochrome BOOLEAN, fromPage INTEGER, toPage INTEGER)</code>                                                                                                         | Function to configure the PDF output.                                                                                |
| <code>fgl_report_configureSVGDevice (antialiasFonts BOOLEAN, antialiasShapes BOOLEAN, embedFonts BOOLEAN, charsetToEmbed STRING)</code>                                                                                                                                          | Function to configure the SVG output.                                                                                |
| <code>fgl_report_configureSVGPreview(preview STRING)</code>                                                                                                                                                                                                                      | Function to configure the SVG Preview                                                                                |
| <code>fgl_report_configureHTMLDevice(fromPage INTEGER, toPage INTEGER, embedImages INTEGER, imageGenerationDirectory STRING, imageURLPrefix STRING, removeWhitespace INTEGER, ignoreRowAlignment INTEGER, ignoreColumnAlignment INTEGER, removeBackgroundImages INTEGER )</code> | Function to configure HTML output                                                                                    |
| <code>fgl_report_configureXLSDevice (fromPage INTEGER, toPage INTEGER, removeWhitespace INTEGER, ignoreRowAlignment INTEGER, ignoreColumnAlignment INTEGER, removeBackgroundImages INTEGER, mergePages INTEGER)</code>                                                           | Function to configure Excel output in XLS format                                                                     |
| <code>f g_l_report_setXLSMergeCells (mergeCells BOOLEAN)</code>                                                                                                                                                                                                                  | Function to configure cell merging in XLS (Excel) output<br>New in 2.40                                              |
| <code>fgl_report_configureXLSXDevice(fromPage INTEGER, toPage INTEGER, removeWhitespace INTEGER, ignoreRowAlignment INTEGER, ignoreColumnAlignment INTEGER, removeBackgroundImages INTEGER, mergePages INTEGER)</code>                                                           | Function to configure Excel output in XLSX format<br>New in 2.40                                                     |
| <code>f g_l_report_setXLSXMergeCells (mergeCells BOOLEAN)</code>                                                                                                                                                                                                                 | Function to configure cell merging in XLSX (Excel) output<br>New in 2.40                                             |
| <code>fgl_report_configureRTFDevice (fromPage INTEGER, toPage INTEGER, imagesResolution INTEGER, imagesFormat STRING)</code>                                                                                                                                                     | Function to output in Microsoft RTF format<br>New in 2.40                                                            |
| <code>fgl_report_configureOORTFDevice (fromPage I INTEGER, toPage INTEGER, imagesResolution INTEGER, imagesFormat STRING)</code>                                                                                                                                                 | Function to configure RTF output for Open Office<br>New in 2.40                                                      |
| <code>f g_l_report_setRTFMemoryThreshold(memoryThreshold INTEGER)</code>                                                                                                                                                                                                         | Function to set the RTF memory threshold<br>New in 2.40                                                              |
| <code>fgl_report_configureImageDevice (antialiasFonts, BOOLEAN, antialiasShapes BOOLEAN, monochrome BOOLEAN, fromPage INTEGER, toPage INTEGER, fileType STRING, filePath STRING, fileNamePrefix STRING, resolution INTEGER)</code>                                               | Function to configure the image output                                                                               |
| <code>fgl_report_configureCompatibilityOutput (pageWidthInCharacters INTEGER, fontName STRING, fidelity BOOLEAN, reportName NULL, reportCategory STRING, systemIf STRING)</code>                                                                                                 | Function to configure the output for ASCII reports. Set the reportName parameter to NULL when calling this function. |
| <code>fgl_report_setAutoformatType (type STRING)</code>                                                                                                                                                                                                                          | Function that sets the auto-formatting type if no .4rp template is specified. New in 2.40                            |

Table 9-14. Output Functions (continued)

| Function                                                                                                                                                   | Description                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| f_gl_report_configureAutoformatOutput (fontName STRING, fontSize INTEGER, fidelity BOOLEAN, reportTitle STRING, fieldNamePatterns STRING, systemId STRING) | Function to configure the output when auto-formatting is enabled. New in 2.40                                                  |
| fgl_report_setCallbackLocalization(value INTEGER)                                                                                                          | Function that configures the report engine to call the function report_getFieldCaption(matchName, fieldName) for localization. |
| fgl_report_setOutputFileName (fileName STRING)                                                                                                             | Function to configure the file location of device output.                                                                      |
| fgl_report_setPDFJPEGIImageEncoding (encodeImagesAsJPEG STRING, jpegQuality BOOLEAN)                                                                       | Function to configure the encoding method of embedded images in PDF output.                                                    |
| fgl_report_setPDFImageResolution (imagesResolution INTEGER))                                                                                               | Function to configure the resolution of embedded images in PDF output.                                                         |
| fgl_report_setSharePortWithGDC (share BOOLEAN)                                                                                                             | Function that configures the report engine to use the same port as GDC for previewing.                                         |
| fgl_report_setSVGCompression (compressOutput BOOLEAN)                                                                                                      | Function to configure SVG compression.                                                                                         |
| fgl_report_setSVGCopies (copies INTEGER)                                                                                                                   | Function to specify the number of copies to be printed                                                                         |
| fgl_report_setSVGPageRange (fromPage INTEGER, toPage INTEGER)                                                                                              | Function to select which pages should be printed                                                                               |
| fgl_report_setSVGPaperSource (paperSource STRING)                                                                                                          | Function to select the input source of the printer                                                                             |
| fgl_report_setSVGPrinterName (printerName STRING)                                                                                                          | Function to select a specific printer by name                                                                                  |
| fgl_report_setSVGSheetCollate(sheetCollate STRING)                                                                                                         | Function to control the collation of multiple copies                                                                           |
| fgl_report_stopGraphicalCompatibilityMode()                                                                                                                | Function to restore text-based report output                                                                                   |

Table 9-15. Printer Functions

| Function                                                     | Description                                                                          |
|--------------------------------------------------------------|--------------------------------------------------------------------------------------|
| fgl_report_setPrinterChromaticity (chromaticity STRING)      | Function to control color selection of the printer.                                  |
| fgl_report_setPrinterCopies (copies INTEGER)                 | Function to specify the number of copies to be printed.                              |
| fgl_report_setPrinterDestinationUrl (destination STRING)     | Function to specify an alternate destination for the spooled printer formatted data. |
| fgl_report_setPrinterFidelity (fidelity INTEGER)             | Function to select printer high fidelity mode.                                       |
| fgl_report_setPrinterJobImpressions (jobImpressions INTEGER) | Function to specify the total size in number of impressions.                         |
| fgl_report_setPrinterJobMediaSheets (jobMediaSheets INTEGER) | Function to specify the total number of media sheets.                                |
| fgl_report_setPrinterJobName (jobName STRING)                | Function to specify a name for the job.                                              |
| fgl_report_setPrinterJobPriority (jobPriority INTEGER)       | Function to specify a priority for the job.                                          |
| fgl_report_setPrinterJobSheets (jobSheets STRING)            | Function to control job sheet printing.                                              |
| fgl_report_setPrinterMediaName (mediaTray STRING)            | Function to select the type of media to use.                                         |
| fgl_report_setPrinterMediaSizeName (mediaSizeName STRING)    | Function to select the media size to be used.                                        |
| fgl_report_setPrinterMediaTray (mediaTray STRING)            | Function to select the tray of the printer.                                          |
| fgl_report_setPrinterName (printerName STRING)               | Function to select a specific printer by name.                                       |
| fgl_report_setPrinterNumberUp (numberUp INTEGER)             | Function to specify a name for the job.                                              |

Table 9-15. Printer Functions (continued)

| Function                                                                                     | Description                                                           |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| <code>fgl_report_setPrinterOrientationRequested</code> ( <i>orientationRequested</i> STRING) | Function to control the paper orientation.                            |
| <code>fgl_report_setPrinterPageRanges</code> ( <i>pageRanges</i> STRING)                     | Function to specify the ranges of pages to print.                     |
| <code>fgl_report_setPrinterPrintQuality</code> ( <i>printQuality</i> STRING)                 | Function to control the quality used by the printer.                  |
| <code>fgl_report_setPrinterRequestingUserName</code> ( <i>requestingUserName</i> STRING)     | Function to specify end user's name.                                  |
| <code>fgl_report_setPrinterResolution</code> ( <i>resolution</i> INTEGER)                    | Function to specify an exact resolution.                              |
| <code>fgl_report_setPrinterSheetCollate</code> ( <i>sheetCollate</i> STRING)                 | Function to control the quality used by the printer.                  |
| <code>fgl_report_setPrinterSides</code> ( <i>sides</i> STRING)                               | Function to specify the mapping of pages on the physical media.       |
| <code>fgl_report_setPrinterWriteToFile</code> ( <i>file</i> STRING)                          | Function to specify a file where the report is written in postscript. |

**Usage: output functions:** For example, the following BDL code segment uses the `fgl_report_selectDevice` function to change the default output device to "PDF":

```
IF fgl_report_loadCurrentSettings(r_filename) THEN -- load the .4rp file
-- and continue if successful
 CALL fgl_report_selectDevice("PDF") -- change default
 LET handler = fgl_report_commitCurrentSettings() -- commit changes
ELSE
 EXIT PROGRAM
END IF
```

#### **fgl\_report\_configureCompatibilityOutput:**

```
FUNCTION fgl_report_configureCompatibilityOutput(
 pageWidthInCharacters INTEGER,
 fontName STRING,
 fidelity BOOLEAN,
 reportName NULL, reportCategory STRING,
 systemId STRING)
```

Function to optionally configure the output for BDL ASCII reports (compatibility reports) being run in graphical mode using Genero Report Writer.

This function is applicable when no .4rp template has been specified in the call to either `fgl_report_loadCurrentSettings` or `fgl_report_loadAndCommit`. All arguments to this function are optional (indicated by passing a null value).

#### **Parameters**

- *pageWidthInCharacters* INTEGER - For reports that do not contain a RIGHT MARGIN specification this value should be set. If the report does not contain a RIGHT MARGIN specification and this value is not set, a value of 132 is assumed.
- *fontName* STRING - Specifies the font to use. The default is a fixed pitch font.
- *fidelity* BOOLEAN - Ensures that the text preview and text printout are 100% the same. The font is not embedded in the report document, it is drawn similar to an image. As a result, you may not be able to select the text in the resulting report, depending on the output format chosen (pdf, for example).
- *reportName* - the value for this parameter is now provided internally. However, because of backwards compatibility, you must set the value to NULL when you call this function.

- *reportCategory* STRING - Specifies the "category" of the report. The value specified is passed in the overloadable callback function `compat_placePageBackground` in `CompatCustom.4gl`.
- *systemId* STRING- Specifies an absolute URL against which relative resources such as images in overlays are resolved.

See Usage

#### **fgl\_report\_configureImageDevice:**

```
FUNCTION fgl_report_configureImageDevice (antialiasFonts BOOLEAN,
antialiasShapes BOOLEAN,
monochrome BOOLEAN,
fromPage INTEGER,
toPage INTEGER,
fileType STRING,
filePath STRING,
fileNamePrefix STRING,
resolution INTEGER)
```

Function to configure the image output.

This function is applicable when image output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional (indicated by passing a null value).

#### **Parameters**

- *antialiasFonts* BOOLEAN - Configures whether fonts should be rendered using antialiasing. The default value is false.
- *antialiasShapes* BOOLEAN - Configures whether shapes should be rendered using antialiasing. The default value is false.
- *monochrome* BOOLEAN - Configures whether color values should be converted to monochrome output. The default value is false.
- *fromPage* INTEGER - Selects the lower bound of the range of pages to create images for. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to create images for. By default, images are created for all pages.
- *fileType* STRING - One of `jpg|png|bmp|gif`
- *filePath* STRING - Path of the target directory where the images are created
- *fileNamePrefix* STRING - Name prefix of the generated files (e.g. setting `namePrefix` to "Chart" will cause the creation of files called "Chart0001", "Chart0002", and so on.
- *resolution* INTEGER - Controls the resolution used for creating the images. If the image is later viewed unscaled on a device with the specified resolution, all items will have their correct metric length. Beware that high values may require enormous amounts of memory and the resulting files may become very large. The formula for calculating the memory consumption in bytes is  $\text{resolution\_in\_dpi\_x} * \text{page\_width\_in\_inches} * \text{resolution\_in\_dpi\_y} * \text{page\_height\_in\_inches} * 3\text{byte}$  for a color image. For a page of format "letter" at 96 DPI we therefore get  $96\text{DPI} * 8.5" * 96\text{DPI} * 11" * 3\text{byte} = 2.6 \text{ MB}$ . At 300 DPI this is 25 MB (color) and 8 MB (grayscale). The renderer currently requires the entire page to be in memory.

See Usage

#### **fgl\_report\_configureLabelOutput:**



```

FUNCTION fgl_report_configureLabelOutput(
 paperWidth STRING,
 paperHeight STRING,
 labelWidth STRING,
 labelHeight STRING,
 labelsPerRow INTEGER,
 labelsPerColumn INTEGER)

```

Function that configures the physical layout of a label page.

This function is applicable when selecting "label" as the mapping option by calling the function `fgl_report_selectLogicalPageMapping`. This figures the physical layout by specifying the paper dimensions, the physical label size and the n x m layout. The physical margins (distance between page border and label) are specified by calling `fgl_report_setPaperMargins`.

#### Parameters

- *paperWidth* STRING - width of the page (e.g. a4width)
- *paperHeight* STRING - height of the page (e.g. a4length)
- *labelWidth* STRING - Physical width of a label (e.g. 99.1mm) The value specified here should be the same or larger than the width in the .4rp file. The value is optional (indicated by passing null). In this case the width specified in the .4rp file is used.
- *labelHeight* STRING - Physical height of a label (e.g. 42.3mm) The value specified here should be the same or larger than the height in the .4rp file. The value is optional (indicated by passing null). In this case the height specified in the .4rp file is used.
- *labelsPerRow* INTEGER - the number of labels across
- *labelsPerColumn* INTEGER - the number of labels down

For additional information about the strings that can be used to specify the parameter values, see Dimension Resolver.

See Usage.

#### **fgl\_report\_configureMultipageOutput:**

```

FUNCTION fgl_report_configureMultipageOutput(
 pageExponent STRING,
 isoNumber INTEGER,
 portrait BOOLEAN)

```

Function to configure the multi page output for ISO or JIS formats

This function is applicable for ISO or JIS formats and enables the printing of several logical pages per physical page. The number of pages per page is always a power of 2.

#### Parameters

- *pageExponent* STRING - Specifies the number of pages to print. The actual number of pages is calculated by the formula  $\text{count} = 2^{\text{pageExponent}}$ .
- *isoNumber* INTEGER - Specifies the ISO number (e.g. 4 to ISOA4) This parameter is optional (indicated by passing an null value). If no value is specified, the page size of the logical page is taken from the .4rp file (if specified).
- *portrait* BOOLEAN - TRUE specifies that the page is in portrait orientation; FALSE= landscape. This parameter is optional (indicated by passing a null

value). If no value is specified, the value of the logical page is taken from the .4rp file (if specified), or from the value specified in a call to fgl\_report\_configurePageSize.

See Usage.

#### **fgl\_report\_configurePDFDevice:**

```
FUNCTION fgl_report_configurePDFDevice(
 fontDirectory STRING,
 antialiasFonts BOOLEAN,
 antialiasShapes BOOLEAN,
 monochrome BOOLEAN,
 fromPage INTEGER,
 toPage INTEGER)
```

Function to configure the PDF output

This function is applicable when PDF output has been selected by a call to the function fgl\_report\_selectDevice. All arguments to this function are optional (indicated by passing a null value). If the PDF document should be written to a file, the general function fgl\_report\_setOutputFileName and fgl\_report\_selectPreview are available for this purpose.

#### **Parameters**

- *fontDirectory* STRING - Absolute path to the directory containing the font files
- *antialiasFonts* BOOLEAN - Configures whether fonts should be rendered using antialiasing. The default value is false.
- *antialiasShapes* BOOLEAN - Configures whether shapes should be rendered using antialiasing. The default value is false.
- *monochrome* BOOLEAN - Configures whether color values should be converted to monochrome output. The default value is false.
- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the PDF document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the PDF document. By default all pages are included.

See Usage.

#### **fgl\_report\_configureSVGDevice:**

```
FUNCTION fgl_report_configureSVGDevice(
 antialiasFonts BOOLEAN,
 antialiasShapes BOOLEAN,
 embedFonts BOOLEAN,
 charsetToEmbed STRING)
```

Function to configure the SVG output

This function is applicable when SVG output has been selected by a call to the function fgl\_report\_selectDevice. All arguments to this function are optional (indicated by passing a null value). If the SVG document should be written to a file, the general function fgl\_report\_setOutputFileName and fgl\_report\_selectPreview are available for this purpose.

#### **Parameters**

- *antialiasFonts* BOOLEAN - Configures whether fonts should be rendered using antialiasing. The default value is false.



- *antialiasShapes* BOOLEAN - Configures whether shapes should be rendered using antialiasing. The default value is false.
- *embedFonts* BOOLEAN - Controls whether fonts are embedded within the document. The default value is true.
- *charsetToEmbed* STRING - SVG offers the possibility to embed fonts within the document. Documents with embedded fonts are slightly larger, but they offer the advantage of exact reproduction. Normally one would embed exactly those characters that were used throughout the document, but that would require reading the entire document before creating any SVG output. This attribute provides a solution that does not compromise streaming. The characters in the specified character set are embedded, not requiring all of them to have been used. Note that the character sets are neither restricted to 255 characters, nor is there any restriction on which unicode characters are used. Furthermore, an entry in the character set can be a sequence of unicode characters, thus allowing for ligatures. Valid values are:
  - **DEFAULT** - embeds all characters from the code pages iso-8859-1 through iso-8859-10 (about 600 characters).
  - **ISO-8859-15** - embeds all characters from the named code page (about 200 characters).
  - **ALL** - embeds all characters available in the selected fonts. This option should only be used when Asian characters are needed, since the size of the glyph definitions will significantly grow the document size.

See Usage.

#### **fgl\_report\_configureSVGPreview:**

FUNCTION **fgl\_report\_configureSVGPreview** (  
     *preview* STRING)

Function to select SVG preview mode or other print options

This function is applicable only for the device SVG when previewing is selected.

#### **Parameter**

- *preview* STRING - one of:  
 "Preview"|"ShowPrintDialog"|"PrintOnDefaultPrinter"|"PrintOnNamedPrinter"
  - The Preview option shows the report in a folder tab of the main preview window.
  - The ShowPrintDialog option pops up the system print dialog allowing the user to select and configure a printer. If confirmed, the document is printed in the background and the previewer main window is not shown.
  - The PrintOnDefaultPrinter option prints the report silently on the default printer. The previewer main window is not shown.
  - The PrintOnNamedPrinter option prints the report silently on the named printer. The previewer main window is not shown. In order to provide the name and other parameters, the following functions have been added:  
 fgl\_report\_setSVGCopies, fgl\_report\_setSVGPageRange,  
 fgl\_report\_setSVGPaperSource, fgl\_report\_setSVGPrinterName,  
 fgl\_report\_setSVGSheetCollate.

See Usage.

#### **fgl\_report\_configureXLSDevice:**

```

FUNCTION fgl_report_configureXLSDevice (
 fromPage INTEGER,
 toPage INTEGER,
 removeWhitespace INTEGER,
 ignoreRowAlignment INTEGER,
 ignoreColumnAlignment INTEGER,
 removeBackgroundImages INTEGER,
 mergePages INTEGER)

```

Function to configure the XLS (Excel) output

This function is applicable when XLS output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional, indicated by passing a null value. If the XLS document should be written to a file, the general functions `fgl_report_setOutputFileName` and `fgl_report_selectPreview` are available for this purpose.

#### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the XLS document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the XLS document. By default all pages are included.
- *removeWhitespace* INTEGER - Controls whether or not cells should be created for empty strings. By default whitespace is stripped from the document.
- *ignoreRowAlignment* INTEGER - When set, only those objects that are entirely above or entirely below each other will go in separate rows. When set, the option reduces the amount of rows thereby losing the horizontal alignment. The placement is not changed so that stacked items remain stacked. By default row alignment is ignored.
- *ignoreColumnAlignment* INTEGER - When set, only those objects that are entirely to the left or entirely to the right of each other will go in separate columns. When set, the option reduces the amount of columns thereby losing the vertical alignment. The placement is not changed so that adjacent items remain adjacent. By default column alignment is ignored.
- *removeBackgroundImages* INTEGER - Controls the behavior in case an IMAGEBOX is partially obscured by another element. When set, the image is removed from the resulting document otherwise the handling is as with any other case of overlapping items. By default, background images are removed.
- *mergePages* INTEGER - Controls the behavior when the report has more than one page. By default a separate sheet is created per page. Setting this parameter causes the pages to be merged, creating a single result sheet unless the sheet has more than 65536 rows; in that case, the exceeding rows spill over into extra sheets. Setting this parameter and using a standard page size is the recommended way to produce single-sheet output; using a huge custom page size instead can adversely affect memory reclamation and performance.

See also Sending Data to an Excel spreadsheet.

#### **fgl\_report\_configureXLSXDevice:**

```

FUNCTION fgl_report_configureXLSXDevice (
 fromPage INTEGER,
 toPage INTEGER,
 removeWhitespace INTEGER,
 ignoreRowAlignment INTEGER,
 ignoreColumnAlignment INTEGER,
 removeBackgroundImages INTEGER,
 mergePages INTEGER)

```

Function to configure Excel output in XLSX format. The functionality is identical to the existing "XLS" output except for the following:

- The format can be opened only by newer versions of Microsoft Excel (beginning with 2007). It is possible to download a backward compatibility pack from Microsoft that will allow opening of these files with older versions. Beware that in this case worksheets containing more than 65536 rows will be truncated at this limit.
- The new format overcomes the 65536 row limit of the "XLS" format. This is the main motivation for introducing this format.
- The new format is generated with constant memory consumption so that very large documents can be produced without exhausting heap space.

This function is applicable when "XLSX" output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional, indicated by passing a null value. If the XLSX document should be written to a file, the general functions `fgl_report_setOutputFileName` and `fgl_report_selectPreview` are available for this purpose.

#### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the XLS document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the XLS document. By default all pages are included.
- *removeWhitespace* INTEGER - Controls whether or not cells should be created for empty strings. By default whitespace is stripped from the document.
- *ignoreRowAlignment* INTEGER - When set, only those objects that are entirely above or entirely below each other will go in separate rows. When set, the option reduces the amount of rows thereby losing the horizontal alignment. The placement is not changed so that stacked items remain stacked. By default row alignment is ignored.
- *ignoreColumnAlignment* INTEGER - When set, only those objects that are entirely to the left or entirely to the right of each other will go in separate columns. When set, the option reduces the amount of columns thereby losing the vertical alignment. The placement is not changed so that adjacent items remain adjacent. By default column alignment is ignored.
- *removeBackgroundImages* INTEGER - Controls the behavior in case an IMAGEBOX is partially obscured by another element. When set, the image is removed from the resulting document otherwise the handling is as with any other case of overlapping items. By default, background images are removed.
- *mergePages* INTEGER - Controls the behavior when the report has more than one page.

See also Sending Data to an Excel spreadsheet.

#### **`fgl_report_configureHTMLDevice:`**

```
FUNCTION fgl_report_configureHTMLDevice (
 fromPage INTEGER,
 toPage INTEGER,
 embedImages INTEGER,
 imageGenerationDirectory STRING,
 imageURLPrefix STRING, removeWhitespace INTEGER,
 ignoreRowAlignment INTEGER,
 ignoreColumnAlignment INTEGER,
 removeBackgroundImages INTEGER)
```

## Function to configure the HTML output

This function is applicable when HTML output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional (indicated by passing a null value). If the HTML document should be written to a file, the general functions `fgl_report_setOutputFileName` and `fgl_report_selectPreview` are available for this purpose.

### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the HTML document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the HTML document. By default all pages are included.
- *embedImages* INTEGER - Specifies whether to embed images in the resulting HTML output. By default this is not the case.
- *imageGenerationDirectory* STRING - If images are not embedded, this property specifies the directory into which generated images are written. The directory needs to exist, it is not created. Note that the urls that are created will not take this value into account. By default, the urls that are created contain the image name only, so that the images are expected to reside in the same directory as the document. If needed, the url prefix for the generated urls can be changed with the property `imageURLPrefix`.
- *imageURLPrefix* STRING - If images are not embedded, this property specifies the prefix of the urls of the generated images. As an example consider that an image of the name "12345.png" is created, and that this property is set to the value `"/images/";` then the "src" attribute of the generated "img" element would be set to `"/images/12345.png"`.
- *removeWhitespace* INTEGER - Controls whether cells should be created for empty strings. By default whitespace is stripped from the document.
- *ignoreRowAlignment* INTEGER - When set, only those objects that are entirely above or entirely below each other will go in separate rows. When set, the option reduces the amount of rows, thereby losing the horizontal alignment. The placement is not changed so that stacked items remain stacked. By default row alignment is not ignored.
- *ignoreColumnAlignment* INTEGER - When set, only those objects that are entirely to the left or entirely to the right of each other will go in separate columns. When set, the option reduces the amount of columns, thereby losing the vertical alignment. The placement is not changed so that adjacent items remain adjacent. By default column alignment is not ignored.
- *removeBackgroundImages* INTEGER - Controls the behavior when an IMAGEBOX is partially obscured by another element. When set, the image is removed from the resulting document; otherwise, the handling is as in any other case of overlapping items. By default, background images are removed.

See Usage.

### **fgl\_report\_configureRTFDevice:**

```
FUNCTION fgl_report_configureRTFDevice (
 fromPage INTEGER,
 toPage INTEGER,
 imagesResolution INTEGER,
 imagesFormat STRING)
```

Function to configure RTF output

This function is applicable when RTF output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional (indicated by passing a null value).

If the RTF document should be written to a file, the general functions `fgl_report_setOutputFileName` and `fgl_report_selectPreview` are available for this purpose.

#### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the RTF document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the RTF document. Per default all pages are included.
- *imagesResolution* INTEGER - Specifies the resolution of embedded images. In addition to ImageBoxes, content from BarCodeBoxes, business charts and HTMLBoxes are embedded as images.
- *imagesFormat* STRING - One of : "png", "jpg". Controls the format of images embedded in the RTF document. Select jpg for compactness, png for lossless compression.

#### **fgl\_report\_configureOORTFDevice:**

```
FUNCTION fgl_report_configureOORTFDevice (
 fromPage INTEGER,
 toPage INTEGER,
 imagesResolution INTEGER,
 imagesFormat STRING)
```

Function to configure RTF output for Open Office

This function is applicable when RTF output has been selected by a call to the function `fgl_report_selectDevice`. All arguments to this function are optional (indicated by passing a null value).

If the RTF document should be written to a file, the general function `fgl_report_setOutputFileName` and `fgl_report_selectPreview` are available for this purpose.

#### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to include in the RTF document. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to include in the RTF document. Per default all pages are included.
- *imagesResolution* INTEGER - Specifies the resolution of embedded images. In addition to ImageBoxes, content from BarCodeBoxes, business charts and HTMLBoxes are embedded as images.
- *imagesFormat* STRING - One of : "png", "jpg". Controls the format of images embedded in the RTF document. Select jpg for compactness, png for lossless compression.

#### **fgl\_report\_setRTFMemoryThreshold:**

```
FUNCTION fgl_report_setRTFMemoryThreshold (
 memoryThreshold INTEGER)
```

Function to set the RTF memory threshold

In order to prevent exhaustion of main memory when processing large documents, the processor can be instructed to swap parts of the document to a temporary disk file when the document size exceeds this threshold. The default value is set to 10% of the total available heap space. The default heap space is 64Mb.

#### Parameters

- *memoryThreshold* INTEGER - The threshold in bytes above which documents are swapped to disk.

#### **fgl\_report\_createProcessLevelDataFile:**

```
FUNCTION fgl_report_createProcessLevelDataFile(
 dataFileName STRING)
RETURNING driver om.SaxDocumentHandler
```

Function that configures the report execution to output an XML datafile

This function configures the execution to output data to the specified file. Calling this function will configure the report engine to output the data file only. You can use the function `fgl_report_runReportFromProcessLevelDataFile` to run a report getting the data from this XML file.

#### Parameters

- *dataFileName* STRING - Name of the data file to generate

#### Returns

- *driver* om.SaxDocumentHandler if no error occurred.

See Usage.

#### **fgl\_report\_setProcessLevelDataFile:**

```
FUNCTION fgl_report_setProcessLevelDataFile (
 dataFileName STRING)
```

Function that configures the report execution to output an XML datafile in addition to the regular processing

Like the function `fgl_report_createProcessLevelDataFile`, this function causes a data file to be produced that can be used for archiving and/or reformatting, using the function `fgl_report_runReportFromProcessLevelDataFile`. However, this function causes the file to be created in addition to the regular processing; for example, a PDF file and a data file can be created at the same time. Or, you could create a completely different layout for one of the files by specifying a different .Arp template.

#### Parameters

- *dataFileName* STRING - Name of the data file to generate

See: Usage

#### **fgl\_report\_runReportFromProcessLevelDataFile:**

```
FUNCTION fgl_report_runReportFromProcessLevelDataFile(
 out om.SaxDocumentHandler,
 fileName STRING)
RETURNING ok INTEGER
```

Function that runs a report from a process level file, previously created by a call to `fgl_report_createProcessLevelDataFile`.

This function will replay the report from the file thereby replacing the running of the report (START REPORT, OUTPUT TO REPORT, FINISH REPORT statements).

#### Parameters

- *out* `om.SaxDocumentHandler` - A output pipe obtained by a call to `fgl_report_commitCurrentSettings` or a similar function
- *fileName* `STRING` - A file name of an XML file previously created by a call to `fgl_report_createProcessLevelDataFile()`

#### Returns

- *ok* `INTEGER` - "true" if no error occurred.

See Usage.

#### **fgl\_report\_runFromXML:**

```
FUNCTION fgl_report_runFromXML (
 dataFileName STRING)
RETURNING ok INTEGER
```

Function that can run a report using as the data source an arbitrary XML file - a file that was not created by a Genero program using `fgl_report_createProcessLevelDataFile` .

This function will replay the report from the file, replacing the running of the report (START REPORT, OUTPUT TO REPORT, FINISH REPORT statements).

The function sets up the report engine based on the current settings that have previously been loaded by a call to `fgl_report_loadCurrentSettings()`, and may have been modified by calls to `fgl_report_selectDevice()` or `fgl_report_selectPreview()`. The function automatically calls `fgl_report_commitCurrentSettings()`.

See Data From an XML File for an example program.

#### Parameters

- *datafileName* `STRING` - file name of an XML file containing the data

#### Returns

- *ok* `INTEGER` - "true" if no error occurred.

#### **fgl\_report\_selectDevice:**

```
FUNCTION fgl_report_selectDevice(
 device STRING)
```

Function to select the output device

Selecting a different output device changes the current settings. The choices are listed in Parameters.

The following device-specific configuration functions are also available:

- **PDF:** `fgl_report_configurePDFDevice`,
- **Image:** `fgl_report_configureImageDevice`,



- **SVG:** fgl\_report\_configureSVGDevice,
- **HTML:** fgl\_report\_configureHTMLDevice.
- **Excel output (XLS or SLSX):** fgl\_report\_configureXLSDevice or fgl\_report\_configureXLSXDevice.
- **RTF output: (RTF or OORTF)** fgl\_report\_configureRTFDevice (Microsoft ), fgl\_report\_configureOORTFDevice (Open Office)
- **Printer:** Setting this parameter value selects server-side silent printing. Report Writer selects the printer that best meets the criteria specified by the functions listed in the Printer Functions section.

#### Parameters

- *device* STRING - One of the following:
  - Printer
  - PDF
  - RTF (*Microsoft RTF*)
  - OORTF (*Open Office RTF*)
  - Image
  - SVG
  - XLS
  - XLSX
  - HTML

See Usage.

#### fgl\_report\_selectPreview:

FUNCTION **fgl\_report\_selectPreview**(  
    *preview* INTEGER)

Function to select preview mode

Available only for the devices PDF and SVG, this function sets the output to be shown in the previewer. In the case of PDF, this requires Acrobat Reader to be installed on the client. In the case of SVG, the Genero Report Viewer is a part of the Genero Desktop Client.

#### Parameters

- *preview* INTEGER - 0 = FALSE, 1 = TRUE

See Usage.

#### fgl\_report\_selectLogicalPageMapping:

FUNCTION **fgl\_report\_selectLogicalPageMapping**(  
    *mapping* STRING)

Function that configures the mapping from logical pages (page dimensions in the .4rp file) to physical pages (actual page dimensions on the device).

Table 9-16. Mapping options

| Mapping Option | Description                                                  |
|----------------|--------------------------------------------------------------|
| oneToOne       | The logical and physical pages are identical. Default value. |



Table 9-16. Mapping options (continued)

| Mapping Option | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| labels         | For label printing it is advisable to design one label rather than creating a fixed n x m layout. Such a layout can be printed on an arbitrary physical n x m layout as long as the physical labels have at least the size of the logical labels. The physical dimensions of the layout must be configured using the function <code>fgl_report_configureLabelOutput</code> and <code>fgl_report_setPaperMargins</code> . The margins within a label can be configured by calling <code>fgl_report_setPageMargins</code> . |
| multipage      | For ISO and JIS sized pages it is possible to print several pages per page. This is achieved by setting the mapping to this value and calling the function <code>fgl_report_configureMultipageOutput</code> to configure the required number of pages.                                                                                                                                                                                                                                                                    |

### Parameters

- *mapping* STRING - one of `oneToOne`, `labels`, or `multipage`

See Usage.

### **fgl\_report\_configurePageSize:**

```
FUNCTION fgl_report_configurePageSize (
 pageWidth STRING,
 pageHeight STRING)
```

The page dimensions of the report are read from the `.Arp` file. This function is used to override the values found there.

For BDL ASCII reports being run in graphical mode (compatibility reports, no `.Arp` file is being used), this function is used to set values for these parameters:

### Parameters

- *pageWidth* STRING - width of the page (e.g. `a4width`)
- *pageHeight* STRING - height of the page (e.g. `a4length`)

See `DimensionResolver` for additional examples of the strings that can be used in these parameters.

See Usage.

### **fgl\_report\_setCallbackLocalization:**

```
FUNCTION fgl_report_setCallbackLocalization (share BOOLEAN)
```

By default the field titles in a report are retrieved from Genero localization files compiled by the command `"fglmkstr"`. This function configures the report engine to call the function `report_getFieldCaption(matchName, fieldName)` instead, and doesn't use the localization file.

### Parameters

- *share* BOOLEAN - when set, the report engine will call the GRE library function `report_getFieldCaption` for each field, to retrieve the field caption.

See Localized strings.

#### **fgl\_report\_setOutputFileName:**

```
FUNCTION fgl_report_setOutputFileName(fileName STRING)
```

Function to configure the file location of device output.

This function is applicable when file output has been selected by a call to `fgl_report_selectPreview`.

#### **Parameters**

- *fileName* STRING - Pathname of the file. The name may contain a device-specific suffix. If no suffix is specified then a device-specific suffix is appended.

See Usage.

#### **fgl\_report\_setPageMargins:**

```
FUNCTION fgl_report_setPageMargins(
 topMargin STRING,
 bottomMargin STRING,
 leftMargin STRING,
 rightMargin STRING)
```

Function that configures the logical margins of a report

The logical margins of the report are read from the `.4rp` file. This function is used to override the values found there.

For BDL ASCII reports (compatibility reports, no `.4rp` file is specified), this function is used to set the values of the parameters:

#### **Parameters**

- *topMargin* STRING - top margin value of the logical page (e.g. 0.5cm)
- *bottomMargin* STRING - bottom margin value of the logical page (e.g. 0.5cm)
- *leftMargin* STRING - left margin value of the logical page (e.g. 0.5cm)
- *rightMargin* STRING - right margin value of the logical page (e.g. 0.5cm)

For additional information about the units that can be used to specify the margin values, see Dimension Resolver.

In the case of label printing (see `fgl_report_selectLogicalPageMapping`), this function specifies the margins within a label. Similarly, when multi page output is selected (see `fgl_report_selectLogicalPageMapping`) the values specify the margins of the logical pages which can be smaller than the physical margins since the required width is not limited by device limitations but by aesthetic aspects only.

See Usage.

#### **fgl\_report\_setPaperMargins:**

```

FUNCTION fgl_report_setPaperMargins(
 topMargin STRING,
 bottomMargin STRING,
 leftMargin STRING,
 rightMargin STRING)

```

Function that configures the physical margins of a report

The physical margins of the report can be set by this function for the case that either label printing or multi page output has been selected by a call to `fgl_report_selectLogicalPageMapping`.

#### Parameters

- *topMargin* STRING - top margin value of the physical page (e.g. 0.5cm)
- *bottomMargin* STRING - bottom margin value of the physical page (e.g. 0.5cm)
- *leftMargin* STRING - left margin value of the physical page (e.g. 0.5cm)
- *rightMargin* STRING - right margin value of the physical page (e.g. 0.5cm)

For additional information about the units that can be used to specify the margin values, see Dimension Resolver.

See Usage.

#### **fgl\_report\_setPrinterChromaticity:**

```

FUNCTION fgl_report_setPrinterChromaticity(
 chromaticity STRING)

```

Function to control color selection of the printer

Controls how the print data should be generated or processed. It does not confine the printer selection to printers with the specified capability. Default value is color. Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *chromaticity* STRING - one of: monochrome|color

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterCopies:**

```

FUNCTION fgl_report_setPrinterCopies(
 copies INTEGER)

```

Function to specify the number of copies to be printed.

Specifies the number of copies to be printed. Default value: 1

Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *copies* INTEGER- the number of copies to print

See Usage.

**fgl\_report\_setPrinterDestinationUrl:**

```
FUNCTION fgl_report_setPrinterDestinationUrl(
 destination STRING)
```

Function to specify an alternate destination for the spooled printer formatted data.

A URL indicating an alternate destination for the spooled printer formatted data. Some print services will not support the notion of a destination other than the printer device and so will not support this attribute. A common use for this attribute will be applications which want to redirect output to a local disk file: e.g. "file:out.prn". A more platform independent way is to set the `fgl_report_setPrinterName` to "stdout" so that postscript is written to stdout or to specify the `fgl_report_setPrinterwriteToFile` attribute which causes postscript to be written to the specified file. Another alternative may be the use of the `fgl_report_selectDevice` to generate a PDF file. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

**Parameters**

- *destination* STRING - the destination URL

See Usage.

**fgl\_report\_setPrinterFidelity:**

```
FUNCTION fgl_report_setPrinterFidelity(
 fidelity INTEGER)
```

Function to select printer high fidelity mode

When set, all attributes of the printer have to meet the requested values, otherwise the printout will fail. When set to false, a reasonable attempt to print the document is acceptable. Default value is false. Setting this option reduces the set of usable printers to those matching it.

**Parameters**

- *fidelity* INTEGER - true or false

See Usage.

**fgl\_report\_setPrinterJobImpressions:**

```
FUNCTION fgl_report_setPrinterJobImpressions(
 jobImpressions INTEGER)
```

Function to specify the total size in number of impressions.

Specifies the total size in number of impressions of the document. An "impression" is the image (possibly many print stream pages in different configurations) imposed onto a single media page. The `jobImpressions` attribute describes the size of the job. This attribute is not intended to be a counter; it is intended to be useful routing and scheduling information. The printer may try to compute the attribute's value if it is not supplied. Even if a value is supplied, the printer may choose to change the value if the printer is able to compute a value that is more accurate than the supplied value. The printer may be able to determine the correct value for the `jobImpressions` attribute right at job submission or at a later time. Unlike the `fgl_report_setPrinterJobMediaSheets` function, the value must not include the

multiplicative factors contributed by the number of copies, specified by the `fgl_report_setPrinterCopies` function. Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *jobImpressions* INTEGER - number of jobs

See Usage.

#### **fgl\_report\_setPrinterJobMediaSheets:**

FUNCTION **fgl\_report\_setPrinterJobMediaSheets**(  
*jobMediaSheets* INTEGER)

Function to specify the total number of media sheets.

Specifies the total number of media sheets to be produced for this job. The `jobMediaSheets` attribute describes the size of the job. This attribute is not intended to be a counter; it is intended to be useful routing and scheduling information. The printer may try to compute the attribute's value if it is not supplied. Even if a value is supplied, the printer may choose to change the value if the printer is able to compute a value that is more accurate than the supplied value. The printer may be able to determine the correct value for the `jobMediaSheets` attribute right at job submission or at a later time. The value must include the multiplicative factors contributed by the number of copies, specified by the `fgl_report_setPrinterCopies` function. Default value: not set Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *jobMediaSheets* INTEGER - number of sheets

See Usage.

#### **fgl\_report\_setPrinterJobName:**

FUNCTION **fgl\_report\_setPrinterJobName**(  
*jobName* STRING)

Function to specify a name for the job.

Name of the print job useful for tracking the job. The value does not have to be unique. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *jobName* STRING- name of the job

See Usage.

#### **fgl\_report\_setPrinterJobPriority:**

FUNCTION **fgl\_report\_setPrinterJobPriority**(  
*jobPriority* INTEGER)

Function to specify a priority for the job.

If supplied, the value specifies a priority for scheduling the job. A higher value specifies a higher priority. The value 1 indicates the lowest possible priority. The

value 100 indicates the highest possible priority. Among those jobs that are ready to print, a printer must print all jobs with a priority value of *n* before printing those with a priority value of *n*-1 for all *n*. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *jobPriority* INTEGER - priority of the job

#### Fails if

- argument is not between 1 and 100

See Usage.

#### **fgl\_report\_setPrinterJobSheets:**

FUNCTION **fgl\_report\_setPrinterJobSheets**(  
    *jobSheets* STRING)

Function to control job sheet printing

Controls if job start and end sheets are to be printed. Default: none. Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *jobSheets* STRING - one of: none | standard

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterMediaName:**

FUNCTION **fgl\_report\_setPrinterMediaName**(  
    *mediaTray* STRING)

Function to select the type of media to use

Controls the type of media to choose. This function and the functions `fgl_report_setPrinterMediaSizeName` and `fgl_report_setPrinterMediaTray` are mutually exclusive. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *mediaTray* STRING - one of: iso-a4-transparent | iso-a4-white | na-letter-transparent

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterMediaSizeName:**

FUNCTION **fgl\_report\_setPrinterMediaSizeName**(  
    *mediaSizeName* STRING)

Function to select the media size to be used

Selects the media size to be used. Normally this does not need to be specified, as the program will automatically select the smallest media onto which the current document can be printed without clipping or scaling. This function and the functions `fgl_report_setPrinterMediaTray` and `fgl_report_setPrinterMediaName` are mutually exclusive. Default value not set (is automatically selected). Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *mediaSizeName* STRING - one of: a|b|c|d|e|executive|folio|invoice|iso-a0|iso-a1|iso-a2|iso-a3|iso-a4|iso-a5|iso-a6|iso-a7|iso-a8|iso-a10|iso-b0|iso-b1|iso-b2|iso-b3|iso-b4|iso-b5|iso-b6|iso-b7|iso-b8|iso-b10|iso-c0|iso-c1|iso-c2|iso-c3|iso-c4|iso-c5|iso-c6|iso-designated-long|iso-italian-envelope|iso-oufuko-postcard|jis-b0|jis-b1|jis-b2|jis-b3|jis-b4|jis-b5|jis-b6|jis-a7|jis-b8|jis-b10|ledger|monarch-envelope|na-10x13-envelope|na-10x14-envelope|na-10x15-envelope|na-5x7|na-6x9-envelope|na-8x10|na-9x11-envelope|na-9x12-envelope|na-legal|na-letter|na-number-10-envelope|na-number-11-envelope|na-number-12-envelope|na-number-14-envelope|na-number-9-envelope|personal-envelope|quarto|tabloid

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterMediaTray:**

FUNCTION **fgl\_report\_setPrinterMediaTray**( *mediaTray* STRING)

Function to select the tray of the printer

Controls what tray to take the media from. This function and the functions `fgl_report_setPrintermediaTray` and `fgl_report_setPrintermediaName` are mutually exclusive. Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *mediaTray* STRING - one of: bottom|envelope|large-capacity|main|manual|middle|side|top

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterName:**

FUNCTION **fgl\_report\_setPrinterName**(  
  *printerName* STRING)

Function to select a specific printer by name

Name of requested printer listed as "SERVICE". A special meaning is attached to the name "stdout". If this value is specified, then postscript is written to stdout. Default value is "Not Set". Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *printerName* STRING - the name of the printer

See Usage.

#### **fgl\_report\_setPrinterNumberUp:**

FUNCTION **fgl\_report\_setPrinterNumberUp**(  
    *numberUp* INTEGER)

Function to specify the number of print stream pages for a single side of an instance.

Specifies the number of print stream pages to impose upon a single side of an instance of selected medium. That is, if the *numberUp* value is *n*, the printer must place *n* print stream pages on a single side of an instance of the selected medium. To accomplish this, the printer may add some sort of translation, scaling or rotation.

**Note:** since this feature is available only on a few printers it is advisable to perform the necessary transformations using this API Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### **Parameters**

- *numberUp* INTEGER - number of pages

See Usage.

#### **fgl\_report\_setPrinterOrientationRequested:**

FUNCTION **fgl\_report\_setPrinterOrientationRequested**(  
    *orientationRequested* STRING)

Function to control the paper orientation

Controls the paper orientation. Normally this value should not be set. The style sets the value internally by analyzing the page's width and height values. Default: not set Setting this option reduces the set of usable printers to those matching it.

#### **Parameters**

- *orientationRequested* STRING - one of: landscape | portrait | reverse-landscape | reverse-portrait

#### **Fails if**

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterPageRanges:**

FUNCTION **fgl\_report\_setPrinterPageRanges**(  
    *pageRanges* STRING)

Function to specify the ranges of pages to print.

Specifies the ranges of print stream pages that the printer uses for each copy of the document printed. Nothing is printed for any pages identified that do not exist in the document. Default: "not set" (everything is printed) Setting this option reduces the set of usable printers to those matching it.

#### **Parameters**



- *pageRanges* STRING - page ranges following the syntax [1-9][0-9]\*-[1-9][0-9]\*([1-9][0-9]\*-[1-9][0-9]\*)\*

#### **Fails if**

- argument does not follow syntax

See Usage.

#### **fgl\_report\_setPrinterPrintQuality:**

FUNCTION **fgl\_report\_setPrinterPrintQuality**(  
    *printQuality* STRING)

Function to control the quality used by the printer

Specifies the print quality used by the printer. Default: not set Setting this option reduces the set of usable printers to those matching it.

#### **Parameters**

- *printQuality* STRING - one of: draft|high|normal

#### **Fails if**

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterRequestingUserName:**

FUNCTION **fgl\_report\_setPrinterRequestingUserName**(  
    *requestingUserName* STRING)

Function to specify end user's name.

Specifies the end user's name who submitted the print job. A requesting user name is an arbitrary string defined by the client. The printer does not put the client-specific requestingUserName attribute into the print job's attribute set; rather, the printer puts in a jobOriginatingUserName attribute. This means that services which support specifying a username with this attribute should also report a jobOriginatingUserName in the job's attribute set. Note that many print services may have a way to independently authenticate the user name, and so may state support for a requesting user name, but in practice will then report the user name authenticated by the service rather than that specified by this attribute. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### **Parameters**

- *requestingUserName* STRING - the user name

See Usage.

#### **fgl\_report\_setPrinterResolution:**

FUNCTION **fgl\_report\_setPrinterResolution**(  
    *resolution* INTEGER)

Function to specify an exact resolution.

Specifies an exact resolution supported by a printer or to be used for the job. This attribute assumes that printers have a small set of device resolutions at which they

can operate rather than a continuum. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *resolution* INTEGER - the resolution of the printer

See Usage.

#### **fgl\_report\_setPrinterSheetCollate:**

```
FUNCTION fgl_report_setPrinterSheetCollate(
 sheetCollate STRING)
```

Function to control the quality used by the printer

Specifies whether the pages of the document are to be in sequence, when multiple copies of the document are specified by the `fgl_report_setPrinterCopies` function. When `sheetCollate` is "collated", each copy of the document is printed with the pages in sequence. When `sheetCollate` is "uncollated", each page is printed a number of times equal to the value of the `fgl_report_setPrinterCopies` attribute in succession. For example, suppose a document produces two pages as output, `fgl_report_setPrinterCopies` is 6, and `sheetCollate` is "uncollated"; in this case six copies of the first page are printed followed by six copies of the second page.

**Note:** It is discouraged to set `sheetCollate` to "collated" since it requires caching the entire document which is undesirable for large documents. If necessary, produce the document the required number of times. Default: uncollated Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *sheetCollate* STRING - one of: collated | uncollated

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterSides:**

```
FUNCTION fgl_report_setPrinterSides(
 sides STRING)
```

Function to specify the mapping of pages on the physical media.

Specifies how print-stream pages are to be imposed upon the sides of an instance of a selected medium, i.e., an impression.

- one-sided: Imposes each consecutive print-stream page upon the same side of consecutive media sheets.
- two-sided-short-edge: Imposes each consecutive pair of print stream pages upon front and back sides of consecutive media sheets, such that the orientation of each pair of print stream pages on the medium would be correct as if for binding along the short edge.
- two-sided-long-edge: Imposes each consecutive pair of print stream pages upon front and back sides of consecutive media sheets, such that the orientation of each pair of print stream pages on the medium would be correct as if for binding along the long edge.

Default value: not set Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *sides* STRING - One of: one-sided | two-sided-short-edge | two-sided-long-edge

#### Fails if

- argument is not one of the specified values

See Usage.

#### **fgl\_report\_setPrinterWriteToFile:**

FUNCTION **fgl\_report\_setPrinterWriteToFile**( *file* STRING)

Function to specify a file where the report is written in postscript.

A file name specifying a location where the report output is written in postscript format. If this attribute is set, all other IPP attribute values are ignored. Default: "not set" Setting this option reduces the set of usable printers to those matching it.

#### Parameters

- *file* STRING- the destination file

#### Fails if

- File cannot be opened for writing

See Usage.

#### **fgl\_report\_setSharePortWithGDC:**

FUNCTION **fgl\_report\_setSharePortWithGDC** ( *share* BOOLEAN)

Function that configures the report engine to use the same port as GDC for previewing.

By default the port is shared and the port value specified in the FGLSERVER environment variable is used.

#### Parameters

- *share* BOOLEAN - If set, the engine will use FGLSERVER as client port.

See Usage.

#### **fgl\_report\_setSVGCompression:**

FUNCTION **fgl\_report\_setSVGCompression** ( *compressOutput* BOOLEAN)

Function to configure SVG compression.

This function is applicable when SVG output has been selected by a call to the function `fgl_report_selectDevice`. When set, the function causes SVG to be written in compressed form, causing files and streams to be shrunk to about a tenth of their original size. This can benefit the overall performance on slow networks.

The report viewer (GRV) automatically detects if files or streams are compressed and decompresses them on the fly if necessary.

### Parameters

- *compressOutput* BOOLEAN- Configures whether the output should be compressed. The default value is false.

See Usage.

### **fgl\_report\_setSVGCopies:**

```
FUNCTION fgl_report_setSVGCopies (copies INTEGER)
```

Function to specify the number of copies to be printed.

Specifies the number of copies to be printed. Default value: 1

The function applies only if the option **PrintOnNamedPrinter** is chosen in a call to `fgl_report_configureSVGPreview()`.

### Parameters

- *copies* INTEGER - the number of copies to print

See Usage.

### **fgl\_report\_setSVGPageRange:**

```
FUNCTION fgl_report_setSVGPageRange (fromPage INTEGER,
 toPage INTEGER)
```

Function to select which pages should be printed.

The function applies only if the option **PrintOnNamedPrinter** is chosen in a call to `fgl_report_configureSVGPreview()`.

### Parameters

- *fromPage* INTEGER - Selects the lower bound of the range of pages to print. The default value is 1.
- *toPage* INTEGER - Selects the upper bound of the range of pages to create print. By default, all pages are printed.

See Usage.

### **fgl\_report\_setSVGPaperSource:**

```
FUNCTION fgl_report_setSVGPaperSource (paperSource STRING)
```

Function to select the input source of the printer. Controls what source to take the paper from.

The function applies only if the option **PrintOnNamedPrinter** is chosen in a call to `fgl_report_configureSVGPreview()`. Furthermore, the functionality is available only on Windows.

### Parameters

- *paperSource* STRING- one of: Auto, Cassette, Envelope, EnvelopeManual, FormSource, LargeCapacity, LargeFormat, Lower, Middle, Manual, OnlyOne, Tractor, SmallForm

See Usage.

**fgl\_report\_setSVGPrinterName:**

```
FUNCTION fgl_report_setSVGPrinterName (printerName STRING)
```

Function to select a specific printer by name.

The function applies only if the option **PrintOnNamedPrinter** is chosen in a call to `fgl_report_configureSVGPreview()`.

**Parameters**

- *printerName* STRING - the name of the printer

See Usage.

**fgl\_report\_setSVGSheetCollate:**

```
FUNCTION fgl_report_setSVGSheetCollate (sheetCollate STRING)
```

Function to control the collation of multiple copies.

Specifies whether the pages of the document are to be in sequence, when multiple copies of the document are specified by the `fgl_report_setSVGCopies` function.

If *sheetCollate* is:

- **collated**, each copy of the document is printed with the pages in sequence.
- **uncollated**, each page is printed a number of times equal to the value of the `fgl_report_setSVGCopies` attribute in succession. This is the default.

For example, if a document produces two pages as output, `fgl_report_setSVGCopies` is 6, and *sheetCollate* is "uncollated", then six copies of the first page are printed followed by six copies of the second page.

**Note:** We discourage setting *sheetCollate* to "collated" since it requires caching the entire document, which is undesirable for large documents. If necessary, produce the document the required number of times.

This function applies only if the option **PrintOnNamedPrinter** is chosen in a call to `fgl_report_configureSVGPreview()`.

**Parameters**

- *sheetCollate* STRING - one of collated | uncollated

Fails If:

- argument is not one of the specified values

See Usage.

**fgl\_report\_stopGraphicalCompatibilityMode:**

```
FUNCTION fgl_report_stopGraphicalCompatibilityMode
```

Function to restore text-based report output.

The graphical compatibility mode switched on by calling `fgl_report_loadCurrentSettings(NULL)` is switched off using this function.

See Usage.

**fgl\_report\_setPageSwappingThreshold:**

```
FUNCTION fgl_report_setPageSwappingThreshold (value INTEGER)
```

Function that sets the threshold for page-to-disk swapping to prevent memory exhaustion on very large documents that use " Page N of M"

The function specifies the maximum number of pages that may be held in main memory. When the value is exceeded, pages are swapped to the disk. This parameter is only needed for very large report that contain references to the total number of pages. alternatively it is possible to grow the amount of memory that the Java JVM may allocate by setting the parameter -Xmx (e.g. "java -Xmx512m" for 512 MB) in the script \$GREDIR/bin/greportwriter[.bat]. By default the value is not set so that pages are never swapped to disk.

**Parameters**

- *value* INTEGER - A positive integer greater than zero that specifies the maximum number of pages that may be held in main memory.

See Usage.

**fgl\_report\_setPDFJPEGImageEncoding:**

```
FUNCTION fgl_report_setPDFJPEGImageEncoding (encodeImagesAsJPEG STRING,
 jpegQuality BOOLEAN)
```

Function to configure the encoding method of embedded images in PDF output

Optional JPEG image encoding can be configured to shrink the size of PDF documents using this function. This function is applicable when PDF output has been selected by a call to the function fgl\_report\_selectDevice. All arguments to this function are optional (indicated by passing a null value).

**Parameters**

- *encodeImagesAsJPEG* STRING - Specifies images to be encoded in JPEG format. The default value is False.
- *jpegQuality* BOOLEAN - Sets the compression quality to a value between 0 and 1. By default a value of 1 is used (highest quality).

**fgl\_report\_setPDFImageResolution:**

```
FUNCTION fgl_report_setPDFImageResolution (imagesResolution INTEGER)
```

Function to configure the resolution of embedded images in PDF output

This function is applicable when PDF output has been selected by a call to the function fgl\_report\_selectDevice.

**Parameters**

- *imagesResolution* INTEGER- Specifies the maximum resolution of embedded images.

**fgl\_report\_setXLSMergeCells:**

```
FUNCTION fgl_report_setXLSMergeCells (mergeCells BOOLEAN)
```

Function to configure cell merging in XLS (Excel) output

This function is applicable when XLS output has been selected by a call to the function `fgl_report_selectDevice`.

#### Parameters

- *mergeCells* BOOLEAN - Controls the behavior when an item of the report occupies more than one cell. Setting this value causes the cells to be merged into one cell. The value is TRUE by default.

#### **fgl\_report\_setXLSXMergeCells:**

```
FUNCTION fgl_report_setXLSXMergeCells (
 mergeCells BOOLEAN)
```

Function to configure cell merging in XLSX (Excel) output.

This function is applicable when XLSX output has been selected by a call to the function `fgl_report_selectDevice`.

#### Parameters

- *mergeCells* BOOLEAN - Controls the behavior when an item of the report occupies more than one cell. Setting this value causes the cells to be merged into one cell. The value is TRUE by default.

#### **fgl\_report\_setAutoformatType:**

```
FUNCTION fgl_report_setAutoformatType (type STRING)
```

Function that sets the auto-formatting type if no .4rp template is specified. By default such a report would be rendered in compatibility mode; this function provides additional rendering options:

- A compatibility report - the report will be output in ASCII format, as in legacy reports
- A flat list - this is a simple list design, similar to the output from the List Report template available from the Studio main menu option File>>New, Reports, Report Designs. This design is particularly well suited to produce Excel output from arbitrary reports.

#### Parameters

- *type* STRING - One of "COMPATIBILITY", "FLAT LIST"

#### **fgl\_report\_configureAutoformatOutput:**

```
FUNCTION fgl_report_configureAutoformatOutput (
 fontName STRING,
 fontSize INTEGER,
 fidelity BOOLEAN,
 reportTitle STRING,
 fieldNamePatterns STRING,
 systemId STRING)
```

Function to configure the output when auto-formatting is enabled.

This function is applicable when no .4rp template has been specified in the call to either `fgl_report_loadCurrentSettings` or `fgl_report_loadAndCommit`, and auto-formatting with a value other than COMPATIBILITY has been selected by a call to `fgl_report_setAutoformatType`. All arguments to this function are optional (indicated by passing a null value).

#### Parameters

- *fontName* STRING - Specifies the font to use.
- *fontSize* INTEGER - Specifies the font size to use
- *fidelity* BOOLEAN - Specifies whether or not to set the fidelity property for the produced WORDBOX objects. See WORDBOX for more information.
- *reportTitle* STRING - Title of the report.
- *fieldNamePatterns* STRING - A comma separated list of field name patterns; fields not matching any of the patterns are not printed. The patterns may contain literal characters, the? question mark, the \* star character, and character ranges, as defined for the BDL MATCHES operator.
- *systemId* STRING - Specifies an absolute URL against which relative resources such as images in overlays are resolved.

#### **fgl\_report\_findResourcePath:**

```
FUNCTION fgl_report_findResourcePath (
 reportName STRING)
```

Function that returns the path to a resource searching first FGLRESOURCEPATH, then FGLDBPATH and finally DBPATH.

The function returns the absolute path to the resource specified in the argument. In case that a file cannot be found the function returns the argument passed.

#### **Parameters**

- *reportName* STRING - Specify the name of the report.

#### **Usage**

The function is typically used in conjunction with the function `fgl_report_loadCurrentSettings()` as in the following example:

```
IF fgl_report_loadCurrentSettings(fgl_report_findResourcePath("OrderReport.4rp"))
THEN
 ...
IF
```

Assuming FGLRESOURCEPATH is set to "/home/fjs/reports/lists:/home/fjs/correspondence" and the report 'OrderReport.4rp' is located at /home/fjs/correspondence/OrderReport.4rp", that report will be loaded.

## **Functions to set Report Metadata for Compatibility Reports New in 2.40**

Table 9-17. Functions to set Report Metadata for Compatibility Reports

| Function                                                      | Description                                         |
|---------------------------------------------------------------|-----------------------------------------------------|
| <code>fgl_report_setAuthor</code> ( <i>author</i> STRING)     | Function to set the report author meta data value   |
| <code>fgl_report_setTitle</code> ( <i>title</i> STRING)       | Function to set the report title                    |
| <code>fgl_report_setCreator</code> ( <i>creator</i> STRING)   | Function to set the report creator meta data value  |
| <code>fgl_report_setSubject</code> ( <i>subject</i> STRING)   | Function to set the report subject meta data value  |
| <code>fgl_report_setKeywords</code> ( <i>keywords</i> STRING) | Function to set the report keywords meta data value |



**Important:** These Report Metadata functions are intended to be used for compatibility reports only. For regular reports the Metadata values, select the root node of the report in the Report Designer Properties View, and set its Metadata properties.

If the target file format supports meta data, then the value is inserted into the target document. Typically this is a whitespace-separated list of key words. Calling this function to set the value supersedes the value specified in the **.4rp** template.

## Functions to introspect reports at runtime (librdd) New in 2.40

The module librdd (part of the standard reporting library libgre.42x) provides programmatic access to the structures contained in ".rdd" files (files produced by the 4GL compiler from 4GL sources with the option `--build-rdd`).

Table 9-18. Functions to introspect reports at runtime

| Function                                                                                                          | Description                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rdd_getEveryRowFields (</code><br><i>rddFileName</i> STRING,<br><i>reportName</i> STRING)<br><code>)</code> | Function that loads a rdd file and returns the field names of the fields contained in the first PRINT statement from the ON EVERY ROW section of the specified report. |
| <code>rdd_loadRddAndGetReportInfos (</code><br><i>rddFileName</i> STRING)<br><code>)</code>                       | Function that loads a rdd file and returns a data structure describing the reports contained in this file.                                                             |
| <code>rdd_debugReportInfos (</code><br><i>reportInfos</i> DYNAMIC ARRAY of rddReportInfo)<br><code>)</code>       | Function that displays the report information contained in a rdd file for debugging purposes.                                                                          |

The source of the module is located in `$GREDIR/src/api/librdd.4gl`.

### **rdd\_getEveryRowFields:**

```
FUNCTION rdd_getEveryRowFields (
 rddFileName STRING,
 reportName STRING)
RETURNING fieldNames DYNAMIC ARRAY OF STRING
```

Function that loads a rdd file and returns the field names of the fields contained in the first PRINT statement from the ON EVERY ROW section of the specified report.

Files of type "rdd" files are produced from 4GL source files by the 4GL compiler using the option `--build-rdd`.

### **Parameters**

- *rddFileName* STRING - Full path to a .rdd file.
- *reportName* STRING - Name of the report as specified in the 4GL source file.

### **Returns**

- *fieldNames* DYNAMIC ARRAY OF STRING - Null if error occurred.

### **rdd\_loadRddAndGetReportInfos:**

```
FUNCTION rdd_loadRddAndGetReportInfo (
 rddFileName STRING)
RETURNING reportInfos DYNAMIC ARRAY OF rddReportInfo
```

Function that loads a rdd file and returns a data structure describing the reports contained in this file.

Files of type "rdd" files are produced from 4GL source files by the 4GL compiler using the option "--build-rdd".

#### Parameters

- *rddFileName* STRING - Full path to a .rdd file.

#### Returns

- *reportInfos* DYNAMIC ARRAY OF rddReportInfo - Null if error occurred.

#### rdd\_debugReportInfos:

```
FUNCTION rdd_debugReportInfos (
 reportInfos DYNAMIC ARRAY OF rddReportInfo)
```

Function that displays the report information contained in a rdd file for debugging purposes.

The function displays debugging information for the data structure returned by a call to the function rdd\_loadRddAndGetReportInfos.

#### Parameters

- *reportInfos* DYNAMIC ARRAY OF rddReportInfo  
- Structure obtained by a call to rdd\_loadRddAndGetReportInfos.

**Types used in librdd:** These data types are used in the librdd module.

#### rddReportPrintElementInfo

```
TYPE rddReportPrintElementInfo RECORD
 unionType CHAR(1), #v=variable, e=expression, l=literal
 value STRING, #the value in case of type 'l'
 variableName STRING, #the variable name case of type 'v'
 variableType STRING #the variable or expression type in case of type 'v'
END RECORD
```

#### rddReportPrintInfo

```
TYPE rddReportPrintInfo RECORD
 controlPath DYNAMIC ARRAY OF rddTree, # Path into tree, e.g.
 # REPORT/ON EVERY ROW/IF/THEN/FOR,
 # REPORT/ON EVERY ROW/IF/ELSE/FOR
 printxName STRING,
 printElements DYNAMIC ARRAY OF rddReportPrintElementInfo
END RECORD
```

#### rddReportSectionInfo

```
TYPE rddReportSectionInfo RECORD
 name STRING, # FIRST PAGE HEADER, BEFORE GROUP customer_id, ON EVERY ROW, ...
 reportPrints DYNAMIC ARRAY OF rddReportPrintInfo
END RECORD
```

#### rddReportInfo

```
TYPE rddReportInfo RECORD
 reportName STRING,
 reportSections DYNAMIC ARRAY OF rddReportSectionInfo
END RECORD
```

## rddTree

```
TYPE rddTree RECORD
 firstLine INTEGER,
 lastLine INTEGER,
 type CHAR(1), # l: loop: FOR, FOREACH, WHILE
 # c: conditional: CASE, IF, OTHERWISE, WHEN
 # i: interaction: MENU, {DISPLAY|INPUT} ARRAY, INPUT, PROMPT,
 # ON {KEY|ACTION}, {BEFORE|AFTER} ..
 # d: declaration: GLOBALS, DEFINE, RECORD
 # o: other statements: TRY, CATCH
 # a: trigger in interaction or report
 fileId INTEGER, # references rddFiles (rddFile.id=fileId)
 fileIdEnd INTEGER, # references rddFiles (rddFile.id=fileIdEnd) =
 # the file where the declaration ends
 name STRING # for example BEFORE GROUP
END RECORD
```

## Bar Codes

The report element container for a Bar Code is a Bar Code Box. This flow object draws bar codes. The following properties are specific to Barcode Box:

codeType, codeValue, fidelity, noText, codeValue, check, noDigits, noCheckDigits, thinToThickRelation, thinToGapRelation, controlCharacters, CodeValueExpression

Additional properties are inherited from Layout Node.

Some properties that are specific to a particular bar code are shown in the code's listing.

## Barcode Types

Table 9-19. Barcode Types

| Code Type            | Number of Digits Supported | Normal size                  |
|----------------------|----------------------------|------------------------------|
| ean-13               | 13                         | 37.29mm x 26.26mm (w x h)    |
| ean-8                | 8                          | 26.73mm x 21.64 mm (w x h)   |
| ean-supplemental-2   | 2                          | 6.6mm x 26.26mm (w x h)      |
| ean-supplemental-5   | 5                          | 15.5mm x 26.26mm (w x h)     |
| upc-a                | 12                         | 1.469in x 1.020in (w x h)    |
| upc-e                | 8                          | 0.897in x 1.020in (w x h)    |
| upc-supplemental-2   | 2                          | 0.26in x 1.02in (w x h)      |
| upc-supplemental-5   | 5                          | (0.611in x 1.02in w x h)     |
| code-128             | varies                     | (calculated width x 6.5mm h) |
| code-2-5-industrial  | varies                     | (calculated width x 1in h)   |
| code-2-5-inverted    | varies                     | (calculated width x 1in h)   |
| code-2-5-IATA        | varies                     | (calculated width x 1in h)   |
| code-2-5-interleaved | varies                     | (calculated width x 1in h)   |
| code-2-5-matrix      | varies                     | (calculated width x 1in h)   |
| code-2-5-datalogic   | varies                     | (calculated width x 1in h)   |
| code-BCD-matrix      | varies                     | (calculated width x 1in h)   |
| code-11-matrix       | varies                     | (calculated width x 1in h)   |
| code-39              | varies                     | (calculated width x 1in h)   |

Table 9-19. Barcode Types (continued)

| Code Type        | Number of Digits Supported | Normal size                 |
|------------------|----------------------------|-----------------------------|
| code-39-extended | varies                     | (calculated width x 1in h)  |
| code-32          | 9                          | (calculated w x h)          |
| code-93          | varies                     | (calculated w x h)          |
| code-93-extended | varies                     | (calculated w X h)          |
| codabar 18       | varies                     | (calculated width x 20mm h) |
| codabar-2        | varies                     | (calculated width x 20mm h) |
| pdf-417          | varies                     | (calculated w x h)          |
| data-matrix      | varies                     |                             |

## Code Value of the Bar Code

The character set and semantic constraints depend on the code type selected. The following lists each code type, the expected value type, semantic constraints and size information.

### ean-13

The code represents a 13-digit decimal number. First two digits are the flag code, the next ten digits are the data characters and the last digit is the check character. The check character is calculated as follows:

1. Designate the rightmost character odd.
2. Sum all of the characters in the odd positions and multiply the result by three.
3. Sum all of the characters in the even positions.
4. Add the odd and even totals from steps two and three.
5. Determine the smallest number that, when added to the result from step four, will result in a multiple of 10. This is the check character.

In Europe the first two characters (flag code) are the country identifier and the data characters are split into two groups of five digits each where the first five are a company identifier and the latter five identify an article within that company. The last digit is the check character.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with such a 12 digit value then the 13th digit is automatically calculated by the system.

The nominal size is 37.29mm x 26.26mm (w x h). The bar code area has the measurements 31.35mm x 24.50mm. The left padding measures 3.63mm.

### ean-8

The code represents a 8 digit decimal number. First two digits select the country. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with such a 7 digit value then the 8th digit is automatically calculated by the system.

The nominal size is 26.73mm x 21.64mm (w x h). The bar code area has the measurements 22.11mm x 19.88mm. The left padding measures 2.31mm.

**ean-supplemental-2**

The code represents a 2 digit decimal number that can be used in conjunction with the ean 13 bar code type. Typically the code is placed to the right of the ean 13 bar code. The gap between the rightmost bar of the ean 13 code and the leftmost bar of the supplemental code should not be less than 2.31mm and should not exceed 3.3mm.

The nominal size is 6.6mm x 26.26mm (w x h). The bar code is painted without padding at the sides

**ean-supplemental-5**

The code represents a 5 digit decimal number that can be used in conjunction with the ean 13 bar code type. Typically the code is placed to the right of the ean 13 bar code. The gap between the rightmost bar of the ean 13 code and the leftmost bar of the supplemental code should not be less than 2.31mm and should not exceed 3.3mm.

The nominal size is 15.5mm x 26.26mm (w x h). The bar code is painted without padding at the sides.

**upc-a**

The code represents a 12 digit decimal number. First digit is the 'number system character' code, the next five identify the manufacturer and the following five identify an article. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with such a 11 digit value then the 12th digit is automatically calculated by the system.

The nominal size is 1.469in x 1.020in (w x h). The bar code area has the measurements 1.235in x 0.965in. The left padding measures 0.117in.

**upc-e**

The code represents a 8 digit decimal number. Upc-e is a compressed version of upc-a. By the method of zero suppression some upc a codes are available as upc e codes. The following translation table shows which upc-a numbers can be transformed to short versions:

*Table 9-20. upc-a numbers that can be transformed to short versions*

| UPC-E Number | Digits to insert | Position of Insertion | Resulting UPC-A Number |
|--------------|------------------|-----------------------|------------------------|
| sNNNNN0c     | 00000            | 3                     | sNN00000NNNc           |
| sNNNNN1c     | 10000            | 3                     | sNN10000NNNc           |
| sNNNNN2c     | 20000            | 3                     | sNN20000NNNc           |
| sNNNNN3c     | 00000            | 4                     | sNNN00000NNc           |
| sNNNNN4c     | 00000            | 5                     | sNNNN00000Nc           |

Table 9-20. upc-a numbers that can be transformed to short versions (continued)

| UPC-E Number | Digits to insert | Position of Insertion | Resulting UPC-A Number |
|--------------|------------------|-----------------------|------------------------|
| sNNNNN5c     | 00005            | 6                     | sNNNNN00005c           |
| sNNNNN6c     | 00006            | 6                     | sNNNNN00006c           |
| sNNNNN7c     | 00007            | 6                     | sNNNNN00007c           |
| sNNNNN8c     | 00008            | 6                     | sNNNNN00008c           |
| sNNNNN9c     | 00009            | 6                     | sNNNNN00009c           |

The rightmost digit (checksum digit) can be omitted. When the system is supplied with such a 7 digit value then the 8th digit is automatically calculated by the system.

The nominal size is 0.897in x 1.020in (w x h). The bar code area has the measurements 0.663in x 0.965in. The left padding measures 0.117in.

### upc-supplemental-2

The code represents a 2 digit decimal number that can be used in conjunction with the upc a bar code type. Typically the code is placed to the right of the upc a bar code. The gap between the rightmost bar of the upc a code and the leftmost bar of the supplemental code should not be less than 2.31mm and should not exceed 3.3mm.

The nominal size is 0.26in x 1.02in (w x h). The bar code is painted without padding at the sides.

### upc-supplemental-5

The code represents a 5 digit decimal number that can be used in conjunction with the upc a bar code type. Typically the code is placed to the right of the upc a code. The gap between the rightmost bar of the upc a code and the leftmost bar of the supplemental code should not be less than 2.31mm and should not exceed 3.3mm.

The nominal size is 0.611in x 1.02in (w x h). The bar code is painted without padding at the sides.

The following example shows how the supplemental code can be used in conjunction with a upc a code:

```
<LAYOUTNODE orientation="horizontal" width="min" length="min">
 <BARCODEBOX codeType="upc a" codeValue="01234567891" fontSize="10"
 mirrored="true"/>
 <BARCODEBOX codeType="upc supplemental 2" codeValue="47" fontSize="10"
 mirrored="true"/>
</LAYOUTNODE>
```

### code-128

Code 128 can be used to encode ASCII text of variable length. For this purpose characters can be selected from three character sets, each containing 106 characters. The following table lists the available characters in the character sets A, B and C:

Table 9-21. Available characters in the character sets A, B, and C.

| Reference Number | Character Set A | Character Set B | Character Set C |
|------------------|-----------------|-----------------|-----------------|
| 0                | SP              | SP              | 00              |
| 1                | !               | !               | 01              |
| 2                | "               | "               | 02              |
| 3                | #               | #               | 03              |
| 4                | \$              | \$              | 04              |
| 5                | %               | %               | 05              |
| 6                | &               | &               | 06              |
| 7                | '               | '               | 07              |
| 8                | (               | (               | 08              |
| 9                | )               | )               | 09              |
| 10               | *               | *               | 10              |
| 11               | +               | +               | 11              |
| 12               | ,               | ,               | 12              |
| 13               | -               | -               | 13              |
| 14               | .               | .               | 14              |
| 15               | /               | /               | 15              |
| 16               | 0               | 0               | 16              |
| 17               | 1               | 1               | 17              |
| 18               | 2               | 2               | 18              |
| 19               | 3               | 3               | 19              |
| 20               | 4               | 4               | 20              |
| 21               | 5               | 5               | 21              |
| 22               | 6               | 6               | 22              |
| 23               | 7               | 7               | 23              |
| 24               | 8               | 8               | 24              |
| 25               | 9               | 9               | 25              |
| 26               | :               | :               | 26              |
| 27               | ;               | ;               | 27              |
| 28               | <               | <               | 28              |
| 29               | =               | =               | 29              |
| 30               | >               | >               | 30              |
| 31               | ?               | ?               | 31              |
| 32               | @               | @               | 32              |
| 33               | A               | A               | 33              |
| 34               | B               | B               | 34              |
| 35               | C               | C               | 35              |
| 36               | D               | D               | 36              |
| 37               | E               | E               | 37              |
| 38               | F               | F               | 38              |
| 39               | G               | G               | 39              |

Table 9-21. Available characters in the character sets A, B, and C. (continued)

| Reference Number | Character Set A | Character Set B | Character Set C |
|------------------|-----------------|-----------------|-----------------|
| 40               | H               | H               | 40              |
| 41               | I               | I               | 41              |
| 42               | J               | J               | 42              |
| 43               | K               | K               | 43              |
| 44               | L               | L               | 44              |
| 45               | M               | M               | 45              |
| 46               | N               | N               | 46              |
| 47               | O               | O               | 47              |
| 48               | P               | P               | 48              |
| 49               | Q               | Q               | 49              |
| 50               | R               | R               | 50              |
| 51               | S               | S               | 51              |
| 52               | T               | T               | 52              |
| 53               | U               | U               | 53              |
| 54               | V               | V               | 54              |
| 55               | W               | W               | 55              |
| 56               | X               | X               | 56              |
| 57               | Y               | Y               | 57              |
| 58               | Z               | Z               | 58              |
| 59               | [               | [               | 59              |
| 60               | \               | \               | 60              |
| 61               | ]               | ]               | 61              |
| 62               | ^               | ^               | 62              |
| 63               | _               | _               | 63              |
| 64               | NUL             | `               | 64              |
| 65               | SOH             | a               | 65              |
| 66               | STX             | b               | 66              |
| 67               | ETX             | c               | 67              |
| 68               | EOT             | d               | 68              |
| 69               | ENQ             | e               | 69              |
| 70               | ACK             | f               | 70              |
| 71               | BEL             | g               | 71              |
| 72               | BS              | h               | 72              |
| 73               | HT              | i               | 73              |
| 74               | LF              | j               | 74              |
| 75               | VT              | k               | 75              |
| 76               | FF              | l               | 76              |
| 77               | CR              | m               | 77              |
| 78               | SO              | n               | 78              |
| 79               | SI              | o               | 79              |



Table 9-21. Available characters in the character sets A, B, and C. (continued)

| Reference Number | Character Set A | Character Set B | Character Set C |
|------------------|-----------------|-----------------|-----------------|
| 80               | DLE             | p               | 80              |
| 81               | DC1             | q               | 81              |
| 82               | DC2             | r               | 82              |
| 83               | DC3             | s               | 83              |
| 84               | DC4             | t               | 84              |
| 85               | NAK             | u               | 85              |
| 86               | SYN             | v               | 86              |
| 87               | ETB             | w               | 87              |
| 88               | CAN             | x               | 88              |
| 89               | EM              | y               | 89              |
| 90               | SUB             | z               | 90              |
| 91               | ESC             | {               | 91              |
| 92               | FS              |                 | 92              |
| 93               | GS              | }               | 93              |
| 94               | RS              | ~               | 94              |
| 95               | US              | DEL             | 95              |
| 96               | FNC3            | FNC3            | 96              |
| 97               | FNC2            | FNC2            | 97              |
| 98               | SHIFT           | SHIFT           | 98              |
| 99               | CODEC           | CODEC           | 99              |
| 100              | CODEB           | FNC4            | CODEB           |
| 101              | FNC4            | CODEA           | CODEA           |
| 102              | FNC1            | FNC1            | FNC1            |
| 103              | STARTA          | STARTA          | STARTA          |
| 104              | STARTB          | STARTB          | STARTB          |
| 105              | STARTC          | STARTC          | STARTC          |
| -                | STOP            | STOP            | STOP            |

The code value is expected as a comma-separated list of character names. It must start with a character set selection character STARTA, STARTB, or STARTC and must end with a checksum character followed by the STOP character. If these characters are omitted then the system calculates the checksum automatically and adds the required STOP character.

The control characters CODEA, CODEB and CODEC can be used to switch from one character set to another.

The control character SHIFT changes the character set for the immediately following character from A to B and vice versa.

EAN 128 bar codes can be drawn using this bar code type. Valid EAN 128 codes start with the sequence "STARTC,FNC1". What follows is a sequence of data packets. Each packet starts with a one digit application identifier (AI) from the C character set. The AI is followed by data. The type and amount of expected data is

AI specific. The amount can be fixed or variable. In the case of variable amount of data, the end of the data must be indicated by a FNC1 character. Here is a table with some common AIs.

*Table 9-22. EAN 128 Bar Code Common AIs*

| AI | No of Data Characters            | Description                            |
|----|----------------------------------|----------------------------------------|
| 00 | 18                               | Identification of a delivery unit      |
| 01 | 14                               | An EAN 13 Number including check digit |
| 10 | up to 20 alphanumeric characters | Shipping batch identifier              |
| 11 | 3                                | Production date in the format YYMMDD   |

The complete list of AIs is country specific and is maintained by the local EAN organization. The textual representation requires AIs to be enclosed in round braces. Examples:

*Table 9-23. EAN 128 Bar Code textual representation*

| Textual representation | code value                                                           | Remark                                                                                                               |
|------------------------|----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| (25)03x57              | STARTC, FNC1, 25, 03, CODEB, x, 5, 7                                 | Note that although, it is a variable length AI, no FNC1 is added at the end in the case when the AI is the last one. |
| (30)19(21)3456789012   | STARTC, FNC1, 30, 19, CODEA, FNC1, CODEC, 21, 12, 34, 56, 78, 90, 12 | Note that switching to code a is not necessary, as FNC1 exists in all three character sets.                          |

If a code contains the control character FNC2 then the decoder does not transmit this value. Instead it appends the value to an internal storage. Only after the decoder encounters a value not containing the FNC2 control character, the decoder transmits the temporary storage and the value read. The temporary storage is then cleared. The purpose of this mechanism is to allow the breaking of long text sequences into several lines.

The effects of the control characters FNC3 and FNC4 are decoder specific.

Decoders remove the character STARTA, STARTB, STARTC, CODEA, CODEB, CODEC, SHIFT, FNC1, FNC2, FNC3, FNC4, STOP, and the checksum character from the data before displaying or transmitting the value.

The last but one character is the checksum character that is calculated as follows:  $CS = (Ref(1) + \sum_{i=2}^n ((i-1) * Ref(i))) \bmod 103$  where  $Ref(i)$  is the reference number of the character  $i$ , and  $n$  is the total number of characters. Example:  $codeValue = "STARTB,A,B,C"$   $CS = (104 + (1 * 33) + (2 * 34) + (3 * 35)) \bmod 103$   $CS = 310 \bmod 103$   $CS = 1$  Looking up reference number 1 in character set B yields the exclamation mark '!' character. The full code value including checksum and stop character is therefore:  $codeValue = "STARTB,A,B,C,!,STOP"$

The nominal height is 6.5mm. The width of the bar can be calculated using the following formula:  $L/mm = (5.5N_c + 11N_{ab} + 35) * 0.19$  where  $N_c$  is the number of characters from character set C and  $N_{ab}$  denotes the number of characters from character sets A and B. On each side of the bar area 1.9mm padding is added.

### **code-2-5-industrial**

The code represents a decimal number with a variable number of digits. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars yielding a total of 5 bars per digit. The nominal width of a thin bar is  $THINBARWIDTH = 0.0236in$ . The width of a thick bar is  $THICKBARWIDTH = THINBARWIDTH / thinToThickRelation$  where thinToThickRelation should take values between  $1/3$  and  $1/2$ . Between digits a gap of the width  $GAPWIDTH = THINBARWIDTH / thinToGapRelation$  on page 9-115 is drawn. The padding on both sides measures  $10 * THINBARWIDTH$ .

### **code-2-5-inverted**

The code is the same as code 2/5 industrial, the only difference being that the gaps are drawn instead of the bars.

### **code-2-5-IATA**

The code represents a decimal number with a variable number of digits. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars yielding a total of 5 bars per digit. The nominal width of a thin bar is  $THINBARWIDTH = 0.0236in$ . The width of a thick bar is  $THICKBARWIDTH = THINBARWIDTH / thinToThickRelation$  where thinToThickRelation should take values between  $1/3$  and  $1/2$ . Between digits a gap of width  $GAPWIDTH = THINBARWIDTH / thinToGapRelation$  is drawn. The padding on both sides measures  $10 * THINBARWIDTH$ .

### **code-2-5-interleaved**

The code represents a decimal number with a variable number of digits. The number of digits must be a multiple of 2. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars resulting in a total of 5 bars per digit. The nominal width of a thin bar is  $\text{THINBARWIDTH}=0.0236\text{in}$ . The width of a thick bar is  $\text{THICKBARWIDTH}=\text{THINBARWIDTH}/\text{thinToThickRelation}$  where thinToThickRelation should take values between  $1/3$  and  $1/2$ . Between digits a gap of width  $\text{GAPWIDTH}=\text{THINBARWIDTH}/\text{thinToGapRelation}$  is drawn. The padding on both sides measures  $10*\text{THINBARWIDTH}$ .

### **code-2-5-matrix**

The code represents a decimal number with a variable number of digits. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars yielding a total of 5 bars per digit. The nominal width of a thin bar is  $\text{THINBARWIDTH}=0.0236\text{in}$ . The width of a thick bar is  $\text{THICKBARWIDTH}=\text{THINBARWIDTH}/\text{thinToThickRelation}$  where thinToThickRelation should take values between  $1/3$  and  $1/2$ . Between digits a gap of width  $\text{GAPWIDTH}=\text{THINBARWIDTH}/\text{thinToGapRelation}$  is drawn. The padding on both sides measures  $10*\text{THINBARWIDTH}$ .

### **code-2-5-datalogic**

The code represents a decimal number with a variable number of digits. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars making a total of 5 bars per digit. The nominal width of a thin bar is  $\text{THINBARWIDTH}=0.0236\text{in}$ . The width of a thick bar is  $\text{THICKBARWIDTH}=\text{THINBARWIDTH}/\text{thinToThickRelation}$  where thinToThickRelation thinToGapRelation is drawn. The padding on both sides measures  $10*\text{THINBARWIDTH}$ .

### **code-BCD-matrix**

The code represents a decimal number with a variable number of digits. The number of digits can be specified by setting the noDigits attribute. The last digit is the check character. The check character is calculated the same way as the ean 13 check sum.

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated by the system.

The nominal height is 1in. Digits can differ in width so that two different values having the same number of digits can result in bar codes of differing width. The nominal width of a thin bar is THINBARWIDTH=0.0236in. The width of a thick bar is THICKBARWIDTH=THINBARWIDTH/ thinToThickRelation where thinToThickRelation should take values between 1/3 and 1/2. Between digits a gap of width GAPWIDTH=THINBARWIDTH/ thinToGapRelation is drawn. The default relation value is 1. The padding on both sides measures 10\*THINBARWIDTH.

### code-11-matrix

The code represents a character string with a variable number of characters. The string can contain the digits 0-9 and the '-' character. The number of digits can be specified by setting the noDigits attribute. The code can contain up to two check characters. The attribute noCheckDigits specifies how many check characters are used. If two check characters are used, the rightmost character is the 'K' checksum character and the last but one character is the 'C' checksum character. If only one checksum character is specified then the rightmost character is a 'C' checksum character. The 'C' checksum is calculated as  $C = (\text{Sum}(i=1 \text{ to } n \text{ of } ((i-1 \bmod 10)+1)*\text{Ref}(n-i+1))) \bmod 11$  and the 'K' checksum is calculated using  $K = (\text{Sum}(i=1 \text{ to } n \text{ of } ((i-1 \bmod 9)+1)*\text{Ref}(n-i+1))) \bmod 11$  where n specifies the number of characters to the left of the particular check digit and Ref(i) specifies the value of the character at position i, starting with the leftmost character having the value 1. For digits Ref() yields the digit value itself and for the '-' character Ref() yields the value 10. Example calculating the 'C' checksum: codeValue="12-12345-67890", noDigits="16", noCheckDigits="2",  $n=14$   $C = (1*0+2*9+3*8+4*7+...+10*2+1*1+2*10+3*2+4*1) \bmod 11 = 305 \bmod 11 = 8$  The K checksum can then be calculated as:  $n=15$ ,  $\text{Ref}(15)=C=8$   $K = (1*8+2*0+3*9+4*8+...+9*4+1*3+2*2+3*1+4*10+5*2+6*1) \bmod 11 = 350 \bmod 11 = 9$  resulting in the code value codeValue="12-12345-6789089".

If the value supplied in codeValue has the length noDigits- noCheckDigits then the system automatically calculates and supplies the check digits.

The nominal height is 1 in. Digits can differ in width so that two different values having the same number of digits can result in bar codes of differing width. The nominal width of a thin bar is THINBARWIDTH=0.0236in. The width of a thick bar is THICKBARWIDTH=THINBARWIDTH/ thinToThickRelation where thinToThickRelation should take values between 1/3 and 1/2. Between digits a gap of width GAPWIDTH=THINBARWIDTH/ thinToGapRelation is drawn. The default relation value is 1. The padding on both sides measures 10\*THINBARWIDTH.

### code-39

Code 39 can be used to encode ASCII text of variable length. Characters can be selected from the following set of 43 characters.

Table 9-24. Characters for encoding ASCII text of variable length for Code 39

| Reference Number | Character |
|------------------|-----------|
| 0                | 0         |

Table 9-24. Characters for encoding ASCII text of variable length for Code 39 (continued)

| Reference Number | Character |
|------------------|-----------|
| 1                | 1         |
| 2                | 2         |
| 3                | 3         |
| 4                | 4         |
| 5                | 5         |
| 6                | 6         |
| 7                | 7         |
| 8                | 8         |
| 9                | 9         |
| 10               | A         |
| 11               | B         |
| 12               | C         |
| 13               | D         |
| 14               | E         |
| 15               | F         |
| 16               | G         |
| 17               | H         |
| 18               | I         |
| 19               | J         |
| 20               | K         |
| 21               | L         |
| 22               | M         |
| 23               | N         |
| 24               | O         |
| 25               | P         |
| 26               | Q         |
| 27               | R         |
| 28               | S         |
| 29               | T         |
| 30               | U         |
| 31               | V         |
| 32               | W         |
| 33               | X         |
| 34               | Y         |
| 35               | Z         |
| 36               | -         |
| 37               | .         |
| 38               |           |
| 39               | \$        |
| 40               | /         |

Table 9-24. Characters for encoding ASCII text of variable length for Code 39 (continued)

| Reference Number | Character |
|------------------|-----------|
| 41               | +         |
| 42               | %         |

The last but one character is the checksum character that is calculated as follows:  $CS = \text{Sum}(i=1 \text{ to } n \text{ of Ref}(i)) \bmod 43$  where  $\text{Ref}(i)$  is the reference number of the character  $i$ , and  $n$  is the total number of characters. Example:  
codeValue="DATALOGIC"  
 $CS = (13+10+29+10+21+24+16+18+12) \bmod 43, CS=153 \bmod 43, CS=24$   
Looking up reference number 24 yields the character 'O'. The full code value including checksum and stop character is therefore:  
codeValue="DATALOGICO"

Some scanners support an "extended mode" in which the scanner recognizes special two-character sequences of the code 39 character set and decodes these as ASCII characters. With this method the full 128 character ASCII character set can be encoded using the 43 basic characters of code 39. Scanners are switched into "extended mode" by a bar containing the sequence "+\$". The sequence "-\$" returns the scanner into regular mode. The following table lists the character sequences needed to encode the ASCII character set in "extended mode".

Table 9-25. Character sequences needed to encode the ASCII character set in "extended mode"

| ASCII code | Code 39 sequence |
|------------|------------------|
| NUL        | %U               |
| SOH        | \$A              |
| STX        | \$B              |
| ETX        | \$C              |
| EOT        | \$D              |
| ENQ        | \$E              |
| ACK        | \$F              |
| BEL        | \$G              |
| BS         | \$H              |
| HT         | \$I              |
| LF         | \$J              |
| VT         | \$K              |
| FF         | \$L              |
| CR         | \$M              |
| SO         | \$N              |
| SI         | \$O              |
| DLE        | \$P              |
| DC1        | \$Q              |
| DC2        | \$R              |
| DC3        | \$S              |
| DC4        | \$T              |
| NAK        | \$U              |

Table 9-25. Character sequences needed to encode the ASCII character set in "extended mode" (continued)

| ASCII code | Code 39 sequence |
|------------|------------------|
| SYN        | \$V              |
| ETB        | \$W              |
| CAN        | \$X              |
| EM         | \$Y              |
| SUB        | \$Z              |
| ESC        | %A               |
| FS         | %B               |
| GS         | %C               |
| RS         | %D               |
| US         | %E               |
| SP         |                  |
| !          | /A               |
| "          | /B               |
| #          | /C               |
| \$         | /D               |
| %          | /E               |
| &          | /F               |
| '          | /G               |
| (          | /H               |
| )          | /I               |
| *          | /J               |
| +          | /K               |
| ,          | /L               |
| -          | -                |
| .          | .                |
| /          | /O               |
| 0          | 0                |
| 1          | 1                |
| 2          | 2                |
| 3          | 3                |
| 4          | 4                |
| 5          | 5                |
| 6          | 6                |
| 7          | 7                |
| 8          | 8                |
| 9          | 9                |
| :          | /Z               |
| ;          | %F               |
| >TD/TD     |                  |



Table 9-25. Character sequences needed to encode the ASCII character set in "extended mode" (continued)

| ASCII code | Code 39 sequence |
|------------|------------------|
| =          | %H               |
| >          | %I               |
| ?          | %J               |
| @          | %V               |
| A          | A                |
| B          | B                |
| C          | C                |
| D          | D                |
| E          | E                |
| F          | F                |
| G          | G                |
| H          | H                |
| I          | I                |
| J          | J                |
| K          | K                |
| L          | L                |
| M          | M                |
| N          | N                |
| O          | O                |
| P          | P                |
| Q          | Q                |
| R          | R                |
| S          | S                |
| T          | T                |
| U          | U                |
| V          | V                |
| W          | W                |
| X          | X                |
| Y          | Y                |
| Z          | Z                |
| [          | %K               |
| \          | %L               |
| ]          | %M               |
| ^          | %N               |
| _          | %O               |
| `          | %W               |
| a          | +A               |
| b          | +B               |
| c          | +C               |

Table 9-25. Character sequences needed to encode the ASCII character set in "extended mode" (continued)

| ASCII code | Code 39 sequence |
|------------|------------------|
| d          | +D               |
| e          | +E               |
| f          | +F               |
| g          | +G               |
| h          | +H               |
| i          | +I               |
| j          | +J               |
| k          | +K               |
| l          | +L               |
| m          | +M               |
| n          | +N               |
| o          | +O               |
| p          | +P               |
| q          | +Q               |
| r          | +R               |
| s          | +S               |
| t          | +T               |
| u          | +U               |
| v          | +V               |
| w          | +W               |
| x          | +X               |
| y          | +Y               |
| z          | +Z               |
| {          | %P               |
|            | %Q               |
| }          | %R               |
| ~          | %S               |
| DEL        | %T               |

The code type "code 39 extended" automatically creates the necessary two-character sequences and can be used as a more convenient way of creating extended code 39 bar codes.

If the value specified by codeValue is shorter by one character than the number of digits specified by noDigits then the checksum character is automatically calculated and the character is appended to codeValue.

The nominal height is 1in. The nominal width of a thin bar is THINBARWIDTH=0.0197in. The width of a thick bar is THICKBARWIDTH=THINBARWIDTH/ thinToThickRelation where thinToThickRelation should take values between 1/3 and 1/2. Between digits a gap with the width GAPWIDTH=THINBARWIDTH/ thinToGapRelation is drawn. The

default relation value is 1. The padding on both sides measures 10\*THINBARWIDTH.

### code-39-extended

Code 39 Extended can be used to encode text of variable length using the ASCII character set. The code value is expected as a comma-separated list of ASCII character names. The following table list the names and the textual representation that is used in printout.

*Table 9-26. Code 39 Extended names and the textual representation that is used in printout*

| ASCII code | Textual representation |
|------------|------------------------|
| NUL        | <NUL>                  |
| SOH        | <SOH>                  |
| STX        | <STX>                  |
| ETX        | <ETX>                  |
| EOT        | <EOT>                  |
| ENQ        | <ENQ>                  |
| ACK        | <ACK>                  |
| BEL        | <BEL>                  |
| BS         | <BS>                   |
| HT         | <HT>                   |
| LF         | <LF>                   |
| VT         | <VT>                   |
| FF         | <FF>                   |
| CR         | <CR>                   |
| SO         | <SO>                   |
| SI         | <SI>                   |
| DLE        | <DLE>                  |
| DC1        | <DC1>                  |
| DC2        | <DC2>                  |
| DC3        | <DC3>                  |
| DC4        | <DC4>                  |
| NAK        | <NAK>                  |
| SYN        | <SYN>                  |
| ETB        | <ETB>                  |
| CAN        | <CAN>                  |
| EM         | <EM>                   |
| SUB        | <SUB>                  |
| ESC        | <ESC>                  |
| FS         | <FS>                   |
| GS         | <GS>                   |
| RS         | <RS>                   |
| US         | <US>                   |
| SP         |                        |

Table 9-26. Code 39 Extended names and the textual representation that is used in printout (continued)

| ASCII code | Textual representation |
|------------|------------------------|
| !          | !                      |
| "          | \"                     |
| #          | #                      |
| \$         | \$                     |
| %          | %                      |
| &          | &                      |
| '          | '                      |
| (          | (                      |
| )          | )                      |
| *          | *                      |
| +          | +                      |
| ,          | },                     |
| -          | -                      |
| .          | .                      |
| /          | /                      |
| 0          | 0                      |
| 1          | 1                      |
| 2          | 2                      |
| 3          | 3                      |
| 4          | 4                      |
| 5          | 5                      |
| 6          | 6                      |
| 7          | 7                      |
| 8          | 8                      |
| 9          | 9                      |
| :          | :                      |
| ;          | ;                      |
| <          | <                      |
| =          | =                      |
| >          | >                      |
| ?          | ?                      |
| @          | @                      |
| A          | A                      |
| B          | B                      |
| C          | C                      |
| D          | D                      |
| E          | E                      |
| F          | F                      |
| G          | G                      |

*Table 9-26. Code 39 Extended names and the textual representation that is used in printout (continued)*

| ASCII code | Textual representation |
|------------|------------------------|
| H          | H                      |
| I          | I                      |
| J          | J                      |
| K          | K                      |
| L          | L                      |
| M          | M                      |
| N          | N                      |
| O          | O                      |
| P          | P                      |
| Q          | Q                      |
| R          | R                      |
| S          | S                      |
| T          | T                      |
| U          | U                      |
| V          | V                      |
| W          | W                      |
| X          | X                      |
| Y          | Y                      |
| Z          | Z                      |
| [          | [                      |
| \          | \                      |
| ]          | ]                      |
| ^          | ^                      |
| _          | _                      |
| `          | `                      |
| a          | a                      |
| b          | b                      |
| c          | c                      |
| d          | d                      |
| e          | e                      |
| f          | f                      |
| g          | g                      |
| h          | h                      |
| i          | i                      |
| j          | j                      |
| k          | k                      |
| l          | l                      |
| m          | m                      |
| n          | n                      |

Table 9-26. Code 39 Extended names and the textual representation that is used in printout (continued)

| ASCII code | Textual representation |
|------------|------------------------|
| o          | o                      |
| p          | p                      |
| q          | q                      |
| r          | r                      |
| s          | s                      |
| t          | t                      |
| u          | u                      |
| v          | v                      |
| w          | w                      |
| x          | x                      |
| y          | y                      |
| z          | z                      |
| {          | {                      |
|            |                        |
| }          | }                      |
| ~          | ~                      |
| DEL        | <DEL>                  |

## code-32

The code represents a decimal number with 9 digits. The last digit is the check character. The check character is calculated as follows:  $CS = \text{Sum}(i=1 \text{ to } 8 \text{ of } CT((i-1)\%2*2*Ref(i))) \bmod 10$  where  $Ref(i)$  denotes the value of the digit at position  $i$  (the leftmost digit has the index 1) and  $CT()$  denotes the cross total of its argument (e.g.  $CT(18)=1+8=9$ ).

The rightmost digit (checksum digit) can be omitted. When the system is supplied with a 8 digit value then the check digit is automatically calculated.

The specified 9 digit decimal value is translated into a 6 digit base 32 value which is then drawn using characters from the base 39 character set and bar drawing scheme. The following 32 character table is used to encode the 6 digit base 32 value:

Table 9-27. 32 character table is used to encode the 6 digit base 32 value

| Reference Number | Character |
|------------------|-----------|
| 0                | 0         |
| 1                | 2         |
| 2                | 3         |
| 3                | 4         |
| 4                | 5         |
| 5                | 6         |
| 6                | 7         |

Table 9-27. 32 character table is used to encode the 6 digit base 32 value (continued)

| Reference Number | Character |
|------------------|-----------|
| 7                | 8         |
| 8                | 9         |
| 9                | B         |
| 10               | C         |
| 11               | C         |
| 12               | D         |
| 13               | F         |
| 14               | G         |
| 15               | H         |
| 16               | J         |
| 17               | K         |
| 18               | L         |
| 19               | M         |
| 20               | N         |
| 21               | P         |
| 22               | Q         |
| 23               | R         |
| 24               | S         |
| 25               | T         |
| 26               | U         |
| 27               | V         |
| 28               | W         |
| 29               | X         |
| 30               | Y         |
| 31               | Z         |

Based on this table the following bar codes will produce identical bar pattern:<BARCODEBOX codeType="code 32" check="true" codeValue="026089019" noText="true" orientation="vertical" mirrored="true"/><BARCODEBOX codeType="code 39" check="false" codeValue="0SW5KV" noText="true" orientation="vertical" mirrored="true"/> Note that the bars are drawn without text by setting noText="true". This is necessary to produce identical output since otherwise the code 32 box draws the decimal number "026089019" while the code 39 box would draw the string "0SW5KV".

The nominal height is 1in. Each digit is drawn using a pattern of two wide and three thin bars making a total of 5 bars per digit. The nominal width of a thin bar is THINBARWIDTH=0.0197in. The width of a thick bar is THICKBARWIDTH=THINBARWIDTH/ thinToThickRelation where thinToThickRelation should take values between 1/3 and 1/2. Between digits a gap with the width GAPWIDTH=THINBARWIDTH/ thinToGapRelation is drawn. The padding on both sides measures 10\*THINBARWIDTH.

## code-93

Code 93 can be used to encode ASCII text of variable length. Characters can be selected from the following set of 47 characters. You can use the code-93-extended type to encode the full 128 character ASCII character set using the 47 basic characters of code 93.

*Table 9-28. code-93*

| Reference Number | Character |
|------------------|-----------|
| 0                | 0         |
| 1                | 1         |
| 2                | 2         |
| 3                | 3         |
| 4                | 4         |
| 5                | 5         |
| 6                | 6         |
| 7                | 7         |
| 8                | 8         |
| 9                | 9         |
| 10               | A         |
| 11               | B         |
| 12               | C         |
| 13               | D         |
| 14               | E         |
| 15               | F         |
| 16               | G         |
| 17               | H         |
| 18               | I         |
| 19               | J         |
| 20               | K         |
| 21               | L         |
| 22               | M         |
| 23               | N         |
| 24               | O         |
| 25               | P         |
| 26               | Q         |
| 27               | R         |
| 28               | S         |
| 29               | T         |
| 30               | U         |
| 31               | V         |
| 32               | W         |
| 33               | X         |
| 34               | Y         |



Table 9-28. code-93 (continued)

| Reference Number | Character |
|------------------|-----------|
| 35               | Z         |
| 36               | -         |
| 37               | .         |
| 38               |           |
| 39               | \$        |
| 40               | /         |
| 41               | +         |
| 42               | %         |
| 43               | !         |
| 44               | ?         |
| 45               | \         |
| 46               |           |

The textual representation of the last four characters "!?\|" can be configured by setting the controlCharacters attribute.

noCheckDigits specifies how many check characters are used. In the case of two check characters the rightmost character is the 'K' checksum character and the last but one character is the 'C' checksum character. if only one checksum character is specified then the rightmost character is a 'C' checksum character. The 'C' checksum is calculated as follows  $C = (\text{Sum}(i=1 \text{ to } n \text{ of } ((i-1 \bmod 20)+1) * \text{Ref}(n-i+1))) \bmod 47$  and the 'K' checksum is calculated as follows  $K = (\text{Sum}(i=1 \text{ to } n \text{ of } ((i-1 \bmod 15)+1) * \text{Ref}(n-i+1))) \bmod 47$  where n specifies the number of characters left of the particular check digit and Ref(i) specifies the reference value of the character at position i starting with the leftmost character having the index value 1. Reference numbers can be looked up in the first column of Table 9-28 on page 9-176. Example calculating the 'C' checksum: codeValue="DATALOGIC", noDigits="11", noCheckDigits="2",  $n=9$   $C = (1*12+2*18+3*16+4*24+....7*29+8*10+9*13) \bmod 47 = 757 \bmod 47 = 5$  The K checksum can then be calculated as:  $n=10$ ,  $\text{Ref}(10)=C=5$   $K = (1*5+2*12+3*18+4*16+....8*29+9*10+10*13) \bmod 47 = 915 \bmod 47 = 22$  resulting in the code value codeValue="DATALOGIC5M".

If the value supplied in codeValue has the length noDigits- noCheckDigits then the system automatically calculates and supplies the check digits.

### code-93-extended

Some scanners support an "extended mode" in which the scanner recognizes special two-character sequences of the code 93 character set and decodes these as ASCII characters. With this method the full 128 character ASCII character set can be encoded using the 47 basic characters of code 93. T

The nominal height is 1in. The nominal width of a bar is BARWIDTH=0.022in. The width of the entire bar is  $(1 + (\text{noDigits} + 2) * 9) * \text{BARWIDTH}$ . The padding on both sides measures  $10 * \text{BARWIDTH}$ .

This code type automatically creates the necessary two-character sequences and can be used as a more convenient way of creating extended code 93 bar codes.

Table 9-29. code-93-extended sequence table

| ASCII code | Code 93 sequence |
|------------|------------------|
| NUL        | ?U               |
| SOH        | !A               |
| STX        | !B               |
| ETX        | !C               |
| EOT        | !D               |
| ENQ        | !E               |
| ACK        | !F               |
| BEL        | !G               |
| BS         | !H               |
| HT         | !I               |
| LF         | !J               |
| VT         | !K               |
| FF         | !L               |
| CR         | !M               |
| SO         | !N               |
| SI         | !O               |
| DLE        | !P               |
| DC1        | !Q               |
| DC2        | !R               |
| DC3        | !S               |
| DC4        | !T               |
| NAK        | !U               |
| SYN        | !V               |
| ETB        | !W               |
| CAN        | !X               |
| EM         | !Y               |
| SUB        | !Z               |
| ESC        | ?A               |
| FS         | ?B               |
| GS         | ?C               |
| RS         | ?D               |
| US         | ?E               |
| SP         |                  |
| !          | \A               |
| "          | \B               |
| #          | \C               |
| \$         | \$               |
| %          | %                |
| &          | \F               |
| '          | \G               |

Table 9-29. code-93-extended sequence table (continued)

| ASCII code | Code 93 sequence |
|------------|------------------|
| (          | \H               |
| )          | \I               |
| *          | \J               |
| +          | +                |
| ,          | \L               |
| -          | -                |
| .          | .                |
| /          | /                |
| 0          | 0                |
| 1          | 1                |
| 2          | 2                |
| 3          | 3                |
| 4          | 4                |
| 5          | 5                |
| 6          | 6                |
| 7          | 7                |
| 8          | 8                |
| 9          | 9                |
| :          | \Z               |
| ;          | ?F               |
| <          | ?G               |
| =          | ?H               |
| >          | ?I               |
| ?          | ?J               |
| @          | ?V               |
| A          | A                |
| B          | B                |
| C          | C                |
| D          | D                |
| E          | E                |
| F          | F                |
| G          | G                |
| H          | H                |
| I          | I                |
| J          | J                |
| K          | K                |
| L          | L                |
| M          | M                |
| N          | N                |
| O          | O                |

Table 9-29. code-93-extended sequence table (continued)

| ASCII code | Code 93 sequence |
|------------|------------------|
| P          | P                |
| Q          | Q                |
| R          | R                |
| S          | S                |
| T          | T                |
| U          | U                |
| V          | V                |
| W          | W                |
| X          | X                |
| Y          | Y                |
| Z          | Z                |
| [          | ?K               |
| \          | ?L               |
| ]          | ?M               |
| ^          | ?N               |
| _          | ?O               |
| `          | ?W               |
| a          | A                |
| b          | B                |
| c          | C                |
| d          | D                |
| e          | E                |
| f          | F                |
| g          | G                |
| h          | H                |
| i          | I                |
| j          | J                |
| k          | K                |
| l          | L                |
| m          | M                |
| n          | N                |
| o          | O                |
| p          | P                |
| q          | Q                |
| r          | R                |
| s          | S                |
| t          | T                |
| u          | U                |
| v          | V                |
| w          | W                |

Table 9-29. code-93-extended sequence table (continued)

| ASCII code | Code 93 sequence |
|------------|------------------|
| x          | X                |
| y          | Y                |
| z          | Z                |
| {          | ?P               |
|            | ?Q               |
| }          | ?R               |
| ~          | ?S               |
| DEL        | ?T               |

## codabar-18

Codabar 18 can be used to encode text of variable length by using characters from the following character set:

Table 9-30. Character set for Codabar 18

| Reference Number | Character |
|------------------|-----------|
| 0                | 0         |
| 1                | 1         |
| 2                | 2         |
| 3                | 3         |
| 4                | 4         |
| 5                | 5         |
| 6                | 6         |
| 7                | 7         |
| 8                | 8         |
| 9                | 9         |
| 10               | -         |
| 11               | \$        |
| 12               | :         |
| 13               | /         |
| 14               | .         |
| 15               | +         |
| 16               | a         |
| 17               | b         |
| 18               | c         |
| 19               | d         |
| 16               | t         |
| 17               | n         |
| 18               | *         |
| 19               | e         |

The first character of any code must be either 'a', 'b', 'c' or 'd'. The last character of any code must be either 't', 'n', '\*' or 'e'. All other characters must have an ordinal value less than 16.

The last but one character is the checksum character that is calculated as follows:  $CS = 16 - (\text{Sum}(i=1 \text{ to } n \text{ of Ref}(i))) \bmod 16$  Where Ref(i) is the reference number of the character i, and n is the total number of characters. Example: codeValue="a37859n"  $CS = 16 - ((16+3+7+8+5+9+17) \bmod 16)$ ,  $CS = 16 - (65 \bmod 16)$ ,  $CS = 15$  Looking up reference number 15 yields the character '+'. The full code value including checksum is therefore: codeValue="a37859+n"

When the system is supplied with a value that is one digit shorter than specified by noDigits then the check digit is automatically calculated.

The nominal height is 20mm. The nominal width of a thin bar is THINBARWIDTH=0.165mm. The width of a character is 2.095mm. Between digits a gap with the width GAPWIDTH=THINBARWIDTH/ thinToGapRelation is drawn. The padding on both sides measures 10\*THINBARWIDTH.

## codabar-2

Codabar 2 can be used to encode text of variable length by using characters from the following character set:

Table 9-31. Character set for Codabar 2

| Reference Number | Character |
|------------------|-----------|
| 0                | 0         |
| 1                | 1         |
| 2                | 2         |
| 3                | 3         |
| 4                | 4         |
| 5                | 5         |
| 6                | 6         |
| 7                | 7         |
| 8                | 8         |
| 9                | 9         |
| 10               | -         |
| 11               | \$        |
| 12               | :         |
| 13               | /         |
| 14               | .         |
| 15               | +         |
| 16               | a         |
| 17               | b         |
| 18               | c         |
| 19               | d         |

The first and the last character of any code must be either 'a', 'b', 'c' or 'd'. All other characters must have an ordinal value less than 16.

The last but one character is the checksum character that is calculated as follows:  $CS = 16 - (\text{Sum}(i=1 \text{ to } n \text{ of Ref}(i))) \bmod 16$  where  $\text{Ref}(i)$  is the reference number of the character  $i$ , and  $n$  is the total number of characters. Example: `codeValue="a37859b"`  $CS = 16 - ((16+3+7+8+5+9+17) \bmod 16)$ ,  $CS = 16 - (65 \bmod 16)$ ,  $CS = 15$ . Looking up reference number 15 yields the character '+'. The full code value including checksum is therefore: `codeValue="a37859+b"`

When the system is supplied with a value that is one digit shorter than specified by `noDigits` then the check digit is calculated automatically.

The nominal height is 20mm. The nominal width of a thin bar is `THINBARWIDTH=0.165mm`. The width of a thick bar is `THICKBARWIDTH=THINBARWIDTH/ thinToThickRelation` where `thinToThickRelation` should take values between  $1/3$  and  $1/2$ . Between digits a gap having the width `GAPWIDTH=THINBARWIDTH/ thinToGapRelation` is drawn. The padding on both sides measures  $10 * \text{THINBARWIDTH}$ .

The "American Blood Commission" defines a code type known as Codabar-ABC. There are two types of Codabar-ABC codes that can both be built using the Codabar 2 barcode type:

**Single bar Codabar-ABC** - this type is identical with Codabar 2, the constraint being that values have to be at least 5 digits long.

**Dual bar Codabar-ABC** - This type can be printed by printing two Codebar 2 horizontally adjacent to each other. The gap between the two bars should not exceed 15mm. Additionally the code of the first bar must end with a 'd' character while the second must start with the character 'd'. The following code creates a valid Dual bar Codabar-ABC bar for the values "c1234d" and "d5678a".

## pdf-417

Pdf-417 can be used to encode text and binary data of variable length. It is possible to encode up to 925 code words (minimal error correction degree). Text, binary data and numeric data are compressed into code words so that the limits for these types are:

- 1850 ASCII characters
- 1108 Byte values
- 2710 numeric values

The limits for mixed text are lower, as control characters are inserted to switch between the different compression modes.

Setting the error correction degree to the recommended minimum lowers the maximum number of encodable code words to 863 yielding the following limits for the encodable types:

- 1726 ASCII characters
- 1033 Byte values
- 2528 numeric values

The conversion of the data specified in this attribute (codeValue) to the internal representation is done by an algorithm that minimizes space (Sometimes there is more than one option to encode a particular piece of data). Non ascii characters are encoded using the specified encoding table. Byte values that cannot be represented in an XML document (for example all characters lower 0x20 except 0x9, 0xa and 0xd) can be represented by a backslash ("\") character followed by a 3 digit octal literal. The backslash character itself can be escaped by a sequence of two backslash characters.

The following attributes are unique to this bar code type:

- dataSymbolsPerLine:
  - Type: Integer value.
  - Description: Specifies the number of data symbols per line. The value must be an integer between 1 and 30. Low values cause more narrow printout with more lines. The number of lines is not allowed to exceed 90. It should be noted, that the overall required image space usually grows with lower values because there is a constant amount of organizational information which is added with each additional line. This is not generally the case, since lines have to be filled with padding so that specially with small amounts of data a larger value may actually create a larger image. If the value is not specified, the system computes a value that minimizes image space.
  - Fails if: Value cannot be parsed as a integer value. Value is not in the range 1..30.
  - Default value: A value that minimizes the overall image size
- rawCodeValue:
  - Type: A comma-separated list of integers in the range 0..899.
  - Description: This attribute can be used instead of codeValue to specify the code value at a lower level giving more control on the encoded data.
  - Fails if: Encoding for non ascii characters in the code value.
  - Default value: not set
- errorCorrectionDegree:
  - Type: Integer value.
  - Description: Specifies the error correction degree. Valid values are in the range 0..8. Higher values make the image more robust.
  - Fails if: Value cannot be parsed as a integer value. Value is not in the range 0..8.
  - Default value: A value that proportional to the data size
- encoding
  - Type: Encoding
  - Description: Sets the encoding for non ascii characters in the code value. Run "java CharSetInfo" for a list of character set encodings available on a particular platform. Valid example values are 'ISO-8859-15' or 'IBM437'.
  - Fails if:
    - Value is not a valid host name
    - Socket connection cannot be established
  - Default value: not set (the lower 8 bits of the unicode values are encoded)



## **data-matrix**

A Data matrix barcode can be used to encode text and binary data of variable length. It is possible to encode up to 1558 code words. Text, binary data and numeric data are compressed into code words so that the limits for these types are:

- Alphanumeric data - up to 2335 characters
- 8-bit byte data - 1555 characters
- Numeric data - 3116 digits

The limits for mixed text are lower, as control characters are inserted to switch between the different compression modes.

Data matrix symbols exist in several flavors with varying methods and degrees of error correction. Of the possible options ECC 000, ECC 050, ECC 080, ECC 100, ECC 140 and ECC 200 this implementation supports only ECC 200 using Reed Solomon error correction.

Special control characters like FNC1, ECI and "structured append" are currently not available.

The conversion of the data specified in this attribute (codeValue) to the internal representation is done by an algorithm that minimizes space. Byte values that cannot be represented in an XML document (for example all characters lower than 0x20 except 0x9, 0xa and 0xd) can be represented by a backslash ('\') character followed by a 3 digit octal literal. The backslash character itself can be escaped by a sequence of two backslash characters.

The current implementation can encode any character that exists in the code page ISO-8859-1 (Latin 1). Any attempt to encode other characters will fail. Future versions will insert ECI control characters to switch to other code pages if the character is available there.

The following attribute is unique to this bar code type:

- preferRectangularSymbols
  - Type: Boolean value
  - Description: If you are concerned about running out of space in the vertical of the page, you might prefer a symbol that is wider than it is high. This property produces a rectangular shaped symbol if the encoded data does not exceed 49 code words.
  - Fails if: Value cannot be parsed as a boolean value. Valid values are True, False.
  - Default value: False

## **RTL Classes Overview**

### **Object Classes**

There are object classes for each type of the form item properties. The type of each property is indicated in the Properties page. The methods of these classes may be used in RTL Expressions to define a property value.

- The Boolean Class - contains methods used for all logical operations.
- The Color Class - contains methods and static member variables related to color.

- Enum - a set of classes, consisting of a class for each property of this type; each class contains static member variables that provide a list of valid values for the corresponding property.
- The Date Class - provides methods for date formatting and parsing.
- The FGLNumericVariable Class - For every 4GL numeric variable of report data, an object is created that is an instance of an FGLNumericVariable. These objects hold the value of the 4GL variable.
- The FGLStringVariable Class - For every CHAR, VARCHAR, STRING, TEXT, DATE, DATETIME and INTERVAL 4GL variable of report data, an object is created that is an instance of an FGLStringVariable. These objects hold the value of the 4GL variable.
- The Numeric Class - contains methods used for all numeric operations; the class has the precision of a double and the arithmetic operators are defined for objects of this type.
- The Runtime Class - provides functions that simplify the creation of dynamic designs that change behavior based on the runtime setup.
- The String Class - contains methods used for all string operations.

See also: Using RTL Expressions, Properties

## Dimension Resolver

### Unit Names

Table 9-32. Unit Names

| Unit abbreviations | Unit description                                                           | Point value                                                                                                                                                                                                                                                                                                                 | Example       |
|--------------------|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| point   pt         | Point value<br>(72.27point = 1in)                                          | 1                                                                                                                                                                                                                                                                                                                           | 10point       |
| scrpixels[xy]      | Screen pixel value<br>(e.g. for 96DPI:<br>96pixel = 1in =<br>72.27point)   | Depends on screen resolution of the current screen. The default value is taken from the local VM, not from a potential viewer residing on a different machine.                                                                                                                                                              | 640scrpixelsx |
| prnpixels[xy]      | Printer pixel value<br>(e.g. for 300DPI<br>300pixel = 1in =<br>72.27point) | Depends on printer resolution of current printer (printer resolution defaults to 300 when there is no printer on the current pipe or when the printer is on a part of the pipe that resides on another machine).<br><b>Important:</b> Due to a conceptual problem the value is currently based on a fixed value of 300 DPI. | 150prnpixelsy |
| pica   pc          | Pica value (12point = 1pica)                                               | 12                                                                                                                                                                                                                                                                                                                          | 3pica         |

Table 9-32. Unit Names (continued)

| Unit abbreviations | Unit description                   | Point value       | Example |
|--------------------|------------------------------------|-------------------|---------|
| inch   in          | Inch value<br>(72.27point = 1in)   | 72.27             | 10in    |
| bigpoint   bp      | Big point (72bigpoint<br>= 1in)    | 72.27 / 72        | 10bp    |
| cm                 | Centimeter value<br>(2.54cm = 1in) | 72.27 / 2.54      | 10cm    |
| mm                 | Centimeter value<br>(10mm = 1cm)   | 72.27 / 2.54 / 10 | 10mm    |
| didot   dd         | Didot point (1157dd<br>= 1238pt)   | 1238 / 1157       | 10dd    |
| cicero   cc        | Cicero value(1cc =<br>12dd)        | 12*1238 / 1157    | 0.5cc   |

## Paper Format Abbreviations

Table 9-33. Paper Format Abbreviations

| Unit abbreviations                                      | Unit Description           | Width value | Length value |
|---------------------------------------------------------|----------------------------|-------------|--------------|
| iso4a0(width   length)                                  | ISO 4A0                    | 1682mm      | 2378mm       |
| iso2a0(width   length)                                  | ISO 2A0                    | 1189mm      | 1682mm       |
| (isodesignatedlong   dl  <br>dinlang)(width   length)   | ISO designated long        | 110mm       | 220mm        |
| executive(width   length)                               | Executive                  | 7.25in      | 10.5in       |
| (folio  <br>germanlegalfanfold)(width<br>  length)      | Folio/German legal fanfold | 8.5in       | 13in         |
| (invoice   statement)(width<br>  length)                | Invoice/Statement          | 5.5in       | 8.5in        |
| (ledger   tabloid  <br>11x7)(width   length)            | Ledger/Tabloid             | 11in        | 7in          |
| (naletter   letter  <br>note)(width   length)           | Letter                     | 8.5in       | 11in         |
| (nalegal   legal)(width  <br>length)                    | Legal                      | 8.5in       | 14in         |
| quarto(width   length)                                  | Quarto                     | 215mm       | 275mm        |
| a(width   length)                                       | Engineering A              | 8.5in       | 11in         |
| b(width   length)                                       | Engineering B              | 11in        | 17in         |
| c(width   length)                                       | Engineering C              | 17in        | 22in         |
| d(width   length)                                       | Engineering D              | 22in        | 34in         |
| e(width   length)                                       | Engineering E              | 34in        | 44in         |
| (na10x15envelope  <br>10x15envelope)(width  <br>length) | Envelope 10x15             | 10in        | 15in         |
| (na10x14envelope  <br>10x14envelope)(width  <br>length) | Envelope 10x14             | 10in        | 14in         |

Table 9-33. Paper Format Abbreviations (continued)

| Unit abbreviations                                              | Unit Description        | Width value | Length value |
|-----------------------------------------------------------------|-------------------------|-------------|--------------|
| (na10x13envelope   10x13envelope)(width   length)               | Envelope 10x13          | 10in        | 13in         |
| (na9x12envelope   9x12envelope)(width   length)                 | Envelope 9x12           | 9in         | 12in         |
| (na9x11envelope   9x11envelope)(width   length)                 | Envelope 9x11           | 9in         | 11in         |
| (na7x9envelope   7x9envelope)(width   length)                   | Envelope 7x9            | 7in         | 9in          |
| (na6x9envelope   6x9envelope)(width   length)                   | Envelope 6x9            | 6in         | 9in          |
| (nanumber9envelope   number9envelope   env9)(width   length)    | Envelope number 9       | 3+7/8in     | 8+7/8in      |
| (nanumber10envelope   number10envelope   env10)(width   length) | Envelope number 10      | 1+1/8in     | 9+1/2in      |
| (nanumber11envelope   number11envelope   env11)(width   length) | Envelope number 11      | 4+1/2in     | 10+3/8in     |
| (nanumber12envelope   number12envelope   env12)(width   length) | Envelope number 12      | 4+3/4in     | 11in         |
| (nanumber14envelope   number14envelope   env14)(width   length) | Envelope number 14      | 5in         | 11+1/2in     |
| (envinvite   inviteenvelope   invite)(width   length)           | Invite envelope         | 220mm       | 220mm        |
| (envitaly   italyenvelope   italy)(width   length)              | Italy envelope          | 110mm       | 230mm        |
| (envmonarch   monarchenvelope   monarch)(width   length)        | Monarch envelope        | 3+7/8in     | 7+1/2in      |
| (envpersonal   personalenvelope   personal)(width   length)     | Personal Envelope       | 3+5/8in     | 6+1/2in      |
| (usstandardfanfold   usstdfanfold)(width   length)              | US standard fanfold     | 14.875in    | 11in         |
| (germanstandardfanfold   germanstdfanfold)(width   length)      | German standard fanfold | 8.5in       | 12in         |
| (isoa0   a0   dina0)(width   length)                            | ISO/DIN & JIS A0        | 841mm       | 1189mm       |
| (isoa1   a1   dina1)(width   length)                            | ISO/DIN & JIS A1        | 594mm       | 841mm        |

Table 9-33. Paper Format Abbreviations (continued)

| Unit abbreviations                      | Unit Description  | Width value | Length value |
|-----------------------------------------|-------------------|-------------|--------------|
| (isoa2   a2   dina2)(width   length)    | ISO/DIN & JIS A2  | 420mm       | 594mm        |
| (isoa3   a3   dina3)(width   length)    | ISO/DIN & JIS A3  | 297mm       | 420mm        |
| (isoa4   a4   dina4)(width   length)    | ISO/DIN & JIS A4  | 210mm       | 297mm        |
| (isoa5   a5   dina5)(width   length)    | ISO/DIN & JIS A5  | 148mm       | 210mm        |
| (isoa6   a6   dina6)(width   length)    | ISO/DIN & JIS A6  | 105mm       | 148mm        |
| (isoa7   a7   dina7)(width   length)    | ISO/DIN & JIS A7  | 74mm        | 105mm        |
| (isoa8   a8   dina8)(width   length)    | ISO/DIN & JIS A8  | 52mm        | 74mm         |
| (isoa9   a9   dina9)(width   length)    | ISO/DIN & JIS A9  | 37mm        | 52mm         |
| (isoa10   a10   dina10)(width   length) | ISO/DIN & JIS A10 | 26mm        | 37mm         |
| (isob0   b0   dinb0)(width   length)    | ISO/DIN B0        | 1000mm      | 1414mm       |
| (isob1   b1   dinb1)(width   length)    | ISO/DIN B1        | 707mm       | 1000mm       |
| (isob2   b2   dinb2)(width   length)    | ISO/DIN B2        | 500mm       | 707mm        |
| (isob3   b3   dinb3)(width   length)    | ISO/DIN B3        | 353mm       | 500mm        |
| (isob4   b4   dinb4)(width   length)    | ISO/DIN B4        | 250mm       | 353mm        |
| (isob5   b5   dinb5)(width   length)    | ISO/DIN B5        | 176mm       | 250mm        |
| (isob6   b6   dinb6)(width   length)    | ISO/DIN B6        | 125mm       | 176mm        |
| (isob7   b7   dinb7)(width   length)    | ISO/DIN B7        | 88mm        | 125mm        |
| (isob8   b8   dinb8)(width   length)    | ISO/DIN B8        | 62mm        | 88mm         |
| (isob9   b9   dinb9)(width   length)    | ISO/DIN B9        | 44mm        | 62mm         |
| (isob10   b10   dinb10)(width   length) | ISO/DIN B10       | 31mm        | 44mm         |
| (isoc0   c0   dinc0)(width   length)    | ISO/DIN C0        | 917mm       | 1297mm       |
| (isoc1   c1   dinc1)(width   length)    | ISO/DIN C1        | 648mm       | 917mm        |
| (isoc2   c2   dinc2)(width   length)    | ISO/DIN C2        | 458mm       | 648mm        |

Table 9-33. Paper Format Abbreviations (continued)

| Unit abbreviations                      | Unit Description | Width value | Length value |
|-----------------------------------------|------------------|-------------|--------------|
| (isoc3   c3   dinc3)(width   length)    | ISO/DIN C3       | 324mm       | 458mm        |
| (isoc4   c4   dinc4)(width   length)    | ISO/DIN C4       | 229mm       | 324mm        |
| (isoc5   c5   dinc5)(width   length)    | ISO/DIN C5       | 162mm       | 229mm        |
| (isoc6   c6   dinc6)(width   length)    | ISO/DIN C6       | 114mm       | 162mm        |
| (isoc7   c7   dinc7)(width   length)    | ISO/DIN C7       | 81mm        | 114mm        |
| (isoc8   c8   dinc8)(width   length)    | ISO/DIN C8       | 57mm        | 81mm         |
| (isoc9   c9   dinc9)(width   length)    | ISO/DIN C9       | 40mm        | 57mm         |
| (isoc10   c10   dinc10)(width   length) | ISO/DIN C10      | 28mm        | 40mm         |
| jisb0(width   length)                   | JIS B0           | 1030mm      | 1456mm       |
| jisb1(width   length)                   | JIS B1           | 728mm       | 1030mm       |
| jisb2(width   length)                   | JIS B2           | 515mm       | 728mm        |
| jisb3(width   length)                   | JIS B3           | 364mm       | 515mm        |
| jisb4(width   length)                   | JIS B4           | 257mm       | 364mm        |
| jisb5(width   length)                   | JIS B5           | 182mm       | 257mm        |
| jisb6(width   length)                   | JIS B6           | 128mm       | 182mm        |
| jisb7(width   length)                   | JIS B7           | 91mm        | 128mm        |
| jisb8(width   length)                   | JIS B8           | 64mm        | 91mm         |
| jisb9(width   length)                   | JIS B9           | 45mm        | 64mm         |
| jisb10(width   length)                  | JIS B10          | 32mm        | 45mm         |

## RTL Class Reference

### Topics:

- “The Boolean Class”
- “The Color Class” on page 9-191
- “The Date Class” on page 9-195
- “The Enum Classes” on page 9-196
- “The FGLNumericVariable Class” on page 9-202
- “The FGLStringVariable Class” on page 9-203
- “The Numeric Class” on page 9-204
- “The Runtime Class” on page 9-213
- “The String Class” on page 9-215

## The Boolean Class

### Topics:

- Syntax

- Member Objects
- Usage

See also: RTL Expressions

## Syntax

Boolean

## Member Objects

Table 9-34. Member Objects (Static Member Variables)

| Name  | Description               |
|-------|---------------------------|
| TRUE  | Contains the value TRUE.  |
| FALSE | Contains the value FALSE. |

## Usage

Like all RTL Classes, the "new" keyword is not supported, so that it is not possible to create and subclass objects.

## Member Objects Usage

All expressions in the RTL expression language that contain relational operators return one of these objects, which are static member variables that do not require an object instance.

If you wish to specify TRUE or FALSE in a formula, the variables are prefixed with the Boolean classname and the '.' character.

## Example

Boolean.TRUE

## The Color Class

Topics:

- Syntax
- Member Objects
- Methods
- Usage

See also: RTL Expressions

## Syntax

Color

## Member Objects

These member variables do not require an object. They are prefixed by the Color class name. See Usage.

Table 9-35. Member Objects (Static Member Variables)

| Name  | Description                                     |
|-------|-------------------------------------------------|
| BLACK | Specifies a Color object having the color BLACK |

Table 9-35. Member Objects (Static Member Variables) (continued)

| Name       | Description                                          |
|------------|------------------------------------------------------|
| BLUE       | Specifies a Color object having the color BLUE       |
| CYAN       | Specifies a Color object having the color CYAN       |
| DARK_GRAY  | Specifies a Color object having the color DARK_GRAY  |
| GRAY       | Specifies a Color object having the color GRAY       |
| GREEN      | Specifies a Color object having the color GREEN      |
| LIGHT_GRAY | Specifies a Color object having the color LIGHT_GRAY |
| MAGENTA    | Specifies a Color object having the color MAGENTA    |
| ORANGE     | Specifies a Color object having the color ORANGE     |
| PINK       | Specifies a Color object having the color PINK       |
| RED        | Specifies a Color object having the color RED        |
| WHITE      | Specifies a Color object having the color WHITE      |
| YELLOW     | Specifies a Color object having the color YELLOW     |

## Methods

Class methods do not require an object. The method name is prefixed by the class name. Object methods require an object reference. See Usage.

Table 9-36. Class Methods (Static Member Methods)

| Name                                                              | Description                                                                                                             |
|-------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| fromHSBA(<br>Numeric h,<br>Numeric s,<br>Numeric b)               | Returns a Color object based on the specified values of hue, saturation, and brightness for the HSB color model.        |
| fromHSBA(<br>Numeric h,<br>Numeric s,<br>Numeric b,<br>Numeric a) | Returns a Color object based on the specified values of hue, saturation, brightness, and alpha for the HSB color model. |
| fromRGBA(<br>Numeric r,<br>Numeric g,<br>Numeric b)               | Returns an opaque sRGB Color object with the specified red, green, and blue values.                                     |
| fromRGBA(<br>Numeric r,<br>Numeric g,<br>Numeric b,<br>Numeric a) | Returns an sRGB Color object with the specified red, green, blue, and alpha values.                                     |



Table 9-37. Object Methods

| Name            | Description                                                                   |
|-----------------|-------------------------------------------------------------------------------|
| brighter()      | Brightens the Color object                                                    |
| darker()        | Darkens the Color object                                                      |
| getAlpha()      | Returns the alpha component in the range 0-255                                |
| getBrightness() | Returns the brightness value of the value in the HSB color model              |
| getBlue()       | Returns the blue component in the range 0-255 in the default sRGB space       |
| getGreen()      | Returns the green component in the range 0-255 in the default sRGB space      |
| getRGBA()       | Returns the RGBA value representing the color in the default sRGB color model |
| getHue()        | Returns the hue value of the value in the HSB color model                     |
| getRed()        | Returns the red component in the range 0-255 in the default sRGB space        |
| getSaturation() | Returns the saturation value of the value in the HSB color model              |
| toString()      | Return a string representation in the form "#argb"                            |

## Usage

Like all RTL Classes, the "new" keyword is not supported, so that it is not possible to create and subclass objects.

The default color for a report item, such as a LayoutNode, is black. You can change the color of the item by entering a value for the color or bgColor property in the Properties View. These properties are of type Color; Instead of selecting a color from the Edit Expressions color palette, you can use the members of this class in an expression that returns a Color object.

## Member Objects

These objects are static member variables that do not require an object instance. The variables are prefixed with the Color classname and the '.' character.

Example:

```
Color.RED
```

## Class Methods

These static methods do not require a Color object instance. When you invoke the method, it is prefixed with the Color classname and the '.' character.

```
Color.fromHSBA(h, s, b)
```

Creates a Color object based on the values for the HSB color model where:

- *h* (hue) is any floating-point number. The floor of this number is subtracted from it to create a fraction between 0 and 1, which is then multiplied by 360.
- *s* (saturation) is a floating-point value between zero and one (numbers within the range 0.0-1.0).

- *b* (brightness) is a floating-point value between zero and one (numbers within the range 0.0-1.0).

```
Color.fromHSBA(h, s, b, a)
```

Creates a Color object based on the values for the HSB color model where:

- *h* (hue) is any floating-point number. The floor of this number is subtracted from it to create a fraction between 0 and 1, which is then multiplied by 360.
- *s* (saturation) is a floating-point value between zero and one (numbers within the range 0.0-1.0).
- *b* (brightness) is a floating-point value between zero and one (numbers within the range 0.0-1.0).
- *a* (alpha) is the alpha component.

```
Color.fromRGBA(r, g, b)
```

Creates an opaque sRGB Color object based on the specified values where:

- *r* (red) is within the range (0.0 - 255)
- *g* (green) is within the range (0.0 - 255)
- *b* (blue) is within the range (0.0 - 255)

Alpha defaults to 1.0. The color used depends on the best match from the colors available for the output device.

```
fromRGBA(r, g, b a)
```

Creates an sRGB Color object based on the specified values where:

- *r* (red) is within the range (0.0 - 255)
- *g* (green) is within the range (0.0 - 255)
- *b* (blue) is within the range (0.0 - 255)
- *a* (alpha) is within the range (0.0 - 255)

The color used depends on the best match from the colors available for the output device.

Example:

```
Color.fromRGBA(0.5,0.5,0.5,0.5)
```

## Object Methods

These methods are called on a Color object instance. When you invoke the method, it is prefixed with the object instance name and the "." character.

Example:

```
Color.RED.darker()
brighter()
```

Creates a brighter version of the Color object by applying an arbitrary scale factor to each of the RGB components. Invoking a series of invocations of the darker and brighter methods might give an inconsistent result because of rounding errors.

```
darker()
```

Creates a darker version of the Color object by applying an arbitrary scale factor to each of the RGB components. Invoking a series of invocations of the darker and brighter methods might give an inconsistent result because of rounding errors.

```
getHue()
```

Returns the Numeric hue value of the value in the HSB color model .

`getSaturation()`

Returns the Numeric saturation value of the value in the HSB color model .

`getBrightness()`

Returns the Numeric brightness value of the value in the HSB color model.

`getRed()`

Returns the Numeric red component in the range 0-255 in the default sRGB space.

See Also: `getRGBA()`

`getGreen()`

Returns the Numeric green component in the range 0-255 in the default sRGB space.

See Also: `getRGBA()`

`getBlue()`

Returns the Numeric blue component in the range 0-255 in the default sRGB space.

See Also: `getRGBA()`

`getAlpha()`

Returns the Numeric alpha component in the range 0-255.

See Also: `getRGBA()`

`getRGBA()`

Returns the Numeric RGB value representing the color in the default sRGB color model. (Bits 24-31 are alpha, 16-23 are red, 8-15 are green, 0-7 are blue).

See Also: `getRed()`, `getGreen()`, `getBlue()`

`toString()`

Returns a string representation in the form "#argb"

## The Date Class

The Date class provides methods for date formatting and parsing.

See also: RTL Expressions

### Syntax

Date

### Methods

Class methods are static methods that do not require an object. The method name is prefixed by the class name. See Usage.

Table 9-38. Class Methods (Static Member Methods)

| Name                                                  | Description                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fromIsoValue(String value)</code>               | Constructs a Date from the date value; returns a Date instance representing the value. The value is expected in iso format ("YYYY-MM-DD").<br><br>Using the function guarantees that the Date value is constructed correctly without the danger of runtime parse errors due to changing 4GL date formatting or locale settings . |
| <code>parseString(String value, String format)</code> | Constructs a Date from <i>value</i> . The parsing is based on the passed <i>format</i> pattern. See Formatting Symbols for Dates.                                                                                                                                                                                                |
| <code>format(java.lang.String.format)</code>          | Formats a Date as a String according to a format specification. The format specification is 4GL-compatible. Returns a string representation of the date. See Formatting Symbols for Dates.                                                                                                                                       |
| <code>today()</code>                                  | Constructs a Date from the current date value. Returns a Date instance representing the current date.                                                                                                                                                                                                                            |

## Usage

Like all RTL Classes, the "new" keyword is not supported, so it is not possible to create and subclass objects.

These static methods do not require a Date object instance. When you invoke the method, it is prefixed with the **Date** classname and the '.' For example, the following expression uses the `isoValue` of the variable `orderline.orders.orderdate` to create a valid date for this value in the specified format:

```
Date.fromIsoValue(orderline.orders.orderdate.isoValue).format("DDD DD MMM YYYY")
```

## Formatting Symbols for Dates

Table 9-39. Formatting Symbols for Dates

| Character         | Description                                                                               |
|-------------------|-------------------------------------------------------------------------------------------|
| <code>dd</code>   | Day of the month as a two-digit integer.                                                  |
| <code>ddd</code>  | Three-letter English-language abbreviation of the day of the week, for example, Mon, Tue. |
| <code>mm</code>   | Month as a two-digit integer.                                                             |
| <code>mmm</code>  | Three-letter English-language abbreviation of the month, for example, Jan, Feb.           |
| <code>yy</code>   | Year, as a two-digit integer representing the two trailing digits.                        |
| <code>yyyy</code> | Year as a four-digit number.                                                              |

## The Enum Classes

Topics:

These classes provide the list of valid values for form item properties that are of type Enum.

- Alignment
- TextAlignment
- BaselineType
- LayoutDirection
- Y-SizeAdjustment
- PageNoFormat
- TrimText
- X-SizeAdjustment
- Floating Behavior
- Section
- XYChartDrawAs
- MapChartDrawAs
- CategoryChartDrawAs
- CodeType
- BorderStyles classes

## The Alignment Class

### Syntax

Alignment

The class consists of a set of public static member variables that represent the valid values for the alignment property.

*Table 9-40. Member Objects (Static Member Variables)*

| Name     |
|----------|
| Baseline |
| Center   |
| Far      |
| Near     |
| None     |

## The TextAlignment Class

### Syntax

TextAlignment

The class consists of a set of public static member variables that represent the valid values for the textAlignment property.

*Table 9-41. Member Objects (Static Member Variables)*

| Name   |
|--------|
| Center |
| Left   |
| Right  |

## The BaselineType Class

### Syntax

BaselineType

The class consists of a set of public static member variables that represent the valid values for the baselineType property.

*Table 9-42. Member Objects (Static Member Variables)*

| Name       |
|------------|
| Leftleft   |
| Leftright  |
| Rightleft  |
| Rightright |

## The LayoutDirection Class

### Syntax

LayoutDirection

The class consists of a set of public static member variables that represent the valid values for the layoutDirection property.

*Table 9-43. Member Objects (Static Member Variables)*

| Name              |
|-------------------|
| BottomToTop       |
| HorizontalNatural |
| LeftToRight       |
| RightToLeft       |
| Swapped           |
| TopToBottom       |
| TurnLeft          |
| TurnRight         |
| Unturned          |
| UpsideDown        |
| VerticalNatural   |
| VerticalUnnatural |

## The Y-SizeAdjustment Class

### Syntax

Y-SizeAdjustment

The class consists of a set of public static member variables that represent the valid values for the Y-SizeAdjustment property.

*Table 9-44. Member Objects (Static Member Variables)*

| Name             |
|------------------|
| ExpandtoParent   |
| ShrinktoChildren |

## The PageNoFormat Class

### Syntax

PageNoFormat

The class consists of a set of public static member variables that represent the valid values for the pageNoFormat property.

*Table 9-45. Member Objects (Static Member Variables)*

| Name       |
|------------|
| Arabic     |
| Lowerroman |
| Upperroman |

## The TrimText Class

### Syntax

TrimText

The class consists of a set of public static member variables that represent the valid values for the trimText property.

*Table 9-46. Member Objects (Static Member Variables)*

| Name     |
|----------|
| Both     |
| Compress |
| Left     |
| Right    |

## The X-SizeAdjustment Class

### Syntax

X-SizeAdjustment

The class consists of a set of public static member variables that represent the valid values for the X-SizeAdjustment property.

*Table 9-47. Member Objects (Static Member Variables)*

| Name             |
|------------------|
| ExpandtoParent   |
| ShrinktoChildren |

## Floating Behavior Class

### Syntax

FloatingBehavior

The class consists of a set of public static member variables that represent the valid values for the property.

*Table 9-48. Member Objects (Static Member Variables)*

| Name     |
|----------|
| Encloses |

Table 9-48. Member Objects (Static Member Variables) (continued)

| Name |
|------|
| Free |

## Section Class

### Syntax

Section

The class consists of a set of public static member variables that represent the valid values for the property.

Table 9-49. Member Objects (Static Member Variables)

| Name            |
|-----------------|
| AnyPageFooter   |
| AnyPageHeader   |
| EvenPageHeader  |
| EvenPageFooter  |
| FirstPageHeader |
| FirstPageFooter |
| OddPageFooter   |
| OddPageHeader   |
| ItemSeparator   |

## XYChartDrawAs Class

### Syntax

XYChartDrawAs

The class consists of a set of public static member variables that represent the valid values for the property.

Table 9-50. Member Objects (Static Member Variables)

| Name        |
|-------------|
| Polar       |
| Scatter     |
| Area        |
| StackedArea |
| Line        |
| Step        |
| StepArea    |
| TimeSeries  |

## MapChartDrawAs Class

### Syntax

MapChartDrawAs

The class consists of a set of public static member variables that represent the valid values for the property.



Table 9-51. Member Objects (Static Member Variables)

| Name  |
|-------|
| Bar   |
| Bar3D |
| Pie   |
| Pie3D |
| Ring  |

## CategoryChartDrawAs Class

### Syntax

CategoryChartDrawAs

The class consists of a set of public static member variables that represent the valid values for the property.

Table 9-52. Member Objects (Static Member Variables)

| Name        |
|-------------|
| Area        |
| Bar         |
| Bar3D       |
| Line        |
| Line3D      |
| StackedArea |
| StackedBar  |
| Waterfall   |

## CodeType Class

### Syntax

CodeType

The class consists of a set of public static member variables that represent the valid values for the property.

Table 9-53. Member Objects (Static Member Variables)

| Name                 |
|----------------------|
| Codabar_18           |
| Codabar_2            |
| Code_11_matrix       |
| Code_11_matrix       |
| Code_2_5_datalogic   |
| Code_2_5_IATA        |
| Code_2_5_industrial  |
| Code_2_5_interleaved |
| Code_2_5_inverted    |
| Code_2_5_matrix      |

Table 9-53. Member Objects (Static Member Variables) (continued)

| Name               |
|--------------------|
| Code_32            |
| Code_39            |
| Code_39_extended   |
| Code_93            |
| Code_93_extended   |
| Code_BCD_matrix    |
| Ean_13             |
| Ean_8              |
| Ean_supplemental_2 |
| Ean_supplemental_5 |
| Pdf_417            |
| Upc_a              |
| Upc_e              |
| Upc_supplemental_2 |
| Upc_supplemental_5 |

## BorderStyles Classes

### Syntax

See Table 9-54.

The attributes **borderStyle**, **borderTopStyle**, **borderRightStyle**, **borderBottomStyle** and **borderLeftStyle** have the following set of public member variables that represent the valid values for the property:

Table 9-54. Member Objects (Static Member Variables)

| Name   |
|--------|
| Dashed |
| Dotted |
| Double |
| Groove |
| Inset  |
| Outset |
| Ridge  |
| Solid  |

## The FGLNumericVariable Class

### Topics:

- Syntax
- Member Objects
- Methods
- Usage

For every 4GL numeric variable (INTEGER, SMALLINT, FLOAT, SMALLFLOAT, DECIMAL and MONEY) of report data, an object is created that is an instance of an FGLNumericVariable. These objects hold the value of the 4GL variable.

## Syntax

FGLNumericVariable

## Member Objects

The following fields are defined:

*Table 9-55. Member Objects (Member Variables)*

| Name     | Description                                                                                                |
|----------|------------------------------------------------------------------------------------------------------------|
| value    | A Numeric containing the value of the field; this is the same as the value of the 4GL variable. See Usage. |
| fglValue | A String containing the value of the field as formatted by the DVM. See Usage.                             |
| name     | A String specifying the name of the field.                                                                 |
| caption  | A String specifying the title of the field.                                                                |
| type     | A String specifying the type of the field.                                                                 |

## Methods

This subclass inherits the public methods of the Numeric class.

## Usage

In addition to the value of the 4GL variable, these objects contain member variables, among which is the variable value also containing the numeric value. For this reason, it is legal to write "order\_line.itemprice" in your expression as a shortcut for "order\_line.itemprice.value".

The member variable fglValue contains a String representing the numeric value as formatted by the DVM, taking into consideration such parameters as USING, DBMONEY, etc. In contrast, the member variable value is a numeric value without formatting.

## The FGLStringVariable Class

Topics:

- Syntax
- Member Objects
- Methods
- Usage

For every CHAR, VARCHAR, STRING, TEXT, DATE, DATETIME and INTERVAL 4GL variable of report data, an object is created that is an instance of an FGLStringVariable. These objects hold the value of the 4GL variable.

## Syntax

FGLStringVariable

## Member Objects

The following fields are defined:

Table 9-56. Member Objects (Member Variables)

| Name    | Description                                                                                               |
|---------|-----------------------------------------------------------------------------------------------------------|
| value   | A String containing the value of the field; this is the same as the value of the 4GL variable. See Usage. |
| name    | A String specifying the name of the field.                                                                |
| caption | A String specifying the title of the field.                                                               |
| type    | A String specifying the type of the field.                                                                |

## Methods

This subclass inherits the public methods of the String class.

## Usage

In addition to the value of the 4GL variable, these objects contain a member variable "value" which also contains the value. For this reason, it is legal to write "order\_line.billState" in your expression as a shortcut for "order\_line.billState.value".

## The Numeric Class

These topics document the Numeric class and its public members. Values for this data type are limited to 15 significant digits.

### Topics:

- Syntax
- Methods
- Usage

See also: RTL Expressions

## Syntax

Numeric

## Methods

Table 9-57. Object Methods

| Name             | Description                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------------------|
| abs()            | Returns the absolute value.                                                                                                  |
| atan2(Numeric x) | Returns the angle <i>theta</i> from the conversion of rectangular coordinates (x, y) to polar coordinates (r, <i>theta</i> ) |
| byteValue()      | Returns the value converted to a byte.                                                                                       |
| cbrt()           | Returns the cube root of a numeric value.                                                                                    |
| ceil()           | Returns the smallest numeric value that is greater than or equal to the value and is equal to a mathematical integer.        |
| cos()            | Returns the trigonometric cosine of an angle.                                                                                |
| cosh()           | Returns the hyperbolic cosine of a numeric value.                                                                            |

Table 9-57. Object Methods (continued)

| Name                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>exp()</code>                           | Returns the base -e exponential.                                                                                                                                                                                                                                                                                                                                                                                     |
| <code>floor()</code>                         | Returns the largest numeric value that is less than or equal to the value and is equal to a mathematical integer.                                                                                                                                                                                                                                                                                                    |
| <code>format("format-string")</code>         | <p>Converts the value to a string representation defined by a format string.</p> <p>For example, for DECIMAL and FLOAT data types, <i>format-string</i> consists of pound signs (#) that represent digits and a decimal point; that is, "###.##" produces three places to the left of the decimal point and exactly two to the right. Additional options for the string are listed in the Usage section, format.</p> |
| <code>getExponent()</code>                   | Returns the unbiased exponent used in the representation of a numeric value.                                                                                                                                                                                                                                                                                                                                         |
| <code>intValue()</code>                      | Returns the value converted to an integer.                                                                                                                                                                                                                                                                                                                                                                           |
| <code>isInfinite()</code>                    | Returns TRUE if the value has infinite value                                                                                                                                                                                                                                                                                                                                                                         |
| <code>isNaN()</code>                         | Returns TRUE if the value is not a number                                                                                                                                                                                                                                                                                                                                                                            |
| <code>log()</code>                           | Returns the natural logarithm (base e) of a numeric value.                                                                                                                                                                                                                                                                                                                                                           |
| <code>log10()</code>                         | Returns the base 10 logarithm of a numeric value.                                                                                                                                                                                                                                                                                                                                                                    |
| <code>max(Numeric b)</code>                  | Returns the greater of two values.                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>min(Numeric b)</code>                  | Returns the smaller of two values.                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>rint()</code>                          | Returns the Numeric value that is closest in value to the value and is equal to a mathematical integer.                                                                                                                                                                                                                                                                                                              |
| <code>round()</code>                         | Returns the closes integer to the value.                                                                                                                                                                                                                                                                                                                                                                             |
| <code>signum()</code>                        | Returns the signum function of the value; zero if the value is zero, 1.0 if the value is greater than zero, -1.0 if the value is less than zero.                                                                                                                                                                                                                                                                     |
| <code>sin()</code>                           | Returns the trigonometric sine, measured in radians.                                                                                                                                                                                                                                                                                                                                                                 |
| <code>sinh()</code>                          | Returns the hyperbolic sine of a numeric value.                                                                                                                                                                                                                                                                                                                                                                      |
| <code>sqrt()</code>                          | Returns the tangent of an angle measured in radians.                                                                                                                                                                                                                                                                                                                                                                 |
| <code>tan()</code>                           | Returns the tangent of an angle measured in radians.                                                                                                                                                                                                                                                                                                                                                                 |
| <code>tanh()</code>                          | Returns the hyperbolic tangent of a numeric value.                                                                                                                                                                                                                                                                                                                                                                   |
| <code>toChar()</code>                        | Returns the unicode character representation of a numeric value.                                                                                                                                                                                                                                                                                                                                                     |
| <code>getPhysicalPageNumber()</code>         | Gets the current page number of the physical page.                                                                                                                                                                                                                                                                                                                                                                   |
| <code>getTotalNumberOfPhysicalPages()</code> | Gets the total number of physical pages.                                                                                                                                                                                                                                                                                                                                                                             |

Table 9-57. Object Methods (continued)

| Name                                           | Description                                            |
|------------------------------------------------|--------------------------------------------------------|
| getPageNumber(String <i>pageName</i> )         | Gets the page number of the specified page.            |
| getTotalNumberOfPages(String <i>pageName</i> ) | Gets the total number of pages for the specified page. |

## Usage

**Important:** This data type is limited to 15 significant digits.

Like all RTL Classes, the "new" keyword is not supported, so that it is not possible to create and subclass objects.

### abs()

abs()

Returns the absolute value of an int value:

- If the value is not negative, the value is returned.
- If the value is negative, the negation of the value is returned.
- If the value is equal to the value of Integer.MIN\_VALUE, the most negative representable int value, the result is that same value, which is negative.

### cos()

cos()

Returns the trigonometric cosine of an angle. However, If the value is NaN or an infinity, then the result is NaN.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

### sin()

sin()

Returns the trigonometric sine of an angle. However,

- If the value is NaN or an infinity, then the result is NaN.
- If the value is zero, then the result is a zero with the same sign as the value.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

### tan()

tan()

Returns the trigonometric tangent of an angle. Special cases:

- If the value is NaN or an infinity, then the result is NaN.
- If the value is zero, then the result is a zero with the same sign as the value.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

## **byteValue()**

`byteValue()`

Returns the value converted to a byte .

## **cbrt()**

`cbrt()`

Returns the cube root of a double value. For positive finite  $x$ ,  $\text{cbrt}(-x) == -\text{cbrt}(x)$ ; that is, the cube root of a negative value is the negative of the cube root of that value's magnitude.

Special cases:

- If the value is NaN, then the result is NaN.
- If the value is infinite, then the result is an infinity with the same sign as the value.
- If the value is zero, then the result is a zero with the same sign as the value.

The computed result must be within 1 ulp of the exact result.

## **ceil()**

`ceil()`

Returns the smallest (closest to negative infinity) double value that is greater than or equal to the value and is equal to a mathematical integer.

Special cases:

- If the value is already equal to a mathematical integer, then the result is the same as the value.
- If the value is NaN or an infinity or positive zero or negative zero, then the result is the same as the value.
- If the value is less than zero but greater than -1.0, then the result is negative zero.

Note that the value of `Math.ceil(x)` is exactly the value of `-Math.floor(-x)`.

## **cosh()**

`cosh()`

Returns the hyperbolic cosine of a double value. The hyperbolic cosine of  $x$  is defined to be  $(e^x + e^{-x})/2$  where  $e$  is Euler's number.

Special cases:

- If the value is NaN, then the result is NaN.
- If the value is infinite, then the result is positive infinity.
- If the value is zero, then the result is 1.0.

The computed result must be within 2.5 ulps of the exact result.

## **exp()**

`exp()`

Returns Euler's number  $e$  raised to the power of a double value. Special cases:

- If the value is NaN, the result is NaN.
- If the value is positive infinity, then the result is positive infinity.
- If the value is negative infinity, then the result is positive zero.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

### **floor()**

`floor()`

Returns the largest (closest to positive infinity) double value that is less than or equal to the value and is equal to a mathematical integer.

Special cases:

- If the value is already equal to a mathematical integer, then the result is the same as the value.
- If the value is NaN or an infinity or positive zero or negative zero, then the result is the same as the value.

### **getExponent()**

`getExponent()`

Returns the unbiased exponent used in the representation of a float.

Special cases:

- If the value is NaN or infinite, then the result is `Float.MAX_EXPONENT + 1`.
- If the value is zero or subnormal, then the result is `Float.MIN_EXPONENT - 1`.

### **intValue()**

`intValue()`

returns the value converted to an integer (signed 32 bit)

### **isInfinite()**

`isInfinite()`

returns the Boolean value true in case the value has the infinite value

### **isNaN()**

`isNaN()`

returns the Boolean value true in case that the value is not a number

### **log()**

`log()`

Returns the natural logarithm (base e) of a double value.

Special cases:

- If the value is NaN or less than zero, then the result is NaN.
- If the value is positive infinity, then the result is positive infinity.
- If the value is positive zero or negative zero, then the result is negative infinity.



The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

## **log10()**

log10()

Returns the base 10 logarithm of a double value.

Special cases:

- If the value is NaN or less than zero, then the result is NaN.
- If the value is positive infinity, then the result is positive infinity.
- If the value is positive zero or negative zero, then the result is negative infinity.
- If the value is equal to  $10^n$  for integer  $n$ , then the result is  $n$ .

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

## **rint()**

rint()

Returns the double (floating-point) value that is closest in value to the value and is equal to a mathematical integer. If two double values that are mathematical integers are equally close, the result is the integer value that is even.

Special cases:

- If the value is already equal to a mathematical integer, then the result is the same as the value.
- If the value is NaN or an infinity or positive zero or negative zero, then the result is the same as the value.

Returns:

the closest floating-point value to a that is equal to a mathematical integer.

## **round()**

round()

Returns the closest int to the value. The result is rounded to an integer by adding  $1/2$ , taking the floor of the result, and casting the result to type int. In other words, the result is equal to the value of the expression:

```
(int)Math.floor(a + 0.5f)
```

Special cases:

- If the value is NaN, the result is 0.
- If the value is negative infinity or any value less than or equal to the value of Integer.MIN\_VALUE, the result is equal to the value of Integer.MIN\_VALUE.
- If the value is positive infinity or any value greater than or equal to the value of Integer.MAX\_VALUE, the result is equal to the value of Integer.MAX\_VALUE.

See Also:

Integer.MAX\_VALUE, Integer.MIN\_VALUE

## **signum ()**

signum ()

Returns the signum function of the value; zero if the value is zero, 1.0 if the value is greater than zero, -1.0 if the value is less than zero.

Special Cases:

- If the value is NaN, then the result is NaN.
- If the value is positive zero or negative zero, then the result is the same as the value.

Returns:

the signum function of the value

## **sinh()**

sinh()

Returns the hyperbolic sine of a double value. The hyperbolic sine of  $x$  is defined to be  $(e^x - e^{-x})/2$  where  $e$  is Euler's number.

Special cases:

- If the value is NaN, then the result is NaN.
- If the value is infinite, then the result is an infinity with the same sign as the value.
- If the value is zero, then the result is a zero with the same sign as the value.

The computed result must be within 2.5 ulps of the exact result.

## **sqrt()**

sqrt()

Returns the correctly rounded positive square root of a double value.

Special cases:

- If the value is NaN or less than zero, then the result is NaN.
- If the value is positive infinity, then the result is positive infinity.
- If the value is positive zero or negative zero, then the result is the same as the value.

Otherwise, the result is the double value closest to the true mathematical square root of the value

## **tanh()**

tanh()

Returns the hyperbolic tangent of a double value. The hyperbolic tangent of  $x$  is defined to be  $(e^x - e^{-x})/(e^x + e^{-x})$ , in other words,  $\sinh(x)/\cosh(x)$ . Note that the absolute value of the exact tanh is always less than 1.

Special cases:

- If the value is NaN, then the result is NaN.
- If the value is zero, then the result is a zero with the same sign as the value.

- If the value is positive infinity, then the result is +1.0.
- If the value is negative infinity, then the result is -1.0.

The computed result must be within 2.5 ulps of the exact result. The result of tanh for any finite input must have an absolute value less than or equal to 1. Note that once the exact result of tanh is within 1/2 of an ulp of the limit value of  $\pm 1$ , correctly signed  $\pm 1.0$  should be returned.

### **toBoolean()**

toBoolean()

Returns the Boolean false in case that value is 0. Returns true for any other value.

### **toColor()**

toColor()

Returns a color object. The value is interpreted as a RGB integer.

### **toDegrees()**

toDegrees()

Converts the value from radians to degrees.

### **toRadians()**

toRadians()

Converts the value from degrees to radians.

### **toString ()**

toString ()

Converts the value to a string representation

### **toChar()**

toChar()

Converts the value to a unicode character representation (e.g. 65.toChar yields "A").

### **atan2(Numeric x)**

atan2(Numeric x)

Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta). This method computes the phase theta by computing an arc tangent of y/x in the range of -pi to pi.

Special cases:

- If either value is NaN, then the result is NaN.
- If the value is positive zero and the argument is positive, or the value is positive and finite and the argument is positive infinity, then the result is positive zero.
- If the value is negative zero and the argument is positive, or the value is negative and finite and the argument is positive infinity, then the result is negative zero.

- If the value is positive zero and the argument is negative, or the value is positive and finite and the argument is negative infinity, then the result is the double value closest to  $\pi$ .
- If the value is negative zero and the argument is negative, or the value is negative and finite and the argument is negative infinity, then the result is the double value closest to  $-\pi$ .
- If the value is positive and the argument is positive zero or negative zero, or the value is positive infinity and the argument is finite, then the result is the double value closest to  $\pi/2$ .
- If the value is negative and the argument is positive zero or negative zero, or the value is negative infinity and the argument is finite, then the result is the double value closest to  $-\pi/2$ .
- If both value and argument are positive infinity, then the result is the double value closest to  $\pi/4$ .
- If the value is positive infinity and the argument is negative infinity, then the result is the double value closest to  $3\pi/4$ .
- If the value is negative infinity and the argument is positive infinity, then the result is the double value closest to  $-\pi/4$ .
- If both value and argument are negative infinity, then the result is the double value closest to  $-3\pi/4$ .

The computed result must be within 2 ulps of the exact result. Results must be semi-monotonic.

Parameters:

$x$  - the abscissa coordinate

### **max(Numeric $b$ )**

`max(Numeric  $b$ )`

Returns the greater of two int values. That is, the result is the argument closer to the value of `Integer.MAX_VALUE`. If the argument has the same value as the object's value the result is that same value.

Parameters:

$b$  - an argument.

See Also: `Long.MAX_VALUE`

### **min(Numeric $b$ )**

`min(Numeric  $b$ )`

Returns the smaller of two int values. That is, the result is the argument closer to the value of `Integer.MIN_VALUE`. If the argument has the same value as the object's value the result is that same value.

Parameters:

$b$  - an argument.

See Also: `Long.MIN_VALUE`

## **format("format-string")**

`format("format-string")`

Converts the value to a string representation defined by a format string. The format string syntax is compatible to the D4GL "USING" format string. The formatting takes the values of the environment variables DBFORMAT and DBMONEY into account.

*Table 9-58. Formatting symbols*

| Character  | Description                                                                                                                                                                                                                           |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *          | Fills with asterisks any position that would otherwise be blank.                                                                                                                                                                      |
| &          | Fills with zeros any position that would otherwise be blank.                                                                                                                                                                          |
| #          | This does not change any blank positions in the display.                                                                                                                                                                              |
| <          | Causes left alignment.                                                                                                                                                                                                                |
| , (comma)  | Defines the position of the thousands separator. The thousands separator is not displayed if there are no digits to the left. By default, the thousands separator is a comma, but it can be another character as defined by DBFORMAT. |
| . (period) | Defines the position of the decimal separator. Only a single decimal separator may be specified. By default, the decimal separator is a period, however it can be another character as defined by DBMONEY or DBFORMAT.                |
| -          | Displays a minus sign for negative numbers.                                                                                                                                                                                           |
| \$         | This is the placeholder for the front specification of DBMONEY or DBFORMAT.                                                                                                                                                           |
| (          | Displayed as left parentheses for negative numbers (accounting parentheses).                                                                                                                                                          |
| )          | Displayed as right parentheses for negative numbers (accounting parentheses).                                                                                                                                                         |

## **Examples**

1. The following example converts the number 65 to "A", its unicode character representation: `65.toChar()`
2. This example formats the numeric value of overall total as a string:  
`overalltotal.format("###.##")`

## **The Runtime Class**

The Runtime class provides methods that simplify the creation of dynamic designs that change behavior based on the runtime setup.

See also: RTL Expressions

## **Syntax**

Runtime

## Methods

Class methods are static methods that do not require an object. The method name is prefixed by the class name. See Usage.

Table 9-59. Class Methods (Static Member Methods)

| Name                     | Description                                                                                                                                                                                                                                                                                                 |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| getOutputDeviceName()    | Returns a STRING indicating the output device name selected in the API call fgl_report_selectDevice; can be one of "PDF", "XLS", "XLSX", "HTML", "Image", "Printer" or "SVG".<br><br>For more information on fgl_report_selectDevice, refer to the Genero Studio >> Report Writer documentation.            |
| producingPDFOutput()     | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "PDF"; otherwise FALSE.                                                                                                                                                                                       |
| producingXLSOutput()     | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "XLS"; otherwise FALSE.                                                                                                                                                                                       |
| producingXLSXOutput()    | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "XLSX"; otherwise FALSE.                                                                                                                                                                                      |
| producingExcelOutput()   | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "XLS" or "XLSX"; otherwise FALSE.                                                                                                                                                                             |
| producingSVGOutput()     | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "SVG"; otherwise FALSE.                                                                                                                                                                                       |
| producingHTMLOutput()    | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "HTML"; otherwise FALSE.                                                                                                                                                                                      |
| producingImageOutput()   | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "Image"; otherwise FALSE.                                                                                                                                                                                     |
| producingPrinterOutput() | BOOLEAN; returns TRUE if the device name selected in the API call fgl_report_selectDevice was "Printer"; otherwise FALSE.                                                                                                                                                                                   |
| producingForPreview()    | BOOLEAN; returns TRUE if preview was selected in the API call fgl_report_selectPreview; otherwise FALSE.                                                                                                                                                                                                    |
| getSVGPrinterName()      | Returns a STRING indicating the printer name specified in the API call fgl_report_setSVGPrinterName.                                                                                                                                                                                                        |
| getSVGPaperSource()      | Returns a STRING indicating the paper source specified in the API call fgl_report_setSVGPaperSource; can be one of the following:<br><br>"Auto", "Cassette", "Envelope", "EnvelopeManual", "FormSource", "LargeCapacity", "LargeFormat", "Lower", "Middle", "Manual", "OnlyOne", "Tractor" or "SmallFormat" |
| xlsPagesAreMerged()      | BOOLEAN, returns TRUE if page merging was selected in the API call fgl_report_configureXLSDevice; otherwise FALSE.                                                                                                                                                                                          |
| getPrinterName()         | Returns a STRING indicating the printer name specified in the API call fgl_report_setPrinterName.                                                                                                                                                                                                           |

Table 9-59. Class Methods (Static Member Methods) (continued)

| Name                                                     | Description                                                                                                                 |
|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>getPrinterMediaSizeName()</code>                   | Returns a STRING indicating the media size name specified in the API call <code>fgl_report_setPrinterMediaSizeName</code> . |
| <code>getPrinterMediaTray()</code>                       | Returns a STRING indicating the media tray name specified in the API call <code>fgl_report_setPrinterMediaTray</code> .     |
| <code>getPrinterMediaName()</code>                       | Returns a STRING indicating the media name specified in the API call <code>fgl_report_setPrinterMediaName</code> .          |
| <code>getDebugLevel()</code>                             | Numeric, returns the current debug level specified in the environment variable GREDBUG, or 0 if no debug level was set.     |
| <code>getEnvironmentVariable(String variableName)</code> | Returns a String containing the value of the specified environment variable                                                 |

## Usage

The class provides a number of methods that simplify the job of creating dynamic designs that change behavior based on the runtime setup.

Like all RTL Classes, the "new" keyword is not supported, so it is not possible to create and subclass objects.

The methods can be used from within RTL expressions. Some common uses might be:

- To suppress headers and footers when Excel output is selected.
- To conditionally Insert a logo based on the printer tray is selected.
- To set a background color when debugging is enabled

These static methods do not require a Runtime object instance. When you invoke the method, it is prefixed with the Runtime class name and the '.' For example, you can suppress a header by setting its "visibilityCondition" property to the following expression:

```
"!Runtime.producingExcelOutput()"
```

## The String Class

These topics document the String class and its public members.

### Topics:

- Syntax
- Methods
- Usage and Examples

See also: RTL Expressions

## Syntax

String

## Methods

Table 9-60. Object Methods

| Name                                 | Description                                                                                                                                                                                                             |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| charAt(Numeric index)                | Returns a String containing the character at the specified <i>index</i> in the current String.                                                                                                                          |
| contains(String s)                   | Returns a Boolean value (TRUE/FALSE) specifying whether <i>s</i> is contained within the current String.                                                                                                                |
| endsWith(String s)                   | Returns a Boolean value (TRUE/FALSE) specifying whether the current String ends with <i>s</i> .                                                                                                                         |
| equals( String s)                    | Returns a Boolean value (TRUE/FALSE) specifying whether <i>s</i> matches the current String. If one of the Strings is null the method returns FALSE.                                                                    |
| equalsIgnoreCase( String s)          | Returns a Boolean value (TRUE/FALSE) specifying whether <i>s</i> matches the current String, ignoring character case. If one of the Strings is null the method returns FALSE.                                           |
| format (Numeric number, Enum format) | Sets the format of the page number string for a PAGENOBOX. The value for format can be: ARABIC, LOWERROMAN, UPPERROMAN                                                                                                  |
| indexOf( String s)                   | Returns a Numeric value representing the index of <i>s</i> within the current String.                                                                                                                                   |
| indexOf( String s, Numeric index )   | Returns a Numeric value representing the index of <i>s</i> within the current String, starting from byte position <i>index</i> . Returns zero if the substring was not found. Returns -1 if the current String is null. |
| isEmpty()                            | Returns a BOOLEAN value (TRUE/FALSE) specifying whether the current String has a length of zero.                                                                                                                        |
| lastIndexOf(String s)                | Returns a Numeric value representing the index within the current String of the last occurrence of <i>s</i> , searching backward.                                                                                       |
| lastIndexOf(String s, Numeric index) | Returns a Numeric value representing the index within the current String of the last occurrence of <i>s</i> , searching backward. starting at the specified position <i>index</i> .                                     |
| length( )                            | Returns a Numeric value representing the length of the current String.                                                                                                                                                  |
| matches(String s)                    | Returns a Boolean value specifying whether the current String matches the regular expressions.                                                                                                                          |
| replace(String old, String new)      | Replaces <i>old</i> in the current String with <i>new</i> .                                                                                                                                                             |
| replaceAll(String old, String new)   | Replaces every occurrence of <i>old</i> in the current String with <i>new</i> .                                                                                                                                         |
| replaceFirst(String old, String new) | Replaces every occurrence of <i>old</i> in the current String with <i>new</i> .                                                                                                                                         |
| startsWith(String s)                 | Returns a Boolean value specifying whether the current String begins with <i>s</i> .                                                                                                                                    |



Table 9-60. Object Methods (continued)

| Name                                                    | Description                                                                                                                                                                            |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>startsWith(String s, Numeric index)</code>        | Returns a Boolean value specifying whether the substring in the current String beginning at the position <i>index</i> contains <i>s</i> . The first byte of a string is at position 0. |
| <code>subString( Numeric index, Numeric eindex )</code> | Returns the substring starting at byte position <i>sindex</i> and ending at the character at <i>eindex</i> -1. The first byte of a string is at position 0.                            |
| <code>substring(Numeric index)</code>                   | Returns the substring starting at the position <i>index</i> in the current String. The first byte of a string is at position 0.                                                        |
| <code>toLowerCase( )</code>                             | Converts the current String to lowercase.                                                                                                                                              |
| <code>toString()</code>                                 | Converts the current Numeric value to a String.                                                                                                                                        |
| <code>toUpperCase( )</code>                             | Converts the current String to uppercase.                                                                                                                                              |
| <code>translate( )</code>                               | Uses the current String as the key for a lookup in Genero localization files; returns any entry found, otherwise returns the current String itself.                                    |
| <code>trim( )</code>                                    | Removes white space characters from the beginning and end of the current String.                                                                                                       |
| <code>trimCompress()</code>                             | Removes white space characters at the beginning and end of the current String, as well as any contained white space.                                                                   |
| <code>trimLeft( )</code>                                | Removes white space characters from the beginning of the current String.                                                                                                               |
| <code>trimRight( )</code>                               | Removes white space characters from the end of the current String.                                                                                                                     |
| <code>urlencode( )</code>                               | Returns a URL encoding of the current String.                                                                                                                                          |

## Usage and Examples

Like all RTL Classes, the "new" keyword is not supported, so that it is not possible to create and subclass objects.

All literal String values in an expression must be delimited by double quotes.

All the methods require an object instance. When you invoke the method, it is prefixed with the object instance name and the "." character. You can get an object instance by referencing a 4GL variable or by calling a method on another object. The object can be a literal value, for example:

```
"Test".length()
```

## Numeric data items in WordBoxes and WordWrapBoxes

If you enter an expression for the text property of a WordBox or WordWrapBox, the value must be a String. You can use the **toString()** function in your expressions to convert numbers to Strings. When you drag a Numeric data item onto the Report Design Window, it is automatically placed in a WordBox element, and an expression for the text property is created to convert it to a String.

For example:

```
order_line.unitprice.toString()
```

### The indexes of a String (example subString)

When specifying the character position (index) of a string, the first character value is at position 0.

For example, when using the **subString** function, the substring begins at the specified *startIndex* and ends at the character at *endindex* -1. The length of the string is *endIndex* minus *startIndex*:

```
order_line.billState.subString(1,5)
```

If the value of the String billState is "smiles" (indexes 012345), the substring returned is "mile", and the length of the string is 5 minus 1 = 4.

### Concatenating Strings

Use the + operator to concatenate strings.

Example:

```
("Total:"+" "+order_line.totalorderprice).toString()
```

This expression returns the current value of totalorderprice as part of a String value:

```
"Total: 12.95"
```

---

## Chapter 10. Code Analyzer

### Topics:

- Overview
- Sequence Diagram
- Dependency Diagram

---

### Genero Studio Code Analyzer

The **Code Analyzer** reverse engineers your existing applications and can generate the following diagrams, providing you with an overview of your application:

- Dependency Diagram - displays the complex relationships between application and its components, the dependencies between the various objects. To display the diagram, right-click the application node or the project node, and select **Open dependency diagram** from the menu of options.
- Sequence Diagram - displays the application's BDL functions and how they interact with each other (which functions call and/or are called by other functions). To display the diagram, right-click the function name and select **Open sequence diagram** from the menu of options.

---

### Sequence Diagrams

- Overview
  - Structure View
  - Sub calls menu
- Creating the Diagrams

#### Overview

This diagram documents the functions of the application and how they interact with each other - which functions call and/or are called by other functions. The diagram indicate the flow of your application logic visually:

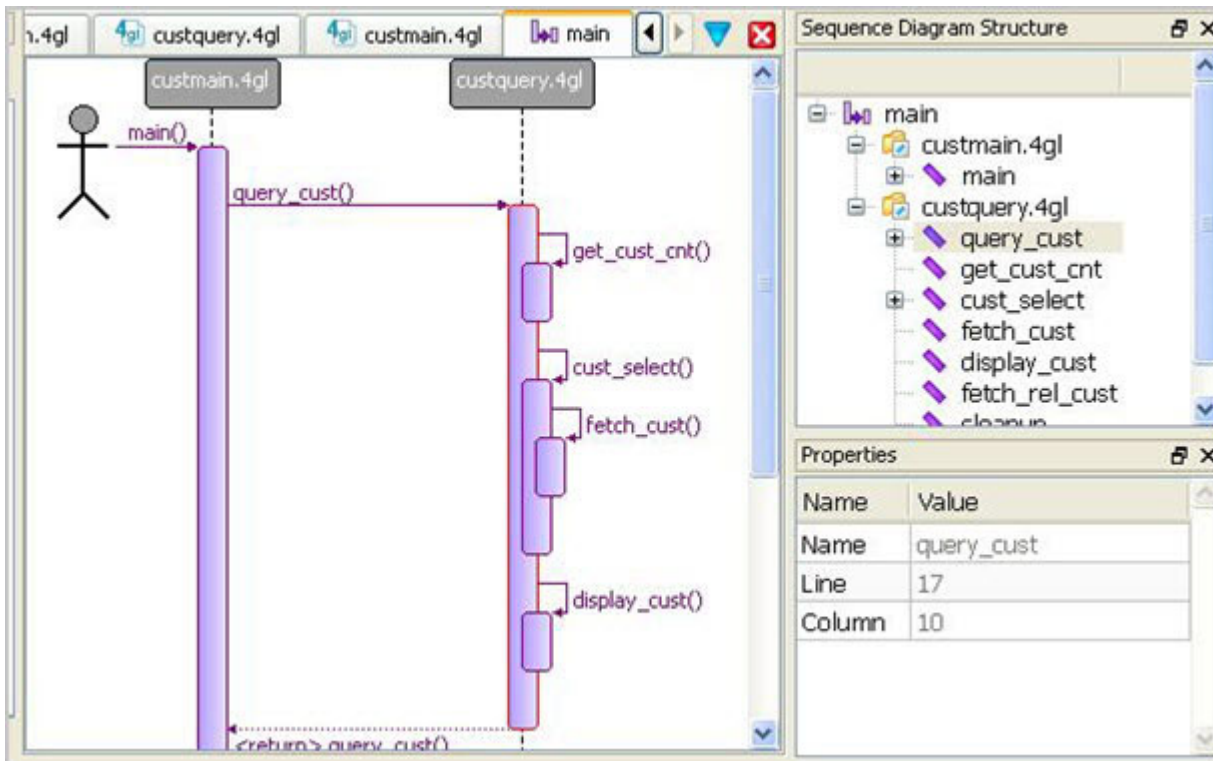


Figure 10-1. Sequence Diagram

- The starting point of the application is indicated by the stick-figure *actor*, the user who interacts with the application.
- Processes that exist simultaneously are represented by *parallel lines* - for a BDL program these lines represent the **source code modules** in your application.
- *Boxes* on the lines represent the **functions** in each module. The sequence is indicated by the order in which the boxes are listed; functions which are called by other functions have their boxes stacked on top. Plus/minus signs on each box allow you to display or hide *sub calls*.
- *Horizontal arrows* display the interaction between functions, the messages (**calls**) that are exchanged between them, and the order in which the calls occur.

In the example, the user interacts with the **MAIN** program block (function), which calls the **query\_cust** function in the **custquery.4gl** source code module. The **query\_cust** functions calls other functions in that module in the order indicated. For example, it calls the function **cust\_select**, which calls **fetch\_cust**. The **query\_cust** function returns to the MAIN.

You can use **Ctrl-mouse wheel** to zoom in/zoom out on the diagram.

### Structure View

The structure of the program modules is shown in a tree in this view. Use the plus/minus signs to display/hide the functions in a module. Select a function in the Structure view to display its properties in the Properties view.

### Sub calls menu

Right-click on the sequence diagram to display the menu options that will display or hide all the sub calls in a program.

## Creating the Diagrams

To create and display the diagram for a function from an **open .4gl source code file** - right-click the function name to display the context menu option **Open sequence diagram**. To diagram the entire application, right-click the word MAIN in the MAIN program block.

In an open Sequence Diagram, the right-click option **View source** opens the source code module (.4gl file) for the function, allowing you to switch back and forth between the source and the diagram.

In the Function calls view of an open Dependency Diagram, the right-click menus for Called Function and Caller Function have an option **Open sequence diagram**.

---

## Dependency Diagrams

- Displaying the Dependency Diagram
- Structure View
- Filter View

### Displaying the Dependency Diagram

To display the diagram, right-click the application node or the project node, and select **Open dependency diagram** from the menu of options.

The diagram displays a graphical view of the complex relationships between the various pieces of a project - the components that depend on other components, and/or have components that depend on them:

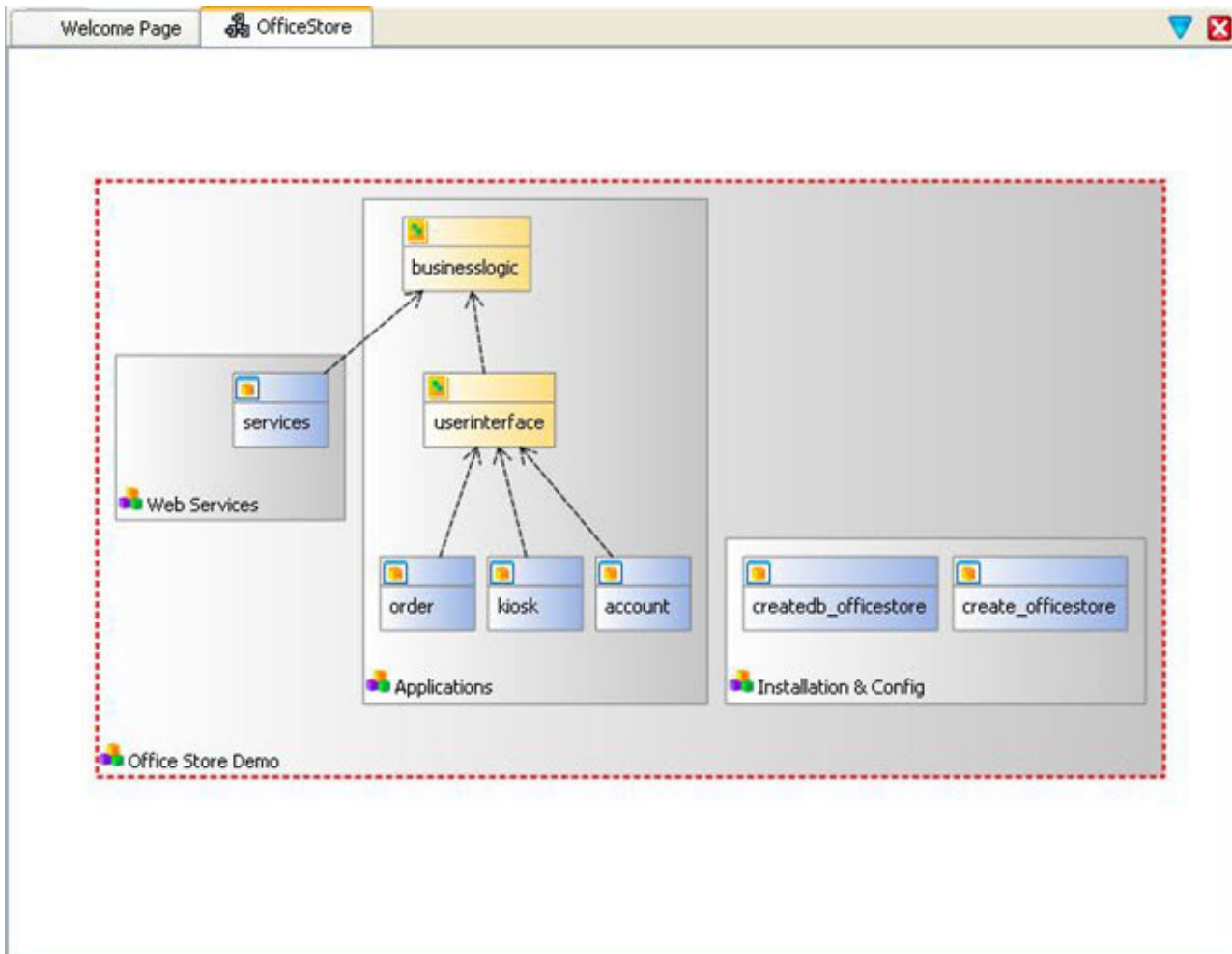


Figure 10-2. Dependency Diagram

Right-click a component in the diagram to display menu options:

- **Expand/Collapse** - to display or hide subcomponents
- **Locate in Project** - highlight the component in the project
- **Open <Project/Application/Library Node Name> in new tab**
- **Open in new Tab** - open a new tab containing the diagram
- **Show filter view** - display the Filter View
- **Hide selected item/Show All Items** - selectively display or hide items

You can use **Ctrl+mouse wheel** to zoom in and out on the diagram.

## The Structure Panel

The structure of the project is displayed in a tree in the **Structure view**.

Select a component in the diagram to display its properties in the **Properties view**.

Select the link in the diagram between components to display the associated function calls in the **Function calls view**.

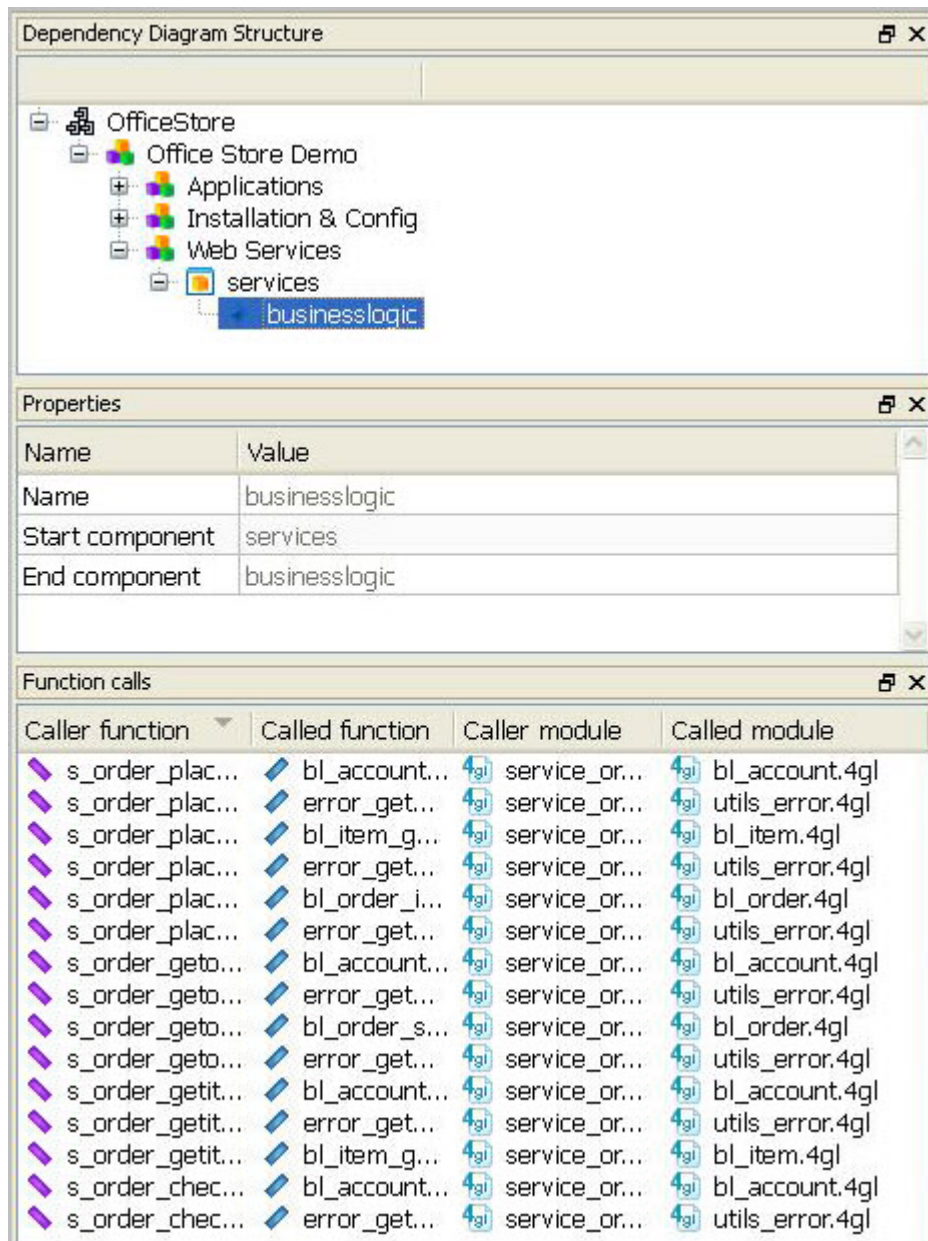


Figure 10-3. Dependency Diagram Structure view

Right-click menu options in Function calls view:

- Caller function
  - Symbol definition
  - Open call location
  - Open sequence diagram
- Called function
  - Symbol definition
  - Open sequence diagram
- Caller/Called Module
  - View source

## The Filter View

Right-click the diagram or a component and select the **Show Filter view** option of the context menu. Checkboxes on the Filter view allow you to specify which components are displayed in the diagram:

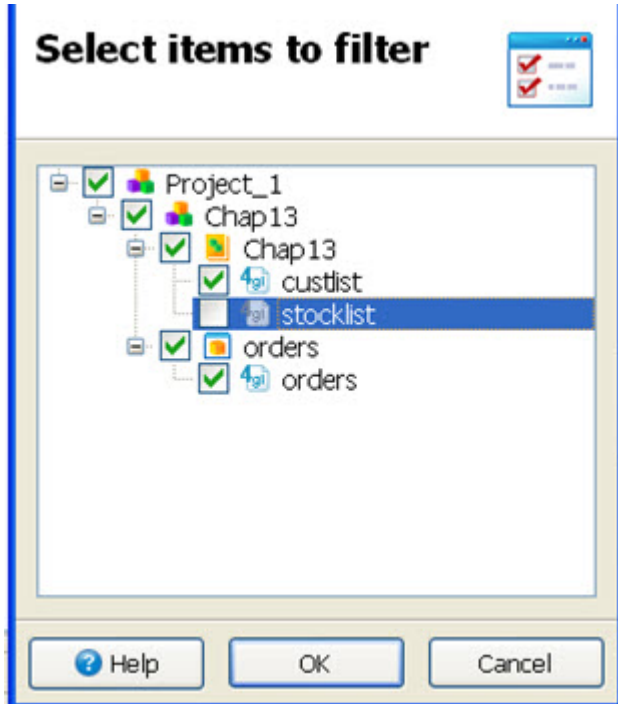


Figure 10-4. Filter View dialog



---

## Chapter 11. Customization

### Topics:

- “Customizing Genero Studio: Preferences”
- “Genero Studio Preferences” on page 11-3
- “User Interface Preferences” on page 11-7
- “Compiler and Runtime Preferences” on page 11-9
- “Form Designer Preferences” on page 11-10
- “Code Editor Preferences” on page 11-12
- “Additional Preferences” on page 11-17
- “Application Generator Preferences” on page 11-18
- “Graphical Differential Configuration” on page 11-19
- “Meta-schema Manager Preferences” on page 11-20
- “Welcome Page Preferences” on page 11-20
- “Report Writer Preferences” on page 11-21

---

## Customizing Genero Studio: Preferences

### Topics:

- “Studio Preferences Menu”
- “Saving changes to preferences” on page 11-2
- “Import Preferences” on page 11-2

## Studio Preferences Menu

You can customize the way the Genero Studio works, to meet your needs. Use the **Tools>>Preferences** menu item, or the toolbar icon, to modify the behavior of the following:

- General
- User interface
- Compiler and runtime
- Source Code Management
  - Subversion
  - Colors
- Meta Schema Manager
- File Browser
- Form Designer
- Code Editor
- Graphical Differential
- Application Generator
- Sequence Diagram
- Report Writer
  - Paper Settings
- Welcome Page

## Saving changes to preferences

The following buttons allow you to save, test, or cancel changes you have made:

- **Save to default** - save your updates as the default configuration
- **Load from default** - Reloads the initial default configuration
- **OK** - Save and apply all modifications, then exit preferences window
- **Cancel** - Undo all modifications and exit the Preferences window. The last saved values are restored.
- **Apply** - Confirms your updates, allowing you to test the new configuration. (No save is performed.) If you want this configuration to become permanent, you must save it by clicking OK.

## Import Preferences

When you install a new version of Genero Studio you can choose to import previous Preference settings from older installations of Genero Studio. The following dialog is displayed when Genero Studio is launched for the first time and previous installations are found:

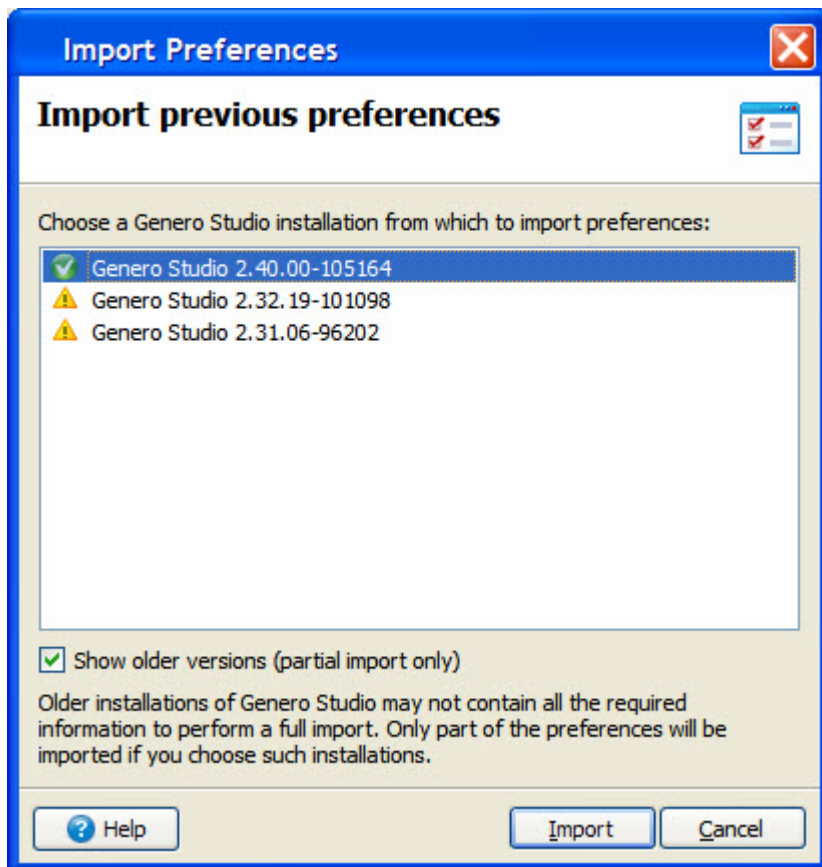


Figure 11-1. Import Preferences

Select an installation to import its Preferences settings and update the new installation of Genero Studio.

## Show Older Versions checkbox

If the version of Genero Studio is too old, it won't be listed in the dialog unless the "Show old versions (partial import only)" checkbox is checked. If an old version is chosen, only part of the Preferences will be imported (for example, Preferences such as menus or toolbars won't be imported).

---

## Genero Studio Preferences

### Topics:

- General preferences
- Workspace configuration
- User action configuration
- File associations configuration
- Messages configuration
- History configuration

## General preferences

Table 11-1. General preferences

| Preference         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Text file encoding | <p>To support <i>internationalization</i> ( <b>I18N</b> ), characters typed at the keyboard are intercepted and changed automatically, based on the Text encoding method selected. The default charset is the charset of your local system. To change the text encoding to one other than the default, make a selection from the combobox.</p> <ul style="list-style-type: none"><li>• If you are using Studio in a remote configuration, your encoding method must match the charset (locale) on the Studio server.</li><li>• If you are using CVS or other version control systems, the encoding method must be the same for all text stored in the system.</li></ul> <p>If you set Text Encoding to change the default charset, you must set the corresponding LANG environment variable in your current environment set for Genero Studio Configurations. Invalid LANG settings may cause compilation errors.</p> <p>See Studio Text Encoding Settings for additional information.</p> |
| Proxy Setting      | <p>If you need to access a web service using a proxy, enter:</p> <ul style="list-style-type: none"><li>• Host - the hostname or host IP address</li><li>• Port - The port number on which the service is listening</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

Table 11-1. General preferences (continued)

| Preference                 | Description                                                                                                                                                                                                                                                                                                                                |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Browser Setting</b>     | You can specify the browser to be used with the Genero Web Client: <ul style="list-style-type: none"><li>• Use default web browser</li><li>• Use specific web browser - enter the filename of the executable, including path, of the browser that you wish to use, or select the file in your file system using the Browse icon.</li></ul> |
| <b>Users and Passwords</b> | Clear all stored users and passwords: check box to clear.                                                                                                                                                                                                                                                                                  |

## Workspace configuration

This tab contains options related to your Studio workspace:

- **Open last session documents on startup**

If checked, the last documents that you had open will be restored when you start the next session.

- Workspaces

You can duplicate a workspace layout in order to define your own layout. Select one of the default workspaces, and use the Duplicate workspace icon to duplicate it in the User section. You can then use the Edit icon to change your new User layout.

### Default workspaces

- **Normal** - displays the usual views - Project, Output, Structure, Properties.
- **Debug** - removes some of the normal views to focus on the Debugger. Studio automatically switches the workspace when starting and stopping a debug session. It keeps track of the current workspace and the one used while debugging, and switches between them when the user starts or stops debugging an application.
- **Document** - displays the Central Workspace containing the document and the Project Manager, to focus on document editing.

### User workspaces

Contains the workspaces that you've duplicated and edited. Only User workspaces may be renamed or deleted.

## User action configuration

User actions allow you to define commands that can be invoked when you select a file within a Genero Studio module.

- **Search** - to find an action

Existing User Actions, if any, are displayed in a list:

- **Name** - names of existing user actions
- **Command Line** - command line for the corresponding action

Use the icons to **Add** a new User Action, or to **Edit** or **Delete** an existing User Action.

The **Add** and **Edit** icons call the User Action Dialog:

### User Action dialog

- **Name** - Assign a name to your User Action
- **Description** - Optional description
- **Icon** - Optional icon
- **Command line** - Enter the command to be executed. Arguments can be included on the command line, as needed. Use the **Insert Variable** button to select from a list of predefined variables that can be used in the command.

The following example command line will execute the program **stest** using the selected file.

```
C:\myprograms\stest ${FileName}
```

Once you have created a new User Action, you can associate it with files of a specific mime type using **Preferences, File Associations**.

## File Associations configuration

You can associate file types handled by Genero Studio modules with Studio predefined actions, or with User Actions that you have defined.

**Search** - search for the file type you wish to view or edit.

### File Types

Toolbar icons let you add, edit, and delete file types.

- **Mime** - predefined file type/identifier, i.e. application/myapp
- **Description** - description (optional)
- **Editor Content type** - type of editor content
- **Extensions** - the extensions matching the file type
- **Filenames** - the filenames matching the file type (optional)

### Associated Actions

Toolbar icons let you add and delete actions, set an action as the default, or set the file open action.

- **Action** - predefined action names
- **Default** - indicates the action is the default; use the icon to change the default
- **Open** - indicates the action is used to open the file; use the icon to change this
- **Command** - predefined command to be executed; may be <internal>

When you select **Add** from the icons at the top of the Actions list, the available actions are displayed in a list for your selection. User Actions must be defined using **Preferences, User Actions** before they will appear in the list.

**Description:** description of the selected action

## Appearance

- **Icon** - search for the icon on your file system. The path is specified for icons that are not in the default Studio icons directory
- **Color**- select a color
- **Preview** - a sample is displayed

## Messages configuration

### Message dialogs

A list of messages is displayed as a tree, grouped by Studio module. Expand a module listing to display its messages.

You can prevent a message from displaying while you are using Studio by unchecking the **Show** checkbox for the message.

Use the **Search** box to locate a specific message.

### Document Error Filters

This list specifies the message id of information or warning messages that you wish to be hidden in the Document Errors tab of the Output View. Use the icons to add and remove message ids from the list.

## History configuration

In the work area of many of the Genero Studio modules, the names of previously used items are stored and displayed as history. Studio allows you to manage these History lists.

### History list

The name of each list that you can manage is displayed. These are the only lists that you can alter.

### History Details

- **History Size** - to change the maximum number of items displayed in a list.
- **Allow Duplicates** - if this box is checked, duplicate names can be added to the list of items. If the list contains duplicate items, unchecking the box will cause a dialog box to be displayed, asking you to confirm the removal of the duplicate items from the list.

### History items

Icons allow you to:

- **Add** - add a new item
- **Delete** - remove the selected item
- **Edit** - change the name of the selected item
- **Up/Down arrow** - Rearranges the items in the list by moving a selected item up and down.

- **Default** - specifies the default item for the selected list.

---

## User Interface Preferences

### Topics:

- Toolbars configuration
- Menubars configuration
- Accelerators configuration

## Toolbars configuration

Expand a toolbar listed in the tree to display its actions.

Select a toolbar or action, and use the icons or right-click to display the menu options:

- Add a new toolbar
- Add an action to a toolbar
- Remove or rename a toolbar
- Remove or rename an action
- Add or remove a separator

Item order within a toolbar may be changed using "drag and drop".

## Menubars configuration

Expand a menu listed in the tree to display its actions.

Select a menu or action, and use the icons or right-click to display the menu options to:

- Add a new menu at the same level, or a submenu
- Add an action to a menu
- Remove or rename a menu
- Remove or rename an action
- Add or remove a separator

Item order may be changed using "drag and drop". This applies to menu order within the tool bar tree, and to icon order within one menu.

## Accelerators configuration

Accelerators and their associated Genero Studio actions are defined in a default set called a profile.

### Profile

Profiles correspond to a set of accelerators for each action. They are saved in Accelerator Profile files having an extension of **.apr**. A **default** profile file is preinstalled in the Genero Studio installation directory. Initially, this is the only profile available, and it cannot be modified. You can create your own profile by duplicating the **default** profile and modifying the accelerators associated with an action. All the accelerators (except menu accelerators) are disabled during editing, and are re-enabled when editing is completed or you leave the Accelerators configuration window.

- **Current** - This combobox displays the currently active profile, with the names of other available profiles displayed in a dropdown list. The **default** profile, which cannot be modified, has a lock icon. To change profiles, select a profile from the dropdown list, and click **OK**.
- **Duplicate** - Creates a new profile. The **Duplicate Profile dialog** allows you to enter the name of a new profile, which will be a copy of the currently active profile. The file is saved in the following user directory:  
**Documents and Settings\<username>\Application Data\<companyname>\Genero Studio<version>**.  
Modifications in the accelerators can be made in your new user profile. All the user profiles that you create will have a user icon associated with their name in the list in the Current combobox.
- **Remove** - Removes the currently active profile. This removes the associated **.apr** file. The **default** profile provided by Studio cannot be removed.

Accelerator profiles can be shared with other users:

- **Import** - Imports an **.apr** file. An Open dialog allows you to browse for the profile file. When you click **OK**, the selected file is copied into the user directory and appears in the current combobox.
- **Export** - Exports an **.apr** file. A Save As dialog allows you to save your profile file.

## Actions

The actions and the associated accelerators for a particular profile are displayed in a table, grouped by application by default. Expand the application group to display all its actions. The **Search** box allows you to locate a specific action in the tree.

You can sort the table to make the resolution of conflicts easier. Use the icons to switch between viewing the tree grouped by application, or in alphabetical order by action. To sort the tree by the accelerator column, click the column title.

## Accelerators

The accelerators that are defined for an action display in a list. Use the icons or right-click for a menu of options to:

- **Add** a new, or additional, accelerator to an action using the Add New Accelerator dialog
- **Edit** the accelerator for an action using the Edit Accelerator dialog
- **Remove** the selected accelerator from an action
- **Set the selected accelerator as the default** for an action

## Add New Accelerator/Edit Accelerator dialog

Enter your own key combination for the accelerator by hitting the keys in order; use the **Tab** button to enter a Tab key.

Use the **Backspace** key to erase the latest keypress, or the **Clear** button to erase the entire key combination.



---

## Compiler and Runtime Preferences

### Topics:

- Preprocessor configuration
- Compilation configuration
- Run/debug configuration
- Build Rules

### Preprocessor configuration

The FGL preprocessor is used to transform your sources before compilation. It allows you to include other files and to define macros that will be expanded when used in the source.

- **Use fglpreprocessor** - when checked, use the Genero preprocessor for compilation of files.
- **Include path** - defines a path used to search files included by the `&include` directives.

### Compilation configuration

#### General

- **Maximum number of parallel compilation tasks**

Specify the maximum number of parallel compilations that you want to allow. although only one link at a time can be done, multiple file nodes can be compiled simultaneously. This speeds up the build process if the machine has multiple processors.

If you specify 0 maximum, Project Manager uses the number of processors available on the local machine (i.e. 2 for a dual core machine).

- **Maximum number of erroneous files** - specify the maximum number of files containing errors to be permitted before compilation is automatically stopped.
- **Compute dependencies between files (recommended)** - This enables the computation of dependencies. They will be computed when the project is loaded, and is enabled by default. It can be useful to disable it for very large projects or projects on network drives with performance problems. However, it should ordinarily be enabled, in order for the build system to take the `#include` and globals into account.
- **Compute additional information about functions** (recommended)

**Output** - specify whether to clear output after each compilation.

### Run/debug configuration

**Console** - specify whether to clear the output after each launch.

### Build Rule configuration

Build rules define how source files are to be compiled. When compiling a file, Project Manager will try to find the build rule corresponding to this file's MIME-type. If there is one, the commands of the rule are executed to build the source.

Build rules are grouped into the following categories:

- *Default* Build rules, provided by Genero Studio, which apply to all source files and cannot be modified.

- *Global* Build rules that you can create, which will apply to all source files in all projects.
- *Project* Build rules that you can create in Project Manager. These rules apply only to the source files within a specific project, and **must be added and viewed from Project Manager** (use the right-click menu option **Edit Build Rules** when a project is open.)

If there are Build rules for the same file type in different categories, only one Build rule is active. Project Build rules take preference over Global and Default Build rules; Global Build rules take preference over Default Build rules.

### Build Rules list

- **Name** - label assigned to the rule
- **File Type** - The accepted file type for input files (mime type or file extensions). There can be only one Build rule for a given file type within each category.

### Build Rules fields

- **Command line** - command to execute. Variables from **Project Manager** may be used in the commands. For example, to replace the current filename without extension: \$(InputBaseName).42m.
- **Output files** - files resulting from the build. The same variables may be used here.
- **Intermediate files** - indicates which files are generated during compilation, as intermediate files, before generating the output file. This allows you to view an intermediate .4gl file, for example, and examine or modify the generated code.
- **Additional dependencies** - additional files required (other than the source file). Each Genero application or library can depend on other libraries, which will be linked with the application.

See Add/Edit Build Rules for additional information about modifying the Build rules.

---

## Form Designer Preferences

### Topics:

- Form settings
- Form Elements settings
  - Table,Tree Settings
  - WebComponent Settings
- Database Form settings

### Form settings

- **Default width**, in number of characters
- **Default height**, in number of lines

**Open form after import** - if checked, form will be opened in Form Designer after being imported.

## Form Elements settings

### Table and Tree Settings

- **Default rows** - number of rows
- **Default columns** - number of columns
- **Default column width** - size for a new column, in characters

### WebComponent settings

Specifies the directory in which the WebComponent XML files and optional icon files are stored. Each of the XML files describes a single WebComponent object. The extension of the XML description file is **.wcsettings**.

Once you have set this directory, you may add a WebComponent widget to your form design document. The components described in the .wcsettings files will be available in the combobox list of the `componentType` property in the Properties View, allowing you to specify the particular WebComponent you wish to add to the form.

See Using Web Components for additional information.

## Database Form settings

### Container

Choose the container for the form.

### Table and Tree

- **Row count** - number of rows
- **Maximum column width** - in characters

### Grid and ScrollGrid

Label and Field Alignment: Icons allow you to specify the following:

-  Label left and field left
-  Label right and field right
-  Label left and field right
-  Label right and field left

Property label and Field:

- **Maximum width** - in characters
- **Number of fields** - number of fields per line
- **Top border** -; in lines
- **Bottom border** - in lines
- **Left margin** - in characters
- **Right margin**  
- in characters
- **Field gutter** - spaces between label and data for a field

- **Field gap** - gap between two fields on the same line

TextEdit:

- **Maximum widget height** - enter value

---

## Code Editor Preferences

Topics:

- General Preferences
- Behavior and Display Preferences
  - Behavior Preferences
  - Display preferences
  - Language-specific preferences
- Color and Font preferences
- Template preferences
- XML Schema/DTD preferences

### General Preferences

The **vi Mode Settings** checkbox allows you to specify that the vi Editor will be used to open files.

### Behavior and Display Preferences

#### Language

Select the programming language to which the preferences will apply; the default is **All Languages**. If you select a specific language, the changes will apply only to that language.

**Use custom settings** - this checkbox appears only if Language is not set to **All Languages**. This box must be checked in order for you to make any changes in the settings for the selected language.

In order to enable the Language-Specific tab, Language must be set to a specific language.

#### Behavior Preferences

Tabs:

- **Tab size** - default size is four characters.
- **Insert spaces for tab**- if checked, automatically replace consecutive whitespace with a tab character when the number of generated by a tab indent is equal to the Tab size.

Indentation:

- **Smart tab** - if checked, when the Tab key is pressed the cursor is moved to align with the first character following a whitespace on the previous non-blank line.
- **Smart backspace** - if checked, backspace intelligently through whitespace.
- **Smart indent** - if checked, indent code based on the indentation of the previous line. New lines are indented to the first non-blank character of the line above.
- **Strip trailing white spaces** - remove trailing whitespace from the file

Automatically close: If checked, the closing symbol will be added to each of the following:

- **String**
- **Single quotes**
- **Parenthesis**
- **Curly braces**
- **Square brackets**
- **Angle brackets**

Smart key options:

- **Smart home** - if checked, the Home key moves the cursor to the left of the first non-blank character on the line.
- **Smart end** - if checked, the End key moves the cursor to the right of the last non-blank character on the line.

Wrapping:

- **Line wrapping** - enabled if checked.

## Display Preferences

Editor:

- **Show line numbers** - Enables or disables line number display
- **Show Right Margin** - Displays a thin line on the right side to indicate the right margin
- **Right margin** - Row number where the right margin line is located. Sets the right margin of the editor. The default is 80 characters
- **Show indentation guide** - vertical lines that indicate relative indentation of text

Highlighting:

- **Current line** - highlight line at caret position
- **Brace Match** - enable brace match highlighting (highlights both braces in a set when the cursor is immediately to the right of one of the braces). The user can change the color and style of highlighting in the Color page of the dialog box.

## Language Specific Preferences

When Language is not set to All languages, the **Language Specific** tab is enabled. Click the tab to display any preferences related to that language.

## Color and Font Preferences

You can specify how specific styles of your Editor code appear.

### Global Style - applies to all languages

- **Theme** - quickly configure the display using predefined color combinations
- **Font family** - specify the font family; when this is changed it applies to all languages, except those for which custom settings have been created.
- **Font size** - specify font size, when this is changed it applies to all languages, except those for which custom settings have been created.

## Style Specific Settings (custom settings) for all Languages

**Language** - the default selection **All Languages** in the combobox displays the default styles and properties applicable to all languages. As a default, specific language pages inherit the settings defined for **All languages**.

**Style** - list of styles. Select a **style** to display the following properties:

- **Color** - specifies the color for the **Foreground** and **Background** of the style, when this is changed it applies to all languages.
- **Font** - specifies the font selected from the available screen fonts installed on your system. **Use custom font for the selected style** must be checked in order to make changes in the following:
  - **Family** - font family associated with your selected font; changes will override the global setting.
  - **Size** - font sizes associated with your selected font. changes will override the global setting.
- **Bold** - specifies whether the selected font is Bold. Inherits the global family and size.
- **Italic** - specifies whether the selected font is Italic. Inherits the global family and size.

## Style Specific Settings (custom settings) for a specific Language

**Language** - list of languages. Select the **language** from the dropdown list, to display the language-specific styles and properties.

**Use custom settings** - this checkbox displays when you select a specific language in the list. This box must be checked in order for you to make any changes in the settings for the selected language. When this box is checked, the settings for the selected language become specific, and it does not inherit the **All languages** settings any longer.

- To change the **Color (Background, Foreground)** or the font properties **Bold** or **Italic** for a specific style, you must first select the **Style** from the Style list. **Use custom settings** must be checked.
- To change the **Font Family** and **Size** properties for a specific style, you must check **Use custom font for the selected style**. This overrides the Global Font family and Font size settings.

## Template Preferences

**Language** - select the desired language. The default is Genero BDL.

### Templates

Displays a list containing the **Name** and **Description** of existing templates. When an existing template is selected, the corresponding template code is displayed in **Preview**.

Icons allow you to:

- **Add** - Displays the **Add Code Template** dialog box, allowing you to enter the new template name and description

- **Delete** - Deletes the selected template
- **Edit** - Displays the **Edit Code Template** dialog box, allowing you to edit the name and description of the selected template.

Add/Edit a Template:

- For new templates, enter the template name and description.
- Type directly in the pane to add the code for a new template or to make changes in existing code.
- Use the pipe character '|', to denote the cursor position in the expanded template. For example, when you insert the following template into your code, the cursor appears inside the parenthesis after the CASE keyword.

```
CASE (|)
 WHEN ()
 WHEN ()
END CASE
```

**Important:** When you modify or create a template, you must type it exactly. For example, you must use indents.

**Preview** - Displays the code for the selected template.

### Import/Export buttons

- **Import** - import 4GL template definitions from an XML file
- **Export** - export 4GL template definitions to an XML file

## XML Schema/DTD Preferences

Many XML documents contain external links to stylesheets, schemas, DTDs etc., which may be stored at remote locations. Studio XML catalog files may be used by Studio to provide an alternative path when the external entities aren't accessible.

Use **Tools>>Preferences** and select **XML Schema/DTD** from the Code Editor page in the Pages tree. A list of the XML catalog files provided with Studio is displayed.

### XML Schema/DTD Configuration

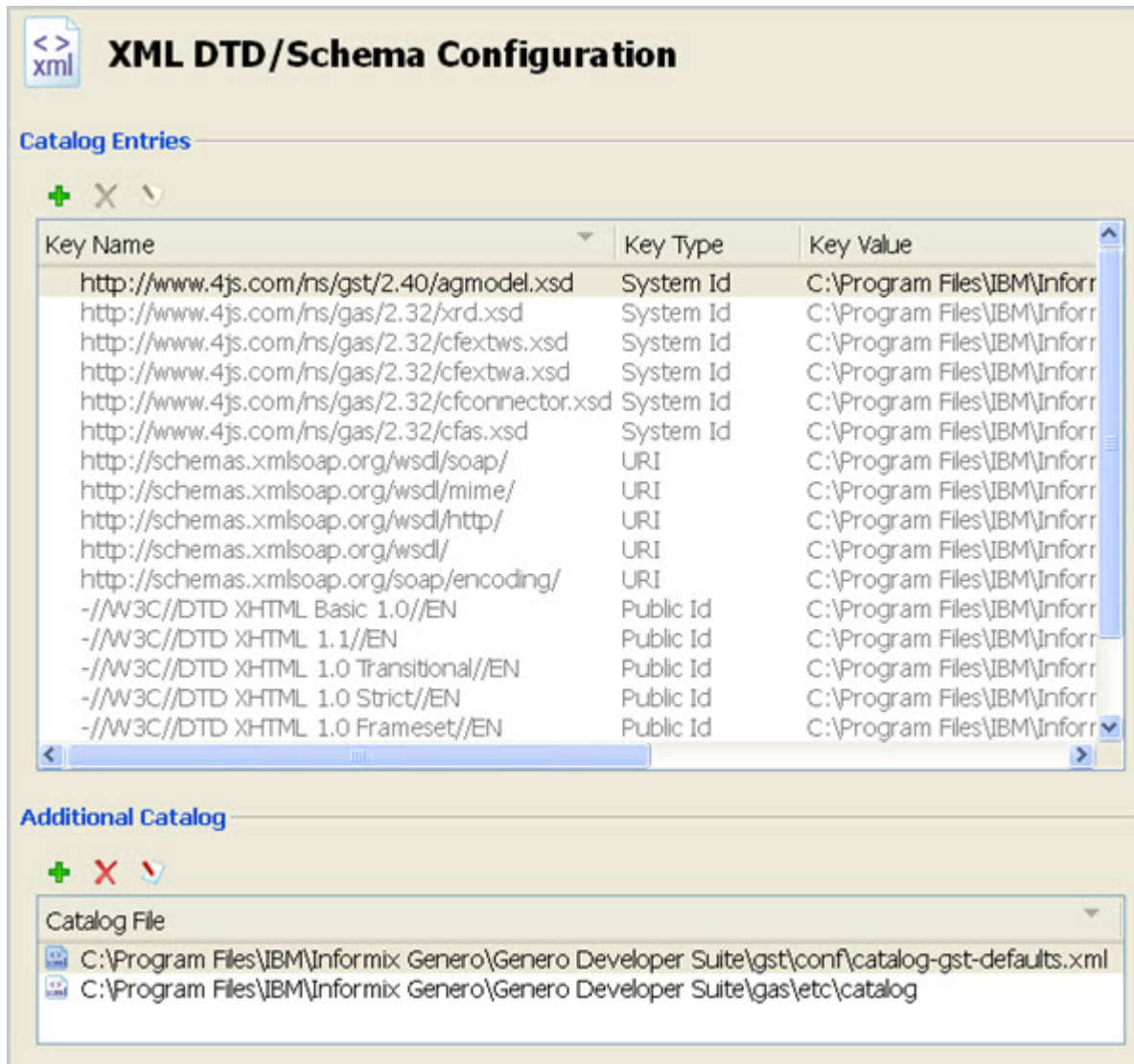


Figure 11-2. XML Schema / DTD Configuration

## Catalog Entries

This section displays the entries for the XML Catalog files that are provided by Studio.

Use the icons at the top of the Catalog Entries section to add/delete or edit catalog entries. An edit icon appears next to catalog entries that may be edited.

The following values are displayed for each Catalog Entry:

- **Key name** - can be empty; already existing names can not be used.
- **Key type** - valid values are:
  - **Public ID**, for the DTD
  - **System ID**, for the DTD
  - **Schema URI**, for the Schema location
- **Key value**
  - For **Public/System IDs** - the absolute file location for the DTD



- For **Schema URI** - the XSD file location; a Browse button is provided to locate the grammar file.

## Additional Catalog

Entries in this section specify an XML catalog file to be added to the end of the current catalog. This allows one catalog to refer to another. If a reference cannot be resolved in the current catalog entry file, then Studio moves to the next catalog specified in the Additional Catalogs section.

Entries defined in a Studio XML Catalog file are given preference over entries that come from a Next Catalog file. This allows you to override the entries in the Next Catalog file to resolve external grammar files.

Use the icons at the top of this section to add/edit/delete Additional Catalog entries.

- **Next Catalog File** - specifies the name of the additional XML catalog file.

See “Studio XML Catalog Files” on page 5-19 for additional information about Catalog files.

---

## Additional Preferences

### Topics:

- Source Code Management Configuration
  - Subversion
  - Colors
- File Browser
- Sequence Diagram

## Source Code Management Configuration

### Subversion

If you are using version control, this page allows you to specify the location of your SVN client.

- **SVN client path** - enter the path to your SVN client; include the bin directory in this path.

Example:

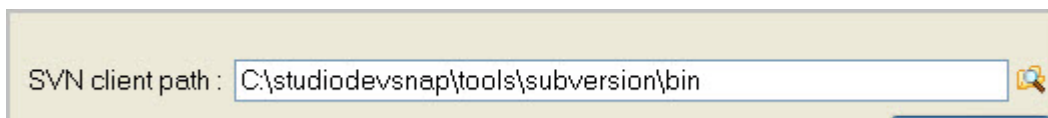


Figure 11-3. SVN client path field

- **SVN version** - press the **Check Version** button and the version information displays in this window.
- **Checkout options** - specify the list of file extensions for the types of files that you wish to be shown in the open file/folder page after completion of checkout.

## Colors

You can choose the color that indicates the status of a file/project under SVN control. Click the corresponding color square to open a color palette.

## File Browser

**Show Hidden Files** - check to display hidden files in the File Structure

## Sequence Diagram

Default maximum dependencies - select maximum number of dependences; default is 1

---

## Application Generator Preferences

You can set preferences for the template sets that are used by Application Generator when you create a program.

### Templates list

- **Standard** - This section displays a list of the standard template sets that are provided by Studio. The default template set is checked. Select the name of a template set to display its details or use the toolbar icon to **set it as the default**.
  - *Multiple Dialog*, the default set of templates, enhances the behavior of applications that interact with a Genero form, and implements the Genero 2.x Multiple Dialogs feature. This default template is recommended unless you are using a version of Genero BDL that does not support Multiple Dialogs.
  - *Single Dialog* is another standard template set that provides the legacy behavior of earlier versions of Application Generator. This template should only be used if you are using an earlier version of Genero BDL that does not support Multiple Dialogs.
- **User** - This section displays a list of the template sets that you have defined. Select **User** and use the toolbar icon to **Add** the details of the template set. Or, use the toolbar icons to **Delete** or **Edit** the details of the selected template set, or to **set it as the default**.

### Template details

- **Location** - directory where the files of the template set are located
- **Description** - short description of the template set

### Options

- **Store generation document** (debug). The file **gen.xml** will be stored in the same directory as the corresponding Application Generator program definition Files.

### Add New Template location dialog

When you click the **Add** icon, this dialog allows you to add a location for a User template set.

You can:

- provide a **name** for the template set
- add the **location** of the template set (directory where the files are stored)
- provide a **description** of the template set

---

## Graphical Differential Configuration

This page allows you to specify an external tool to be used for comparisons, instead of using the integrated Studio Graphical Differential. You can also specify options for the internal and/or external tool.

If you select an external tool, it will be invoked when you compare two files using File Browser or Project Manager. If you use the Studio main menu option **Tools >> Graphical Differential**, the integrated Studio tool will still be invoked.

### Tool

Select the tool you wish to be used when you compare two files using File Browser or Project Manager.

- **Use internal graphical tool** - this is the default, the integrated Studio tool
- **Use external graphical differential tool** - select this radio button to specify a different tool

### Internal tool configuration

To enable the following options, select the checkbox associated with the option:

- Consider uppercase and lowercase to be the same
- Ignore all white space
- Ignore changes in amount of white space
- Ignore changes whose lines are all blank
- Expand tabs to spaces in output
- Try hard to find a smaller set of changes

### External tool configuration

If you have chosen to use an external tool, enter the configuration information:

- **External tool command line** - enter the path of the external tool
- **External tool arguments** - use the following variables to specify the arguments for the command line:

**`$(file1) $(file2)`**

The names of the files will replace the variables when the tool is invoked.

Example:

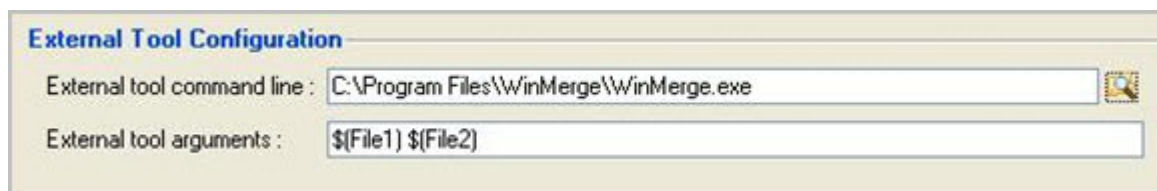


Figure 11-4. External Tool Configuration

---

## Meta-schema Manager Preferences

Database Schema files describe the structure of a database, listing the tables, columns, indexes, and other properties. Specify the schema files (.4db) that will be loaded by default into the Meta Schema Manager when Studio is launched. The schema will be displayed in the DB Schemas Tab of the Project Manager, and available to the other Studio components such as Form Designer.

Toolbar icons allow you to select a schema file to be added, and to delete or edit schemas previously added to the list:

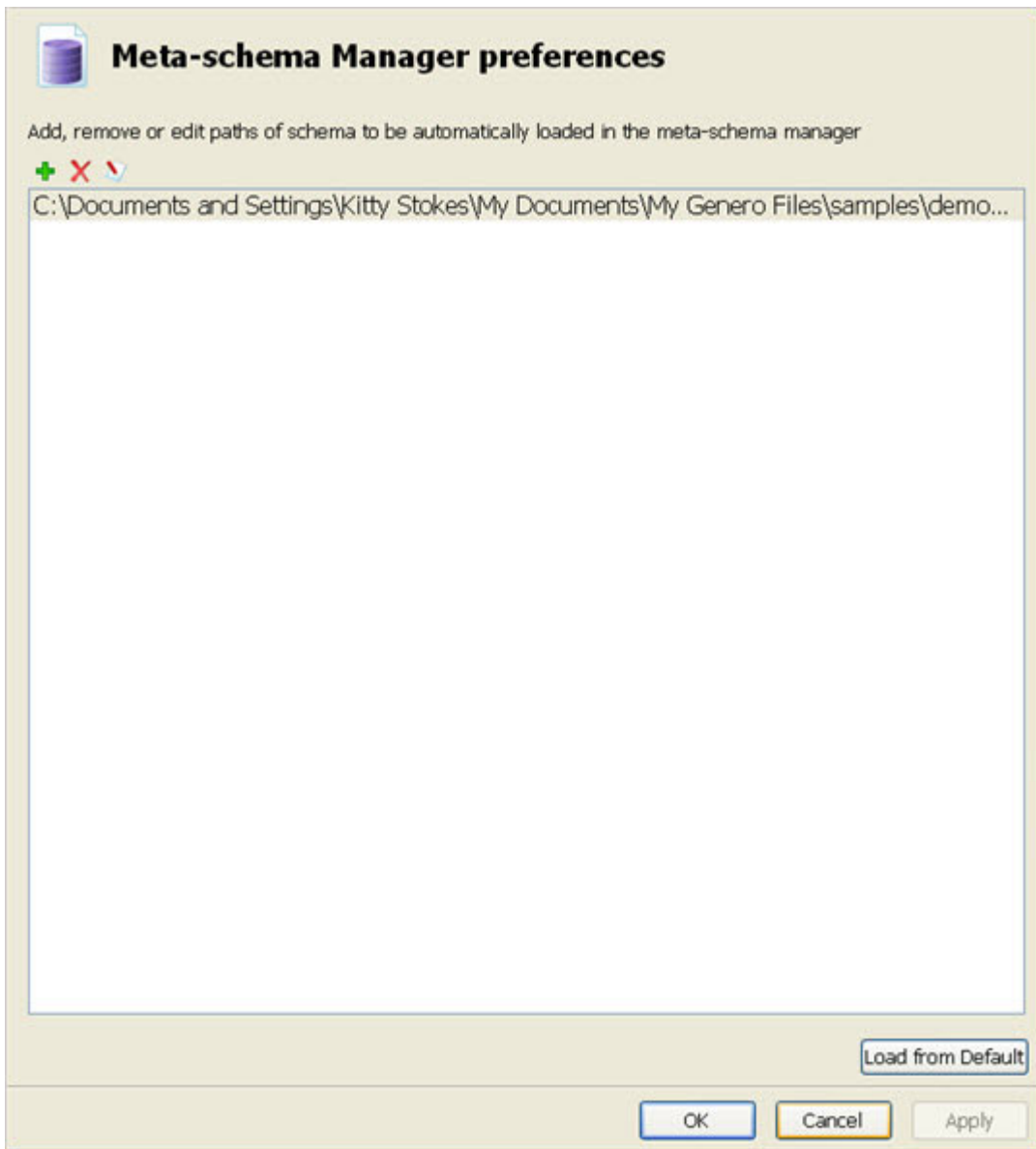


Figure 11-5. Meta-schema Manager preferences

---

## Welcome Page Preferences

This page allows you to change the defaults for the Welcome Page, which provides links to recent projects and files.

- **Show Welcome Page at Startup** - uncheck if you do not wish this page to display.
- **Template** - select the template for the Welcome Page.
- **Clear** - clear all stored cookies.
- **Visible Sections** - check the sections that you wish to be displayed.
- **Parameters** - add or edit parameters used on the page, or edit the RSS location.

---

## Report Writer Preferences

### Paper Settings Preferences

You can specify the default paper settings to be used for report design documents.

- **Orientation:** portrait, landscape
- **Units:** centimeter, inch
- **Page Size Format:** standard - choose a value from the combobox, custom
- **Margins** - left, right, top, bottom



---

## Chapter 12. Source Code Management - SVN

- What is Genero Source Code Management?
  - Specify a Subversion client
- Getting Started
- Checking out an existing project or project files
  - Using the SVN Checkout dialog
  - Process Log
- SVN Log Select revision dialog
- Accessing Source Code Management (SCM)
  - SVN views
    - SVN Log
      - Viewing the SVN Log
      - Specifying the revision range
    - SVN Repository Browser
    - SVN Status
  - SCM menus
- Working with SVN
  - Adding files
  - Delete Files
  - Updating files
  - Commit/Review Changes
  - Revert changes / Unadd files
  - Diff with current revision
  - Cleanup
  - Copy SVN files and directories
  - Merge / Revert revisions
  - Switch (Moving a working copy)
  - Create Patch
  - Apply Patch
  - Manage Properties

---

### What is Genero Source Code Management?

Genero Source Code Management enables collaborative sharing and maintaining of the files in your Genero projects. Current and historical versions of the files are stored in a Subversion repository, allowing you to recover older versions of your data, when necessary. Your files (called working copies) are stored in an ordinary directory on your local file system. When you make changes to these working copies, you copy the changed files to the SVN repository.

A SVN repository often holds the files (or source code) for several projects; usually, each project is a subdirectory in the repository's file system. A user's working copy will usually correspond to a particular subtree of the repository

If a Project is under Source Code Management, the icons for the project nodes will have a status icon superimposed when you open the Project.

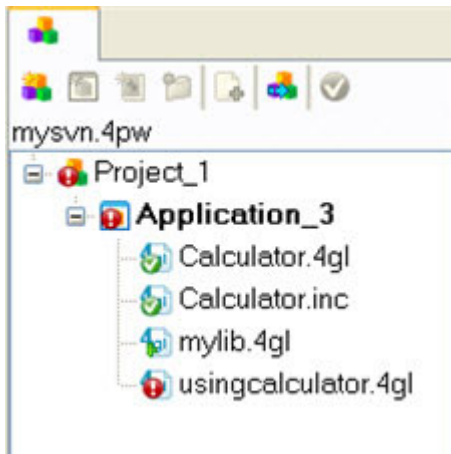


Figure 12-1. Project view with status icons

Table 12-1. Status icons and their meanings

| Icon                                                                                | Description                                                                                                             |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
|    | Item is unchanged since it was last committed to the repository.                                                        |
|    | Item has been changed and has not been committed yet. The application and the project will also have change indicators. |
|  | Item is scheduled for addition to the repository.                                                                       |
|  | Item is in conflict with updates received from the repository.                                                          |

## Specify a Subversion client

A Subversion client must be installed on your local machine.

Genero Studio for Windows can use **Collabnet's Subversion 1.6.2 or later**. For more information about Collabnet's Subversion, see <http://subversion.tigris.org/>.

Genero Studio for GNU/Linux relies on **Subversion 1.6.2 or later**, which must have been installed on the system.

Complete this procedure to specify a subversion client.

1. Select **Tools >> Preferences, Source Code Management, Subversion**
2. In the SVN client path, enter the name of your SVN client.
3. Click Check Version to validate the SVN client and to show SVN version details.



---

## Getting Started

1. Genero Studio allows you to use Subversion to manage the versioning of your files. You must have access to a SVN Repository that has been set up to store your versioned files. Contact your system administrator or Subversion vendor for information and documentation on the set up of Subversion and repositories.
2. Decide which directory / directories on your local system will contain the checkout files.
3. Create a new project in Projects view for the checkout files.
4. Right-click on the project in the Projects view, and choose the **SCM >> Checkout** menu option. An SVN Checkout dialog will guide you through the checkout steps. If the repository is empty (no one has added projects), no files will be added to the Project Structure, but the checkout system is now set up.
5. Create a new Project and save it in the checkout directory. The files to be versioned must be stored in the checkout directory.
6. Use **File >> New** to create files and store them in the checkout directory and the project structure. For existing files in the checkout directory, use the **Add Files** icon or context menu option to add them to the Project Structure.
7. Any files from the checkout directory that have been added to the Project Structure can be committed to the repository.

At this point, you can use the SCM menu options to handle versioning for your files in the checkout directory.

**Note:** although the .4pw Project file is used internally by Studio, we recommend that you include it in the checkout directory and commit it to the repository to share it with other developers. although the .4pw and .deps files are in the checkout directory, they do not have to be added to the Project Structure.

---

## Checking out an existing project or project files

If projects or project files have already been committed to a repository, you can load copies into a working directory on your local system.

### Checking out an existing project from the File Browser view

1. Open the File Browser view.
2. Navigate to the directory that will serve as your new working directory. If the directory does not exist, create it.
3. Right-click on the directory and select **SCM >> Checkout**. The SVN Checkout dialog displays with the Checkout directory path populated. See the SVN Checkout dialog topic for details on completing the dialog.

The files will be checked out and in the checkout directory, but they will not show in the Projects view structure until you open the .4pw file.

### Checking out an existing project from the SCM Menu

Select **SCM >> Checkout** The SVN Checkout dialog displays. See “Using the SVN Checkout dialog” on page 12-4 for details on completing the dialog.

The files will be checked out and in the checkout directory, but they will not show in the Projects view structure until you open the .4pw file.

## Using the SVN Checkout dialog

The SVN Checkout dialog guides you through the checkout process, allowing you to load copies of repository files into a checkout directory on your local system:

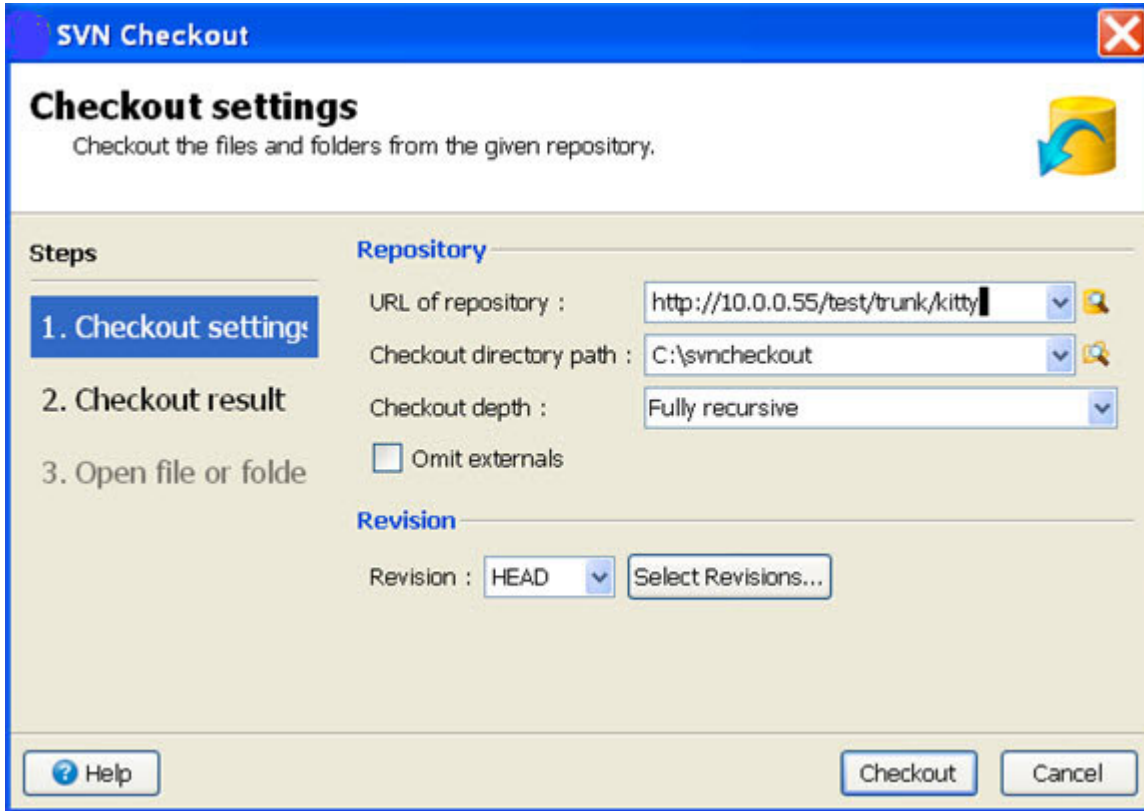


Figure 12-2. SVN Checkout dialog

The **Repository** section:

- **URL of repository** - enter the URL of the SVN repository that you wish to access.
- **Checkout directory path** - enter the path of the directory for the checked out files; the Browse button allows you to locate the path.
- **Checkout depth** - choose the level you desire; for example, "only file children" will checkout the files but not the directories.
- **Omit externals** - check to exclude "external items".

The **Revision** section:

Specify the particular version from which to checkout the files. The Select Revisions button opens the "SVN Log Select revision dialog" on page 12-5.

### After the Checkout button is clicked

Messages related to the checkout operation display in the **Output View**.

The results of the Checkout and other SCM commands are displayed in **Results** pages. A **Run in background** button, at the bottom of the SCM dialogs, allows you

to close the dialog but continue running the process in the background, as indicated in the Task Manager.

## The Process Log

When you complete an operation, a log of operations displays.

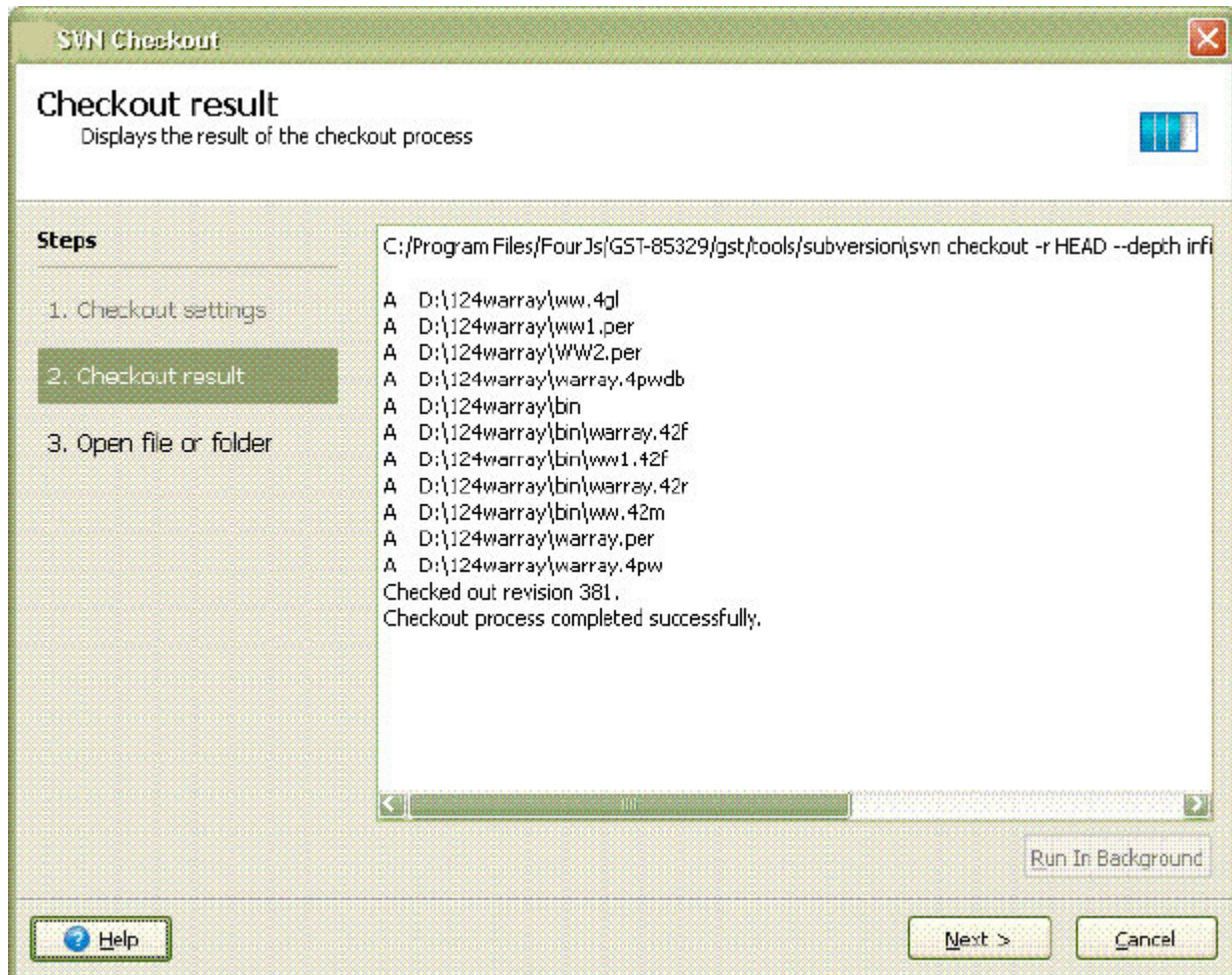


Figure 12-3. SVN Checkout Process Log

## SVN Log Select revision dialog

The SVN Log Select revision dialog allows you to select a specific revision within a version. To reach this dialog, the repository URL and version will already have been selected.

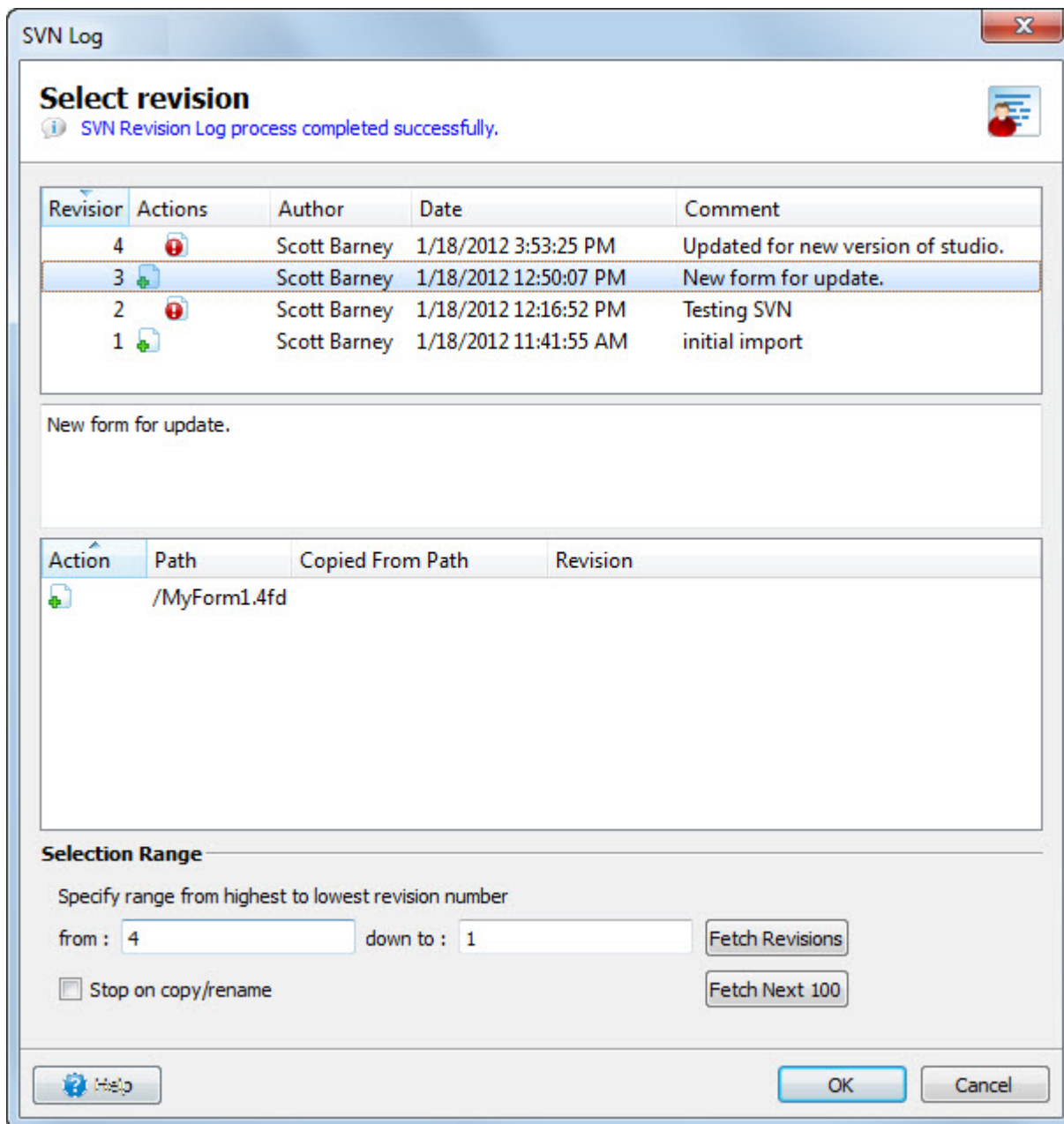


Figure 12-4. SVN Log Select revision dialog

To select a revision:

1. Select a revision from the revision list.

Once a revision is selected, the comments for that revision and the actions/files involved are displayed.

To limit the number of revisions displayed, refine the list of revisions displayed.

2. Click **OK**.

You are returned to the calling view, with the selected revision number displays in Version combobox.

## Refining the list of revisions displayed

When in the SVN Log Select revision dialog, you can refine the revision list displayed at the top of the dialog.

Complete this procedure to refine the revisions displayed in the SVN Log Select revision dialog.

1. In the Selection Range, enter the upper and lower revision number to define the range of revisions to display.
2. Click **Fetch Revisions**. The selected revisions display.
3. To add the next 100 revisions to your list, click **Fetch Next 100**.

---

## Accessing Source Code Management

### SVN Views

There are multiple views available that are related to SVN and the operations that you can do. Use the **Window >> Views** main menu option to display any views that are not visible.

- SVN Log
- SVN Repository Browser
- SVN Status

### SCM Menus

There are two menus for working with SCM: The SCM menu from Studio and the SCM context menu, accessed when you right-click on a file.

The Studio SCM menu provides the following options:

- Checkout
- Show Log
- Review Changes
- Merge/Revert
- Copy
- Switch
- Apply Patch
- Browse Repository

The SCM context menu, accessed by right-clicking on a folder or file in the Projects view, File Browser view, or one of the SVN views, displays options relevant for the selected folder. Select SCM from the context menu, then select the option you wish to complete.

## SVN Log

### Viewing log information

Complete this procedure to view information about the revisions for a specified repository.

1. Select **SCM >> Show Log**.



2. Enter the URL of the repository. Click the Browse repository icon to display the Repository browser.
3. Select the desired version.
4. Click the Fetch the revisions in the revision range icon (single blue arrow). The SVN Log view displays. The Revisions, Actions, Author, Date, and Messages are displayed in the log. The icons for Actions indicate the status.
5. To limit the list of revisions displayed, open and complete the Revision log dialog.
6. Select a specific revision to display information about that revision.
7. The comments and actions/files associated with that revision are displayed.

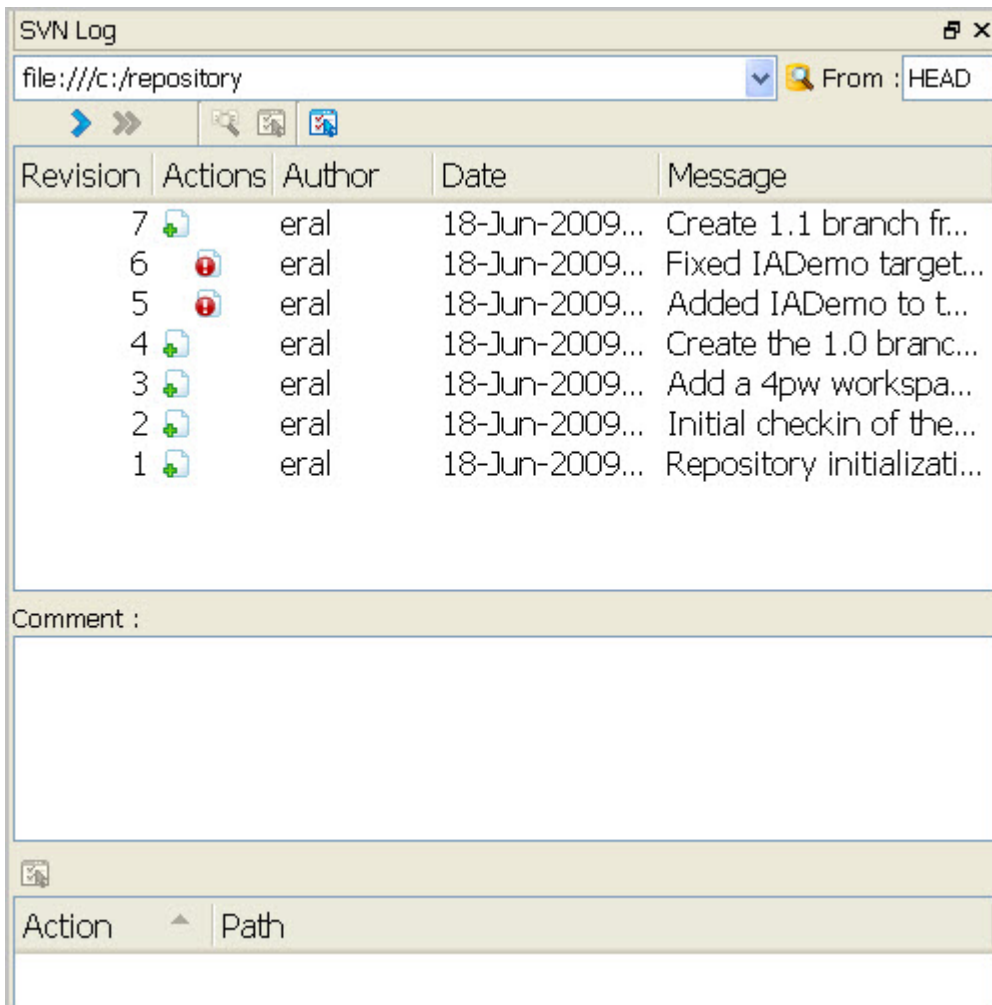


Figure 12-5. SVN Log dialog

### Specify the revision range

From the SVN Log view, you can select the range of revision logs to display using the Revision log dialog.

1. Click the Open revision log options icon.
2. Provide the range in the from: and to: fields, respectively.
3. Click **OK**.

The SVN Log view is updated to only display those revisions in the range specified.

## SVN Repository Browser

The SVN Repository Browser lists the files in your selected repository and branch.

### Specify a repository and branch

1. Open the SVN Repository Browser view (select **Window >> Views >> SVN Repository Browser**).
2. Enter the URL of the repository.
3. Select the desired version.
4. To select a specific revision within the version, click the Select Revision icon. The SVN Log Select revision dialog will guide you through the steps.
5. Click the Refresh icon.

The SVN Repository Browser displays the contents of the selected repository and version.

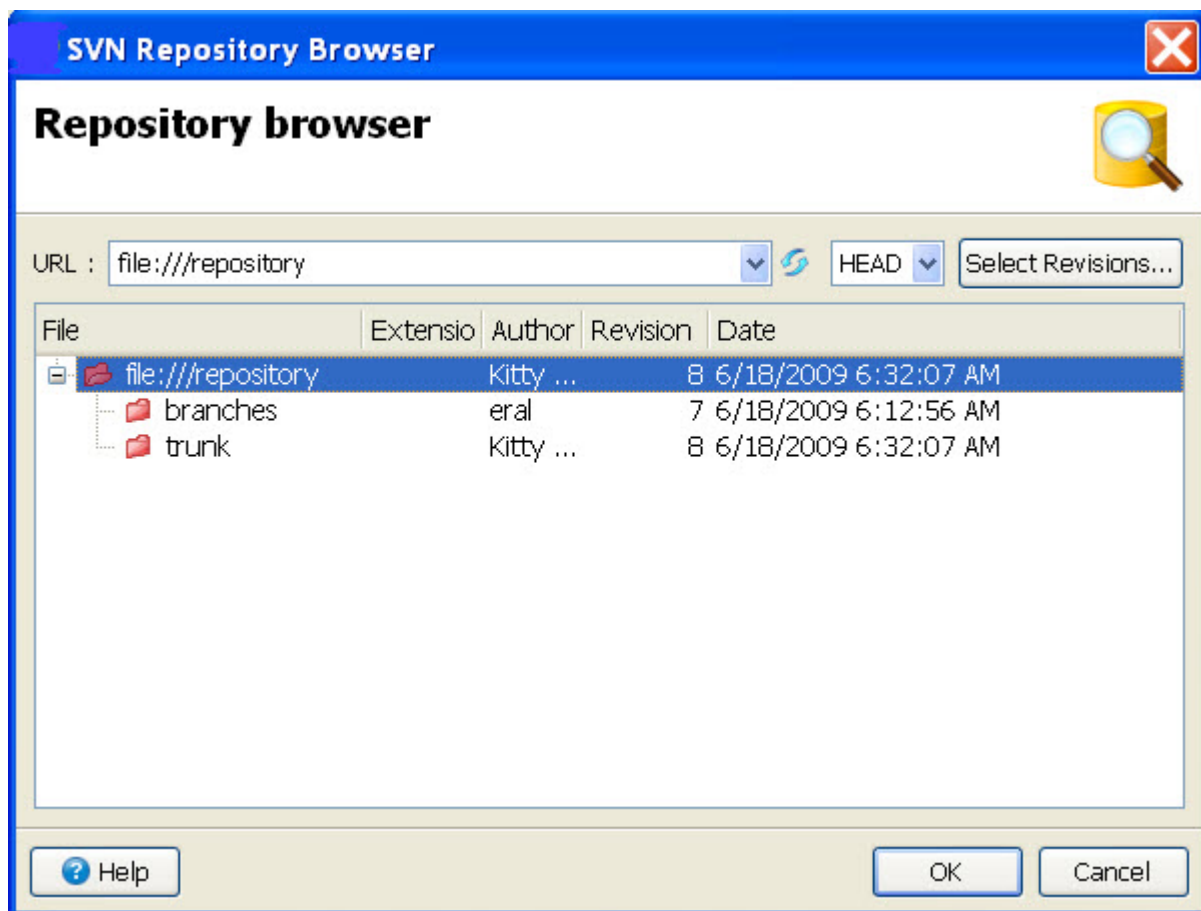


Figure 12-6. SVN Repository Browser

## SVN Status

The **SVN Status** view displays information about changes to your working copy.

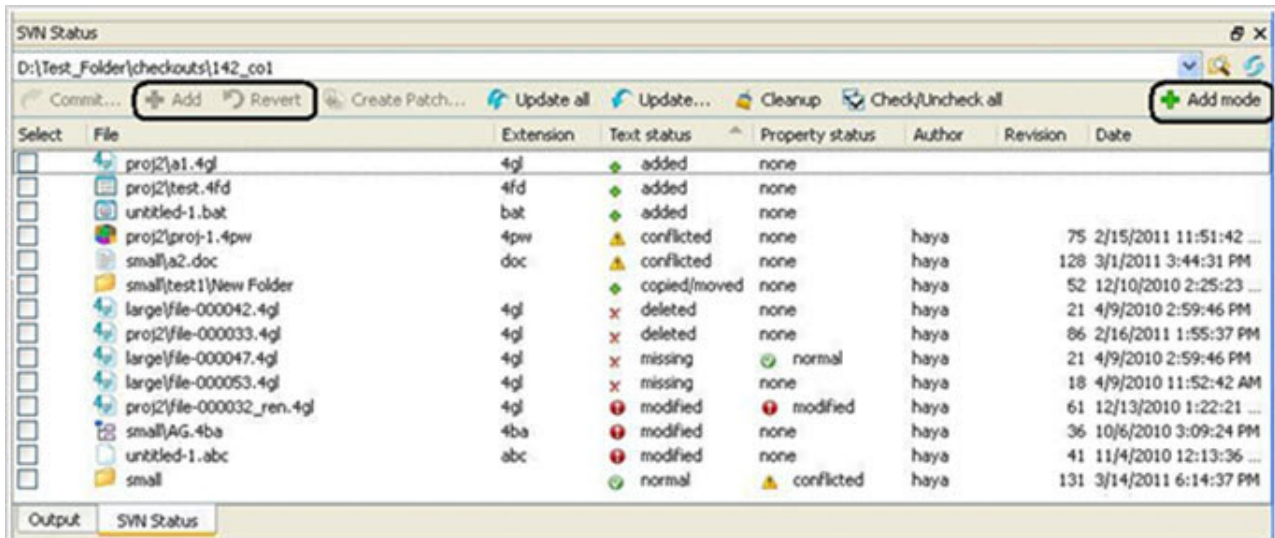


Figure 12-7. SVN Status view

The text status column of the file can indicate:

- **added** - the file has been added to the working copy and needs only to be committed.
- **modified** - the file has been modified in the checkout directory and the update has not yet been committed.
- **normal** - a committed file in the repository.
- **unversioned** - the file has not yet been added to the repository. After being added, it should be committed.
- **not working copy** - all the files within an unversioned folder will have this text status.
- **conflicted** - the file was modified in both the repository and checkout directory, and Subversion cannot resolve the conflict. Use the menu option Merge.
- **missing** - the file has been deleted from the checkout directory. If you commit a file marked as missing, you remove it from the repository.
- **incomplete** - if the checkout has stopped without completion, the directory will have this status.

Toolbar icons allow you to execute commands on the selected files. The files displayed, and the icons enabled, depend on the mode and the status of the files selected.

### SVN Status modes

The SVN Status view has two modes:

- **Commit mode** - Files that can be committed are displayed. Files that cannot be committed but are not supported in Genero Studio are displayed (with the exception of files with 'ignored' and 'external' status). Unversioned files are not displayed. This is the default mode.



- **Add mode** - Files with a status of added, copied/moved, unversioned, or 'not a working copy' are displayed.

To switch between modes, click the **Add mode** button.

---

## Working with SVN

### Adding Files

Complete this procedure to add unversioned files found in the current working directory.

1. Open the SVN Status View.
2. Select your working directory.
3. Click **Add mode**. All unversioned files found in the current repository are listed. The files/folders with 'not a working copy' text status or none property status within an unversioned folder will also be shown.
4. Check those files you wish to add to the repository.
5. Click the **Add** button. The files are added, their text status changes to 'added'. They are not yet committed.
6. To commit these newly added files, see “Commit/Review Changes” on page 12-12.
7. To revert this action (for files that have been added but not committed), see “Revert changes / Unadd files” on page 12-13.

### Deleting Files

To delete a file from your working folder and from the repository:

1. Right-click the file.
2. Select Delete from Disk from the context menu. Your local copy is deleted. The next time you commit, the Projects view will request the source code management system delete the file from its repository.

### Update/Update All

Update / Update All updates any outdated files in your project with the latest version stored in the repository. The Update options are presented as buttons in the Status view toolbar, or can be accessed from the SCM context menu for a specific file selected from the project tree in the Projects view.

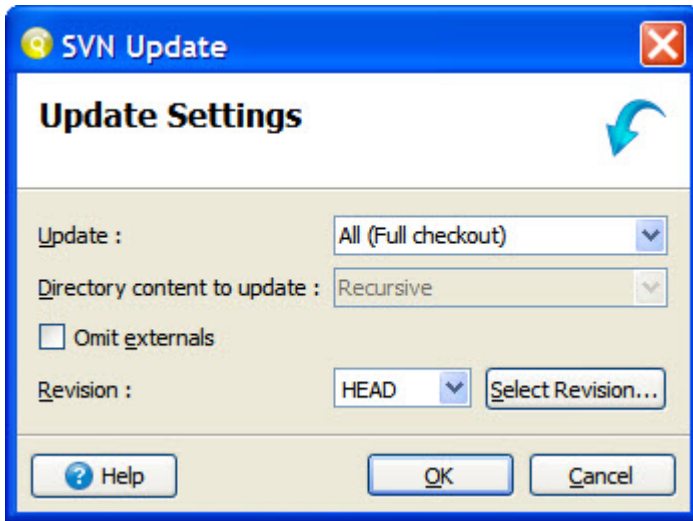


Figure 12-8. SVN Update dialog

The **Update** field - files to be updated. The choices are:

- **All** (a full checkout)
- **Selection** (selected files )

The **Directory content to update** combobox - depth for the update:

- For an All update - the depth is automatically set to Recursive.
- For a Selection update - the choices are Recursive, None, Immediate files only, Immediate files and folders only; for updates from the Projects view using the context menu, the depth will always be None.

The **Omit externals** checkbox - check to exclude "external items".

The **Revision** combobox - select the revision to be updated.

## Commit/Review Changes

To store your updated version of a file in the repository, select the file in your Project and choose **SCM >> Review Changes** from the right-click menu to display the SVN Status view. In this view, make sure there is a check in the Select column for the updated file, and press the **Commit** button at the top of the view. This will commit any checked file.

- **Commit Comments** - Enter a comment in the text box, or select a comment from the dropdown list of recent comments.
- **Files Changed** - the file from which you invoked the menu option is marked with a check. You can check additional files to be committed. The text status of the file can be:
  - **added** - the file has been added to the repository and needs only to be committed.
  - **modified** - the file has been modified in the checkout directory and the update has not yet been committed.
  - **unversioned** - the file has not yet been added to the repository. After being added, it should be committed.
  - **conflicted** - the file was modified in both the repository and checkout directory, and Subversion cannot resolve the conflict. Use the menu option Merge.

- **missing** - the file has been deleted from the checkout directory. If you commit a file marked as missing, it will be skipped during commit and will remain as it is.

## Revert changes / Unadd files

You can discard changes to files not yet committed, and you can unadd files which have been added but not yet committed.

1. Open the SVN Status view. If you are in Commit mode, the Revert button displays. If you are in Add mode, the Unadd button displays.
2. Check those files you wish to revert or unadd.
3. Click **Revert** (Commit mode) or **Unadd** (Add mode).
4. A dialog asks you to confirm your selection. Click **Yes**.

For modified files, the change is reverted and the previously committed version of the files is restored to your local (working) copy. For added files, the file is changed from 'added' to 'unversioned'.

## Diff with Current Revision

Ctrl-Click to select two revisions of a file in the log. Select **Diff** from the right-click Menu to call the Graphical Differential utility to compare the committed file with the file that you have revised.

## Cleanup

The Cleanup option cleans up the working copy, removing stale locks.

1. In the Projects or File Browser view, right-click on a file or directory.
2. Select **SCM >> Cleanup**.

## Copying SVN Files and Directories

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can create another copy of your project, project files, or directories, to allow parallel development, for example.

## Copying directories and files from the File Browser

1. Open the File Browser view.
2. Navigate to the working directory you wish to copy.
3. Right-click on the directory and select **SCM >> Copy**.  
The SVN Copy dialog displays with the From section populated.
4. Complete the SVN copy dialog.

## Copying directories and files from the SCM Menu

1. Select **SCM >> Copy**  
The SVN Copy dialog displays.
2. Complete the SVN copy dialog.

## The SVN Copy dialog

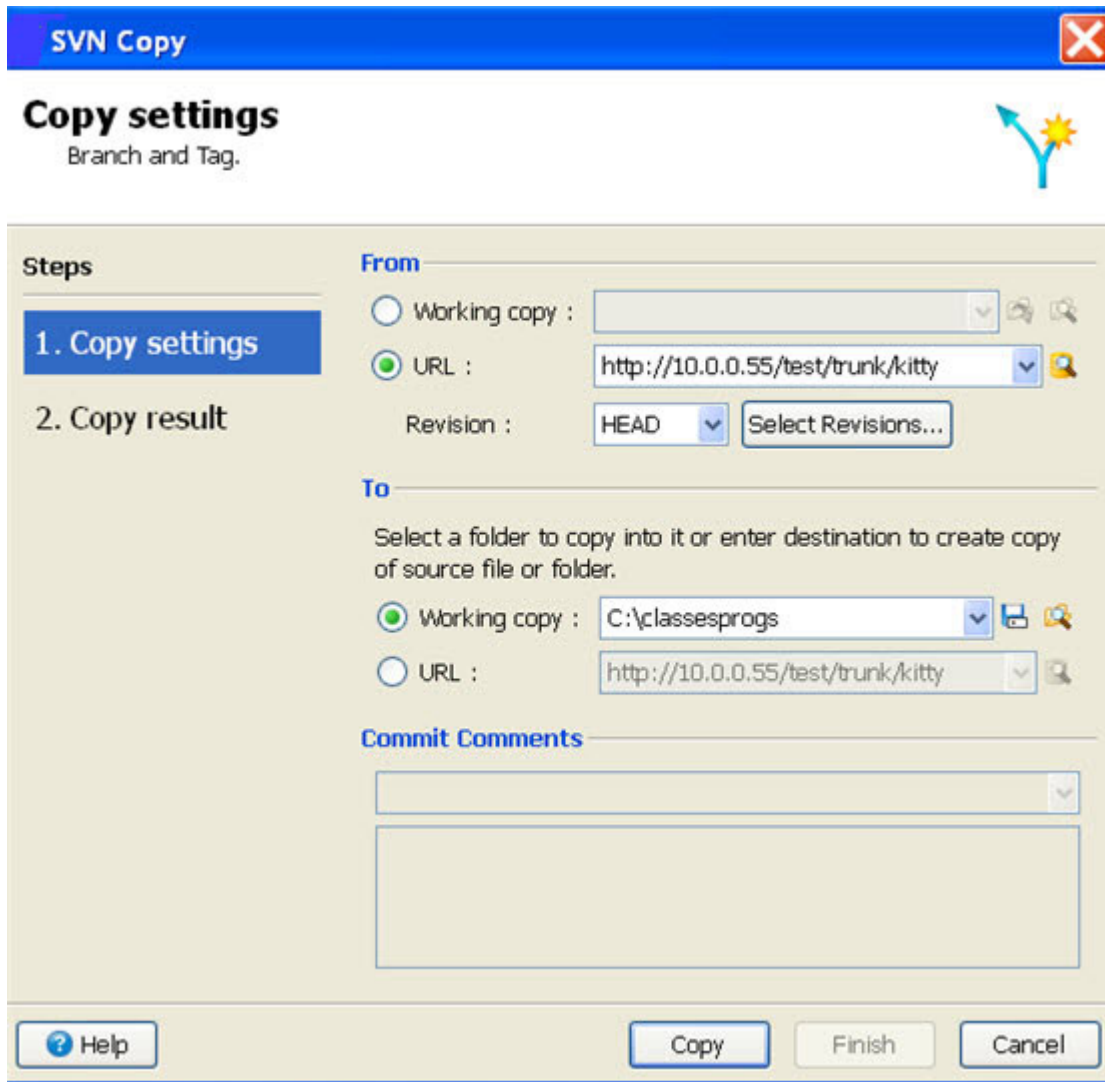


Figure 12-9. SVN Copy dialog

Select the corresponding radio buttons (Working Copy/URL ) to indicate that you will:

- Copy the working copy of the project file from which you invoked the Source Code Control menu to a different repository having the URL that you have entered. Select the revision (HEAD is the default).
- Copy the selected file from a repository having the URL that you have entered, to a working copy having the location that you have entered. Select the revision (HEAD is the default).
- Copy the selected file from one repository to another. Select the revision (HEAD is the default).

The Copy result is shown.

## Merge/Revert

The SVN Merge/Revert dialog lets you merge or revert revisions.

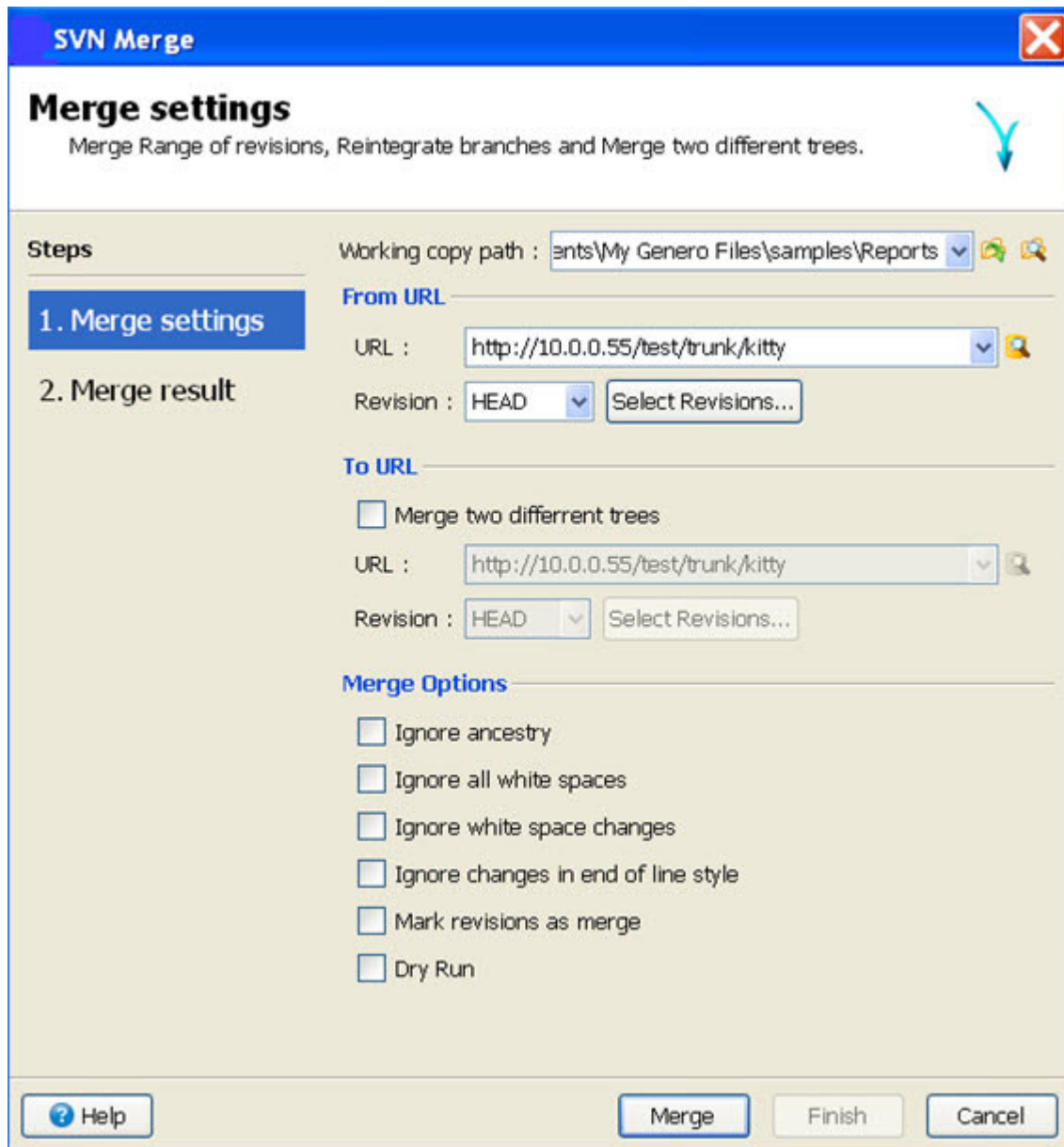


Figure 12-10. SVN Merge dialog

- **URL** - Enter the URL of the repository and select the Revision number, or HEAD for the most recent revision
- **To URL** - Enter the URL of the repository and Revision number at which you want to end, or click the **Use the start URL** button to continue to the end.

To access the dialog:

- Select **SCM >> Merge/Revert** from the Studio menu.
- Right-click on a file and select **SCM >> Merge/Revert** from the context menu.
- Right-click on a revision entry from the SVN Log view and select either Revert changes from selected revisions OR Revert to this revision

You can revert the changes from a single revision, or you can revert the changes across a range of revisions.

## Revert from a single revision

To revert changes from a single revision:

1. Enter the repository URL and version, leaving the Range ends with a different URL check box not selected.
2. Specify the working copy folder to be updated in the working copy path.
3. Select the desired merge options
4. Click **Merge/Revert**.

## Merge and revert from a range of revisions

To merge and revert changes from a range of revisions:

1. Enter the repository URL and Revision
2. Check the Range ends with a different URL check box and specify the revision URL and version to close the range.
3. Specify the working copy folder to be updated in the working copy path.
4. Select the desired merge options
5. Click **Merge/Revert**.

## Switch (Moving a working copy)

This subcommand allows you to move a working copy to a new branch of the repository.

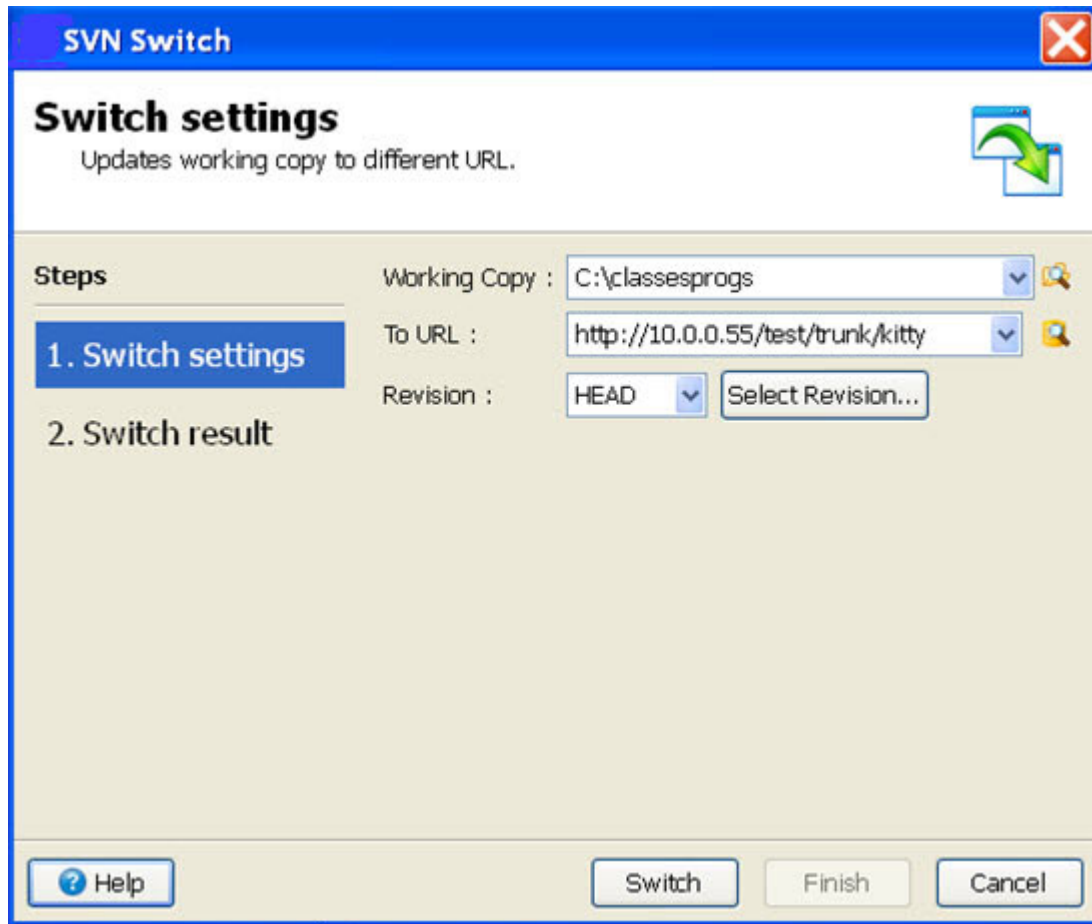


Figure 12-11. SVN Switch dialog

## Create Patch

From the **SVN Status** view, this option allows you to create a **patch file** that contains all the differences between your working copies and the corresponding files in the repository. The created file is in *Unidiff* format.

If one of the checked files is unversioned or missing, the Create Patch command is disabled.

To create the patch file:

1. In the **SVN Status View**, check the file(s) that contain the modifications to be included in the patch.
2. Click **Create Patch**.
3. In the Save file dialog, select a directory and enter a name for the patch file.

The patch file is created and opened in Code Editor.

## Apply Patch

From the **SCM** main menu and the **File Browser** context menu, this command allows you to apply a patch file to files in the repository. Only text modifications will be applied.

Patch file fragment:

```
1 index: bz1_col/al.txt
2 =====
3 --- bz1_col/al.txt (revision 95)
4 +++ bz1_col/al.txt (working copy)
```

Enter the information in the **SVN Apply Patch** dialog:

- **Root dir:** the path of the directory where the patch is to be applied.
- **Patch file:** the path of the selected patch file.
- **Skip directories:** integer value defining the directory level to be search for the file to be patched. Using the patch file as an example, if this entry is set to "0", the complete path will be used to find the file; if set to 1, only al.txt will be used to find the file.
- **Create file if not existing:** if the file to be patched isn't present in the current directory path, the corresponding file will be created as an unversioned file. If the file is unchecked in the Status view, the corresponding file will not be created.
- **Test apply patch:** displays the output of the **Apply Patch** command without actually applying it to the files; files will not be modified.
- **Show Files:** opens a dialog displaying a list of all the files that will be patched.

When the **Apply** button is clicked, a page containing the output of the command will be displayed. The output will indicate Successful when all the files from the patch file are successfully patched. If only some of the files are patched successfully, the output will indicate Aborted.

**Note:** This feature is available for Windows and Linux users only.

## Manage Properties

You can make changes to SVN properties through the context menu option **SCM >> Properties**. Use the integrated toolbar to add, modify, or delete properties. See your Subversion manual for additional information about properties.



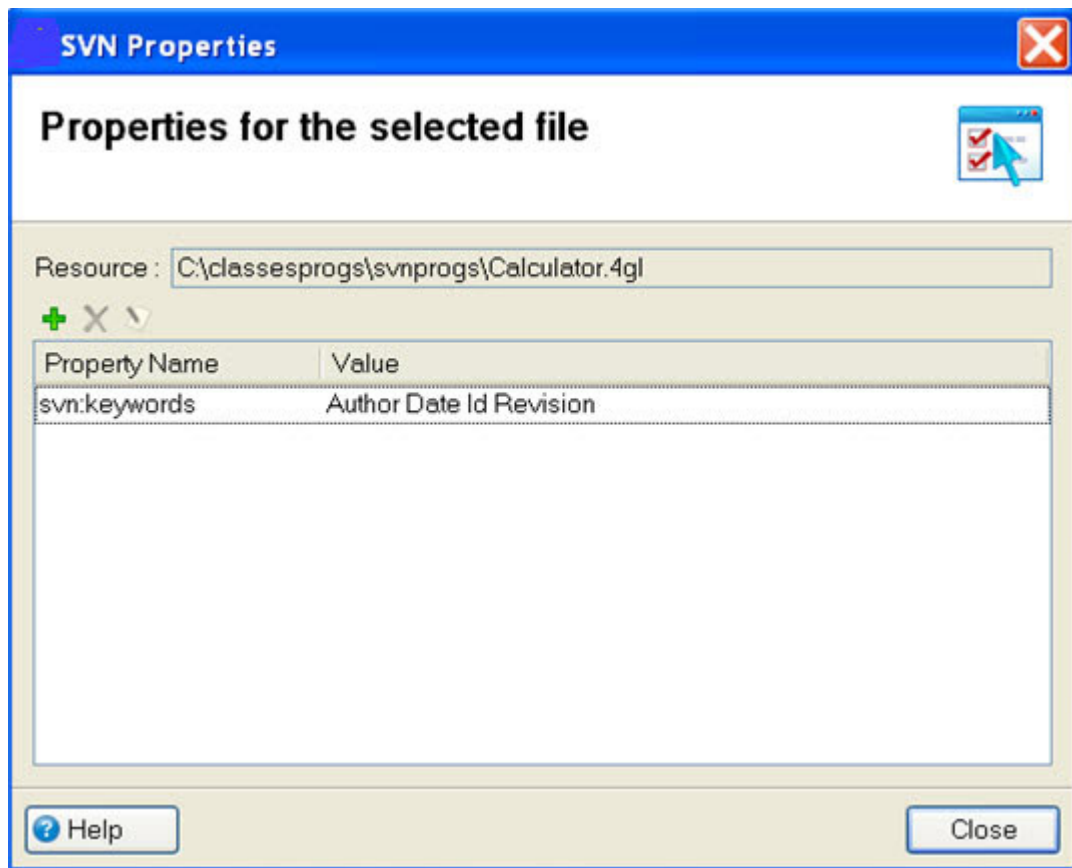


Figure 12-12. SVN Properties dialog



---

## Chapter 13. Other Studio Modules

### Topics:

- “Using Web Services with Studio”
- “Genero Studio Graphical Differential” on page 13-6
- “Genero Studio File Browser” on page 13-9

---

## Using Web Services with Studio

### Topics:

- Creating a Web Services program
- Add Web Service
- The Web Services Wizard
  - Web Services Selection
  - Web Services Project Integration
  - Advanced Code Generation Options
  - Code Generation Result (optional page)
- Building the Application
- Update WSDL

## Creating a Web Services program

You can use Genero Studio to write, compile, and execute a Genero Web Services (GWS) client or server application:

1. Add an application, library, or folder node to a project in Genero Studio Project Manager.
2. Right-click the node, and select **Add Web Services**. This calls the Web Services Wizard, to assist you in adding the Web Services client or server information to the application.
3. Use Studio Code Editor to create or modify a Genero application that calls a Web Service (client) or defines a Web Service (server). Use the WSDL information to determine the service name and functions. Code Editor recognizes and parses the syntax of BDL functions related to Genero Web Services. *See the Genero Web Services User Guide* for details on coding your Web services clients and servers.
4. In Studio Project Manager, add the WSHelper.42m file to the external dependency property of your Genero application; this compiled file contains functions used internally for Web Services.
5. Use the Studio Build menu option to compile and link your new application.

If you must access the web service using a proxy, provide the proxy information using the Studio main menu option Tools>>Preferences, Genero Studio.

## Add Web Service

To access or define a Web service, you must have the WSDL description created by the service provider. You can use the WSDL description to determine the operations of the Web Service to use in your BDL program, the required parameters, and the return values.

To add a Web service:

1. Right-click an **Application**, **folder** or **library node** in Genero Project Manager and select **Add Web Service**. The Web Services Wizard displays.
2. Choose between:
  - a. **Client Consuming WSDL** - to retrieve the WSDL information and generate a .4gl file containing the BDL functions for the operations in the WSDL, which can be called by a GWS client application.
  - b. **Server Implementing WSDL** - to retrieve the WSDL information and generate a .4gl file (GWS Server Application) containing the BDL functions to create and publish a Web Service which can be accessed over the web.

The Web Services Wizard guides you through the steps to add the necessary service information to your application:

### The Web Services Wizard

#### Web Services selection Page

Provide the Service details to get the WSDL description, using one of the following options:

- **URL** - Provide the URL of the web service. Example: `http://servername:port/servicename?WSDL`
- **File** - Provide the name and location of a WSDL file in your file system.

Click the **View** button. The description of the Web Service is retrieved from the WSDL and displayed.

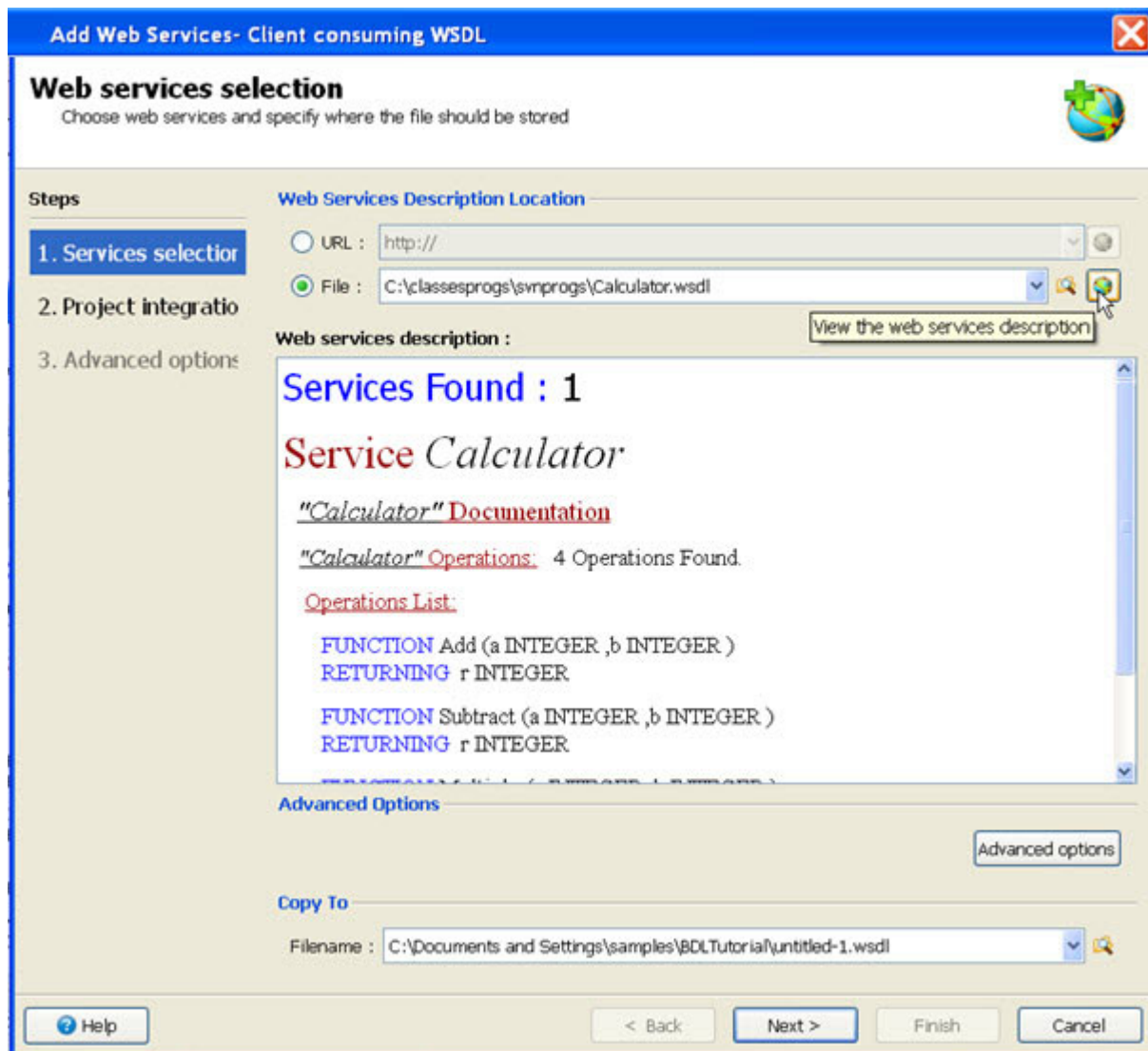


Figure 13-1. Add Web Services - Client consuming WSDL dialog

**Copy to** - Use the Browse button to specify the directory where a copy of the WSDL file should be saved. If you used the File option to specify the WSDL file, this directory must be different.

**Advanced Options button** - if you select this button, a dialog displays that allows you to add an external schema file with the WSDL file.

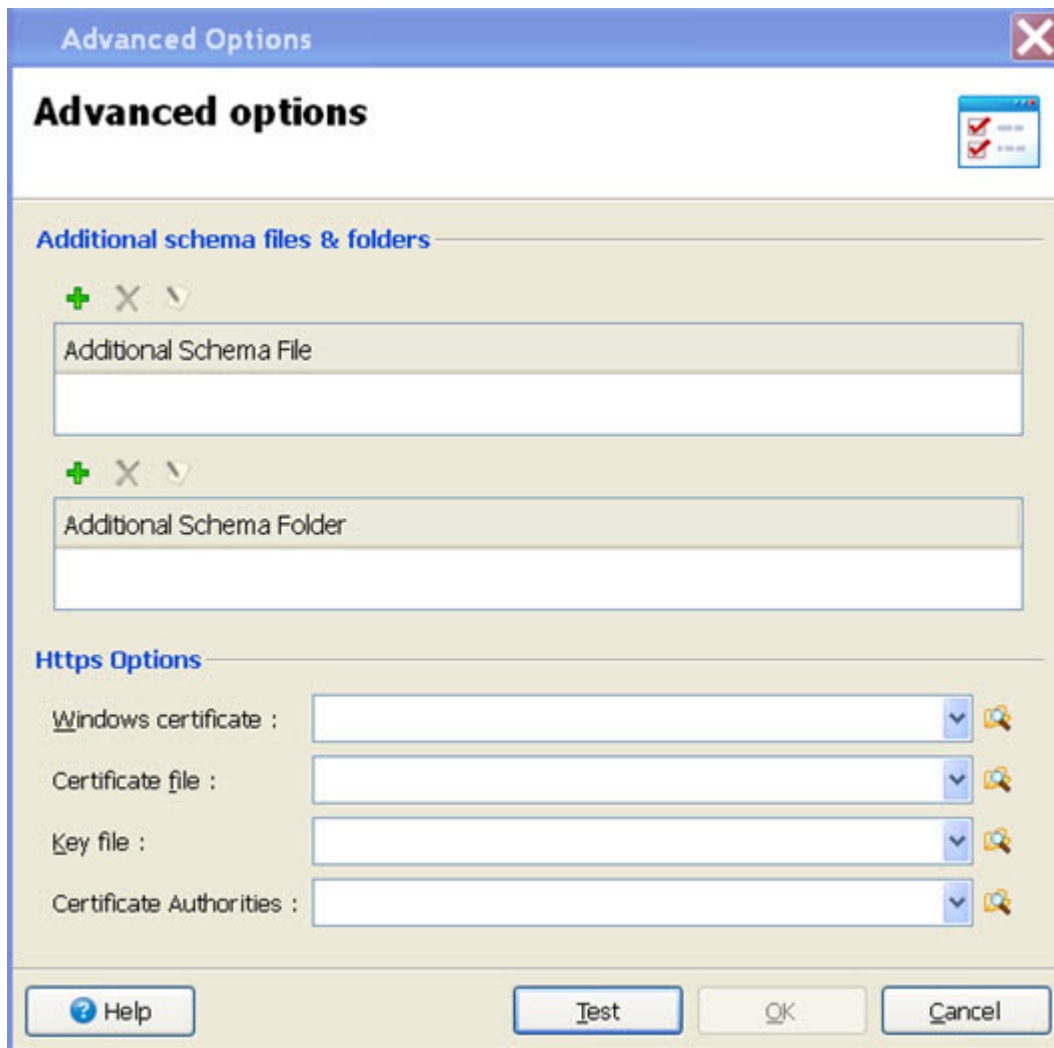


Figure 13-2. Add Web Services - Client consuming WSDL, Advanced options

Within the Advanced Options dialog:

- **Additional Schema File** - press the **Add** icon to add an external schema file. An Add dialog opens, allowing you to search for the file. Other icons allow you to delete a previously added file, or to edit it.
- **Additional Schema Folder**- press the **Add** icon to add the path and filename of the schema folder.
- **Https Options** - Set the options needed for the fglwsdl tool, which implements adding the external schema.

## Web Services Project Integration (Client only)

**Add Web Services to Project** - if this radio button is selected, the WSDL file will be added to your project in Project Manager. Files that show the Web Services operations as functions in a Genero Program are generated and stored in the Target directory of the application:

- **.inc file** - the globals file containing the definitions of the input and output records, and the prototypes of the operations.

- **.4gl file** - contains the definitions of the functions for your GWS client or server application. and the code that manages the Web Service request or publishes a service,

**Advanced** - this radio button gives you the flexibility to write your own web client stub. The .4gl and .inc files are generated and added to your project in Project manager. The wsdl file is created in the "copy to" path.

**Important:** Do not modify the generated .4gl files; they are regenerated on each re-build, so modifications will be lost.

## Advanced Code Generation Options

Select the options for code generation and runtime behavior, if you wish to modify the default behavior. See The fglwsdl tool for a complete list of the command options.

- **Select service/binding** - Choose from the list of all Services or binding in the generated wsdl file.
- **Behavior and Functionality**
  - **Compatible with Genero 1.xx** - Implements the "-compatibility" option of the Genero tool fglwsdl.
  - **Force RPC convention** (applies to RPC only) - Implements "-fRPC" option of fglwsdl.
  - **Generate choice records for inherited types** - Implements "-fInheritance" option of fglwsdl.
  - **Ignore SOAP faults** - Implements "-ignoreFaults" option of fglwsdl.
  - **Use WS-addressing** - Allows support of WS-Addressing (only WS-Addressing 1.0 is supported.)
  - **Check data type constraints (facets) for simple types** - Implements "-noFacets" option of fglwsdl. By default facet is present, so this option is enabled.
  - **Generate XML list of arrays** - Implements "-fArray" option of fglwsdl.
  - **Generate DOM callback handlers** - Implements "-domHandler" option of fglwsdl.
  - **Generate fglprofile aliases for server URL** Generates FGLPROFILE Logical names in place of URLs for the client stub - implements "-alias" option of fglwsdl.
- **Coding Conventions & Style**
  - **Include service documentation** - Implements "-comments" option of fglwsdl.
  - **Prefix functions, variables, and types with** - Implements "-prefix" option of fglwsdl.
  - **Prefix variables and types with namespace** - Implements "-autoNsPrefix" option of fglwsdl. Specify the number of words of the namespace that are to be excluded from the prefix. The default prefix is the full namespace, and you may exclude up to nine words. After the ninth word, you can only add the last word of the namespace as the prefix.
  - **Define prefix for variables and types per namespace** - Implements "-nsPrefix" option of fglwsdl. For each namespace, you can define a prefix for variables and types.
- **Command Line** - additional arguments for the execution of the fglwsdl tool.

The command line is read-only, and is automatically updated when you select an option.

### Code Generation Result - optional page

This page displays only if you have selected **Advanced** in the Web Services Project Integration page of the wizard.

Select **Finish** to generate the WSDL file and accompanying files.

**Note:** If you must access the web service using a proxy, provide the proxy information using the Studio main menu option Tools>>Preferences, Genero Studio.

## Building the Application

Build Rules are defined to link the generated .4gl file into your application. The generated .4gl file in the Target Directory should not be modified, as your changes will be lost if the file is regenerated.

**Important:** The Genero library file **WSHelper.42m** must be linked into every application using Web Services. Add the library file to the application node in Project Manager as an external dependency property, prior to building the application.

When you build the Project, the appropriate files are compiled and linked.

## Update the WSDL

An internet connection is required, unless the Web Services server is local.

Complete this procedure to update the WSDL.

Right-click the WSDL node in the project structure to update the WSDL file.

While updating the file, any new dependency files are also downloaded.

---

## Genero Studio Graphical Differential

**Genero Studio** provides a basic "diff" tool to identify the differences between two files. Launch **Graphical Differential** from the **Tools** menu or use the toolbar icon.

**Topics:**

- Graphical Differential page
- Output
- Using an External tool

### Graphical Differential tabbed page

- **Base file** - enter the name and path of the first file, or use the ... browse button to search for it in the file system
- **File to compare** - enter the name and path of the second file, or use the ... browse button to search for it in the file system



- **Synchronized scroll bars** - A check box "Synchronize scroll bars", bottom left, defines the behavior of the two files when scrolling. If checked, both files are scrolled simultaneously; if unchecked, each file is scrolled independently

Once the two files are selected, select the **Show** button to start the comparison process. The base file and the compared file labels are updated when user clicks the "Show" button. The differences between the two files are displayed with color indicators.

## Output

The default output is "Friendly format (two views)":

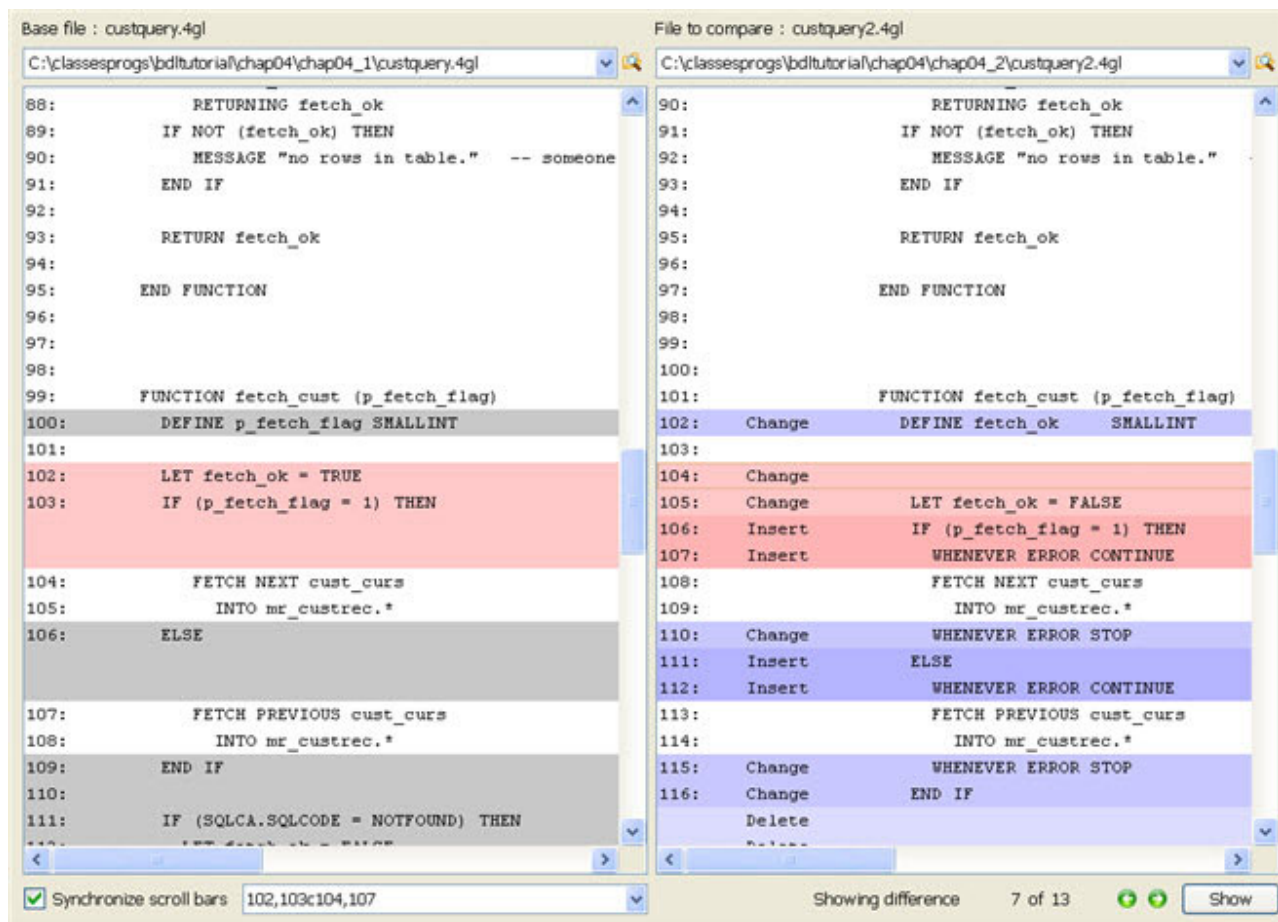


Figure 13-3. Friendly format (two views)

The files are listed side-by-side, with the line numbers for each file in the left margin. In the file being compared (right side), each difference is marked with

- **Insert:** for lines that don't exist in the base file (**dark blue lines**)
- **Change:** for lines that exist but with some alterations (**middle blue lines**)
- **Delete:** for missing lines (**light blue lines**)

Use the arrow **buttons** to navigate to the differences; the selected difference will be highlighted in pink:

- Insertions are highlighted (**dark pink lines**)

- Changes are highlighted (**middle pink lines**)
- Deletions are highlighted in (**light pink lines**)

You can also select the difference from the **dropdown list**, containing all differences by line numbers. The Line number code in this list is:

- The first number / range of numbers indicates reference lines in the base file.
- The letter indicates status in compared file:
  - "a" for *added*
  - "c" for *changed*
  - "d" for *deleted*
- The next number / range of numbers indicates corresponding lines in the compared file.

To change the format, use the **View** menu options, or right-click in the tabbed window to display the context menu options:

- **Diff format** - the differences are displayed in a single list instead of side-by-side. Each group of differences begins with the line number code. The "<" sign indicates the lines located in the first file, and the ">" sign indicates the lines located in the second. Each difference block is separated by one blank line.

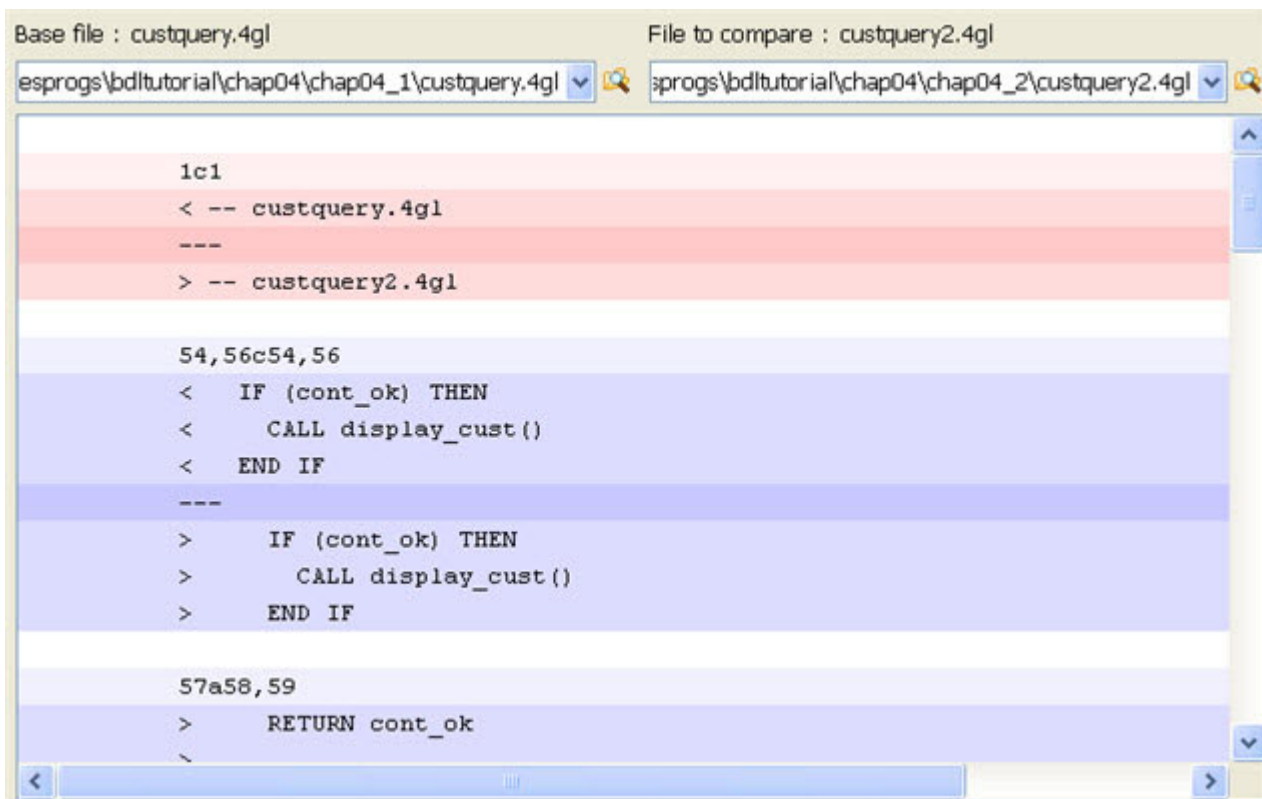


Figure 13-4. Diff format

- **Friendly format (single view)** - Both files are displayed in a single list instead of side-by-side. The corresponding line numbers for each file are shown in the left margin. The "<" sign indicates the lines located in the first file, and the ">" sign indicates the lines located in the second.

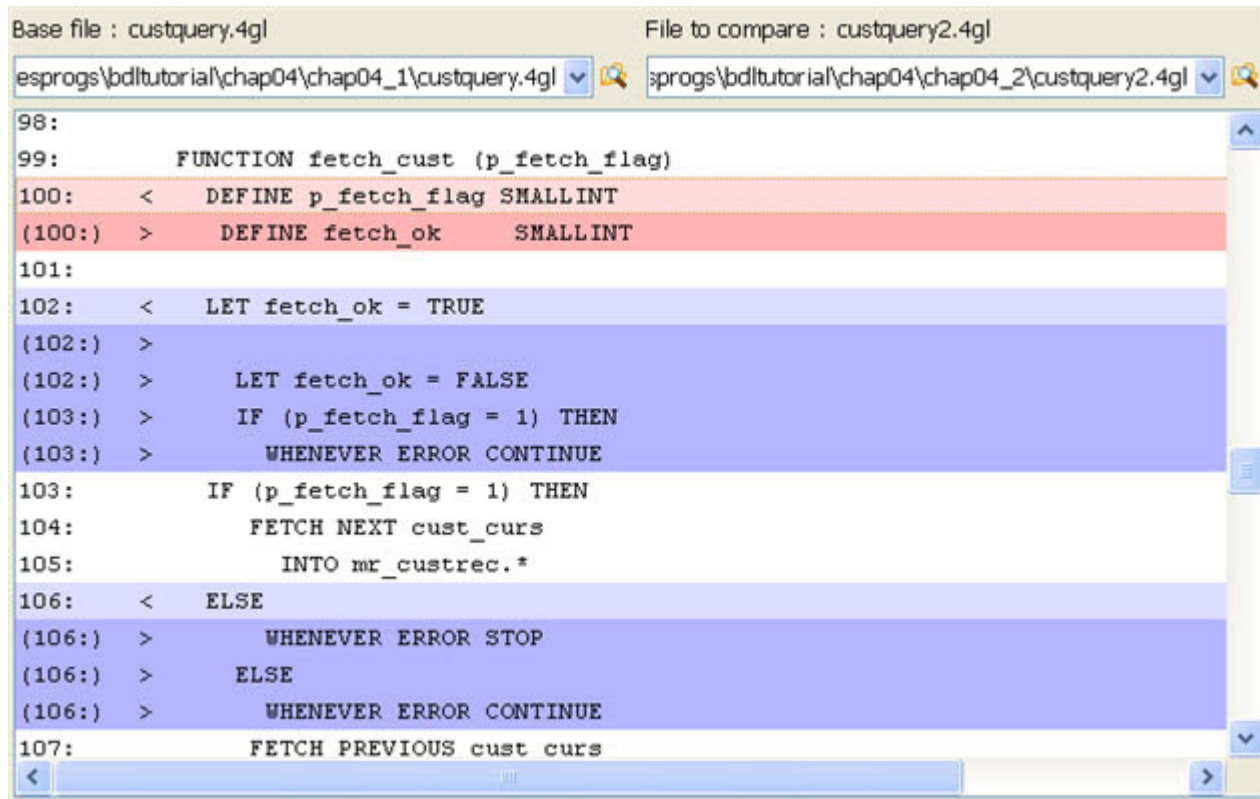


Figure 13-5. Friendly format (single view)

Additional menu options allow you to open the base file or the compared file in Code Editor.

## Using an External tool

You can specify an external tool instead of the integrated one to compare files. Use **Tools >> Preferences** and select the Graphical Differential page from the list.

To invoke the external tool, select two files in Project Manager or the File Browser, and choose the comparison from the right-click context menu. The integrated Studio Graphical Differential will still be invoked if you use the **Tools >> Graphical Differential** menu option.

## Genro Studio File Browser

**File Browser** is a tool to navigate your local file system.

**Topics:**

- File Browser
- File Selection
- Managing Files

### File Browser

Use **Tools>>File Browser** to open the File Browser.

The folders of the file structure are displayed as a tree. Icons indicate the folders and types of the files. If the file is under Version Control, the status is indicated on the icon.

Use the + and - icons to expand the branches of the tree. Scroll the display to the right to see the additional columns of information in this view.

The integrated toolbar includes options for navigating the tree and refreshing the directory. Navigation history is maintained across sessions of Genero Studio and accessed by the drop down arrows of the previous and next folder buttons in toolbar.

Show hidden files in the tree by selecting the Show Hidden Files checkbox in **Tools >> Preferences, File Browser**.

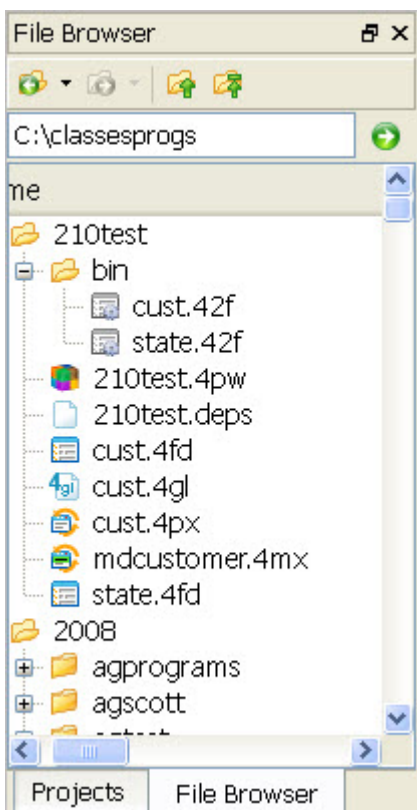


Figure 13-6. File Browser View

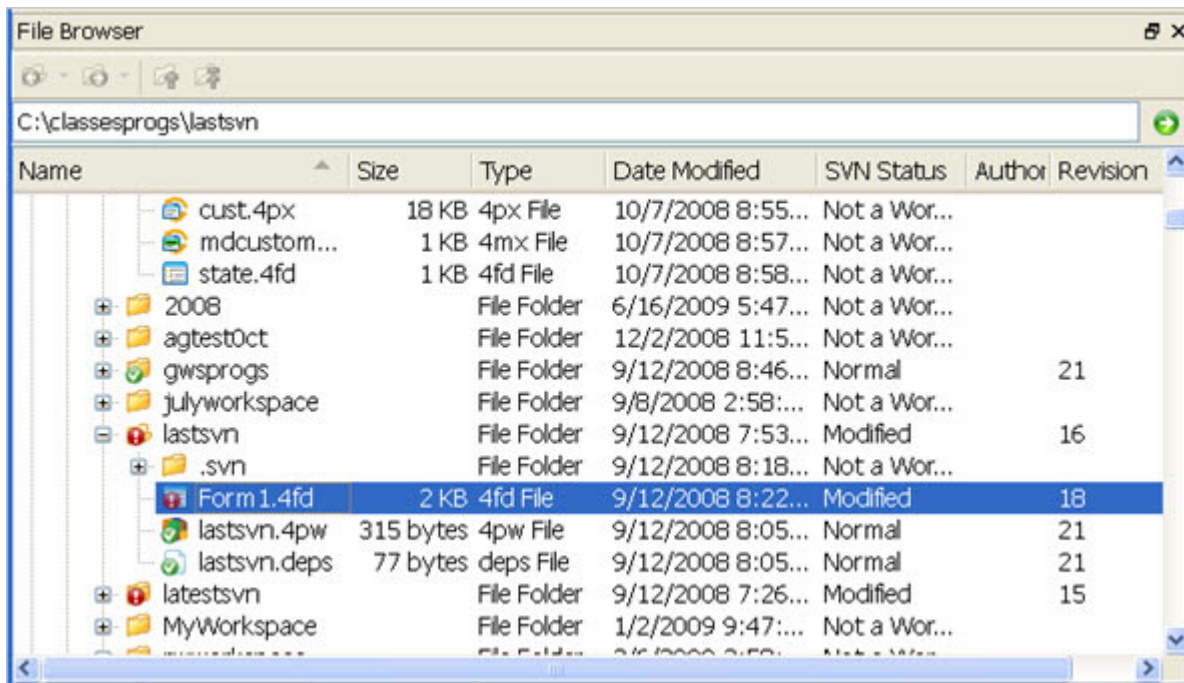


Figure 13-7. File Browser View Expanded

## Selecting Files

Click the filename to select it.

To open a Genero file using the Studio module associated with it, double-click the filename.

To select consecutive files or folders, click the first item, press and hold down SHIFT, and then click the last item. To select nonconsecutive files or folders, press and hold down CTRL, and then click each item.

## Managing Files

You can use the mouse to drag and drop folders and files from one folder into another.

Select a file listing, and right-click to display the **actions** available for that file (**Open**, **Cut**, **Copy**, **Paste**, **Rename**, **Delete**, and so on), based on the mime type of the file.

The **New file** option allows you to create a new document. See the File >> New main menu option for additional information about creating new files.

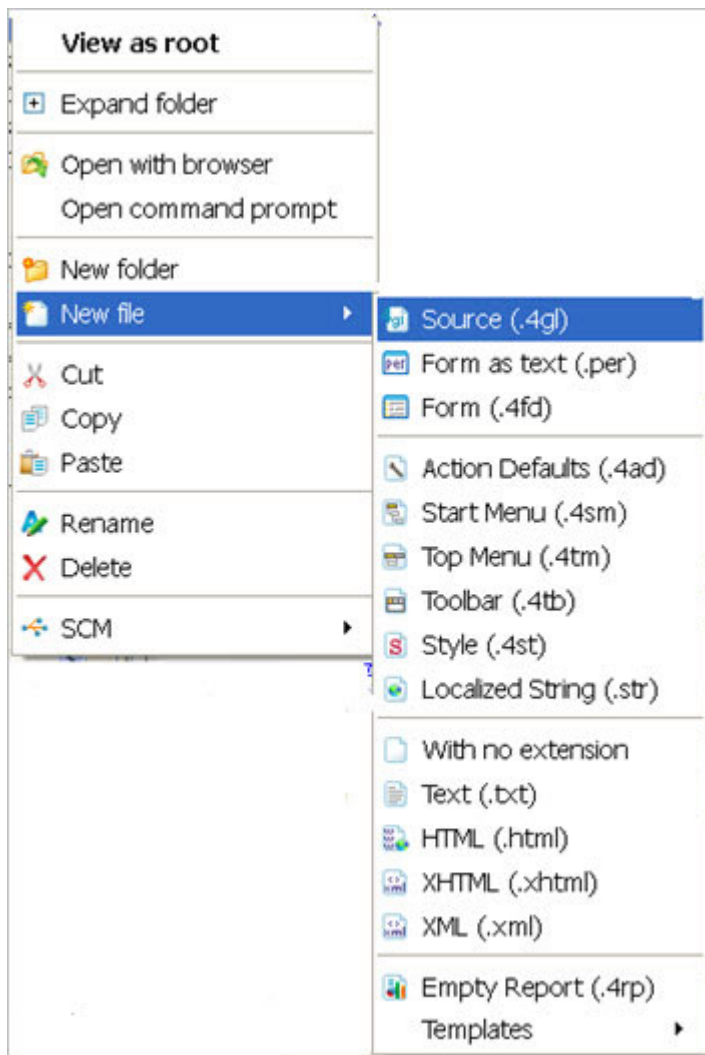


Figure 13-8. The New File menu options

You can associate file types handled by Genero Studio applications with predefined actions, and create user-defined actions, using **Tools >> Preferences, Genero Studio** and then select **File Associations** or **User Actions**.

---

## Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

---

### Accessibility features for IBM Informix products

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

#### Accessibility features

The following list includes the major accessibility features in IBM Informix products. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

#### Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

#### Related accessibility information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

#### IBM and accessibility

See the *IBM Accessibility Center* at <http://www.ibm.com/able> for more information about the IBM commitment to accessibility.





---

## Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
1623-14, Shimotsuruma, Yamato-shi  
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM web sites are provided for convenience only and do not in any manner serve as an endorsement of those web sites. The materials at those web sites are not part of the materials for this IBM product and use of those web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, any such examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Trademarks

IBM, the IBM logo, and `ibm.com`<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [www.ibm.com/legal/us/en/copytrade.shtml](http://www.ibm.com/legal/us/en/copytrade.shtml).

Four Js is a registered trademark of Four Js Development Tools Ltd.

Genero and its logo are registered trademarks of Four Js Development Tools Europe Ltd.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.







Product Number: 5725-D13

Printed in USA

SC27-3856-00

