

Lycia Development Suite



Lycia II Developer Guide

Version 2.3 – July 2014

Revision number 3.00

Querix Lycia II

Lycia Developer Guide

Part Number: 004-023-300-014

Gemma Davis, Elena Krivtsova
Technical Writers

Last Updated July 2014



'Lycia' Developers Guide

Copyright © 2006-2014 Querix Ltd. All rights reserved.

Part Number: 004-023-300-014

Published by:

Querix (UK) Limited. 50 The Avenue, Southampton, Hampshire,
SO17 1XQ, UK

Publication history:

April 2009:	Beta edition
June 2010:	Beta 2 edition
December 2010:	First edition
October 2011:	Updated for Lycia II
July 2014:	Updated for Lycia 2.2

Last Updated:

July 2014

Documentation written by:

Gemma Davis / Elena Krivtsova / Svitlana Ostnek / Olga Gusarenko

Notices:

The information contained within this document is subject to change without notice. If you find any problems in the documentation please submit your comments by email to documentation@querix.com.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix (UK) Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.



Table of Contents

INTRODUCTION	1
ABOUT THIS GUIDE.....	3
DOWNLOADING SOFTWARE.....	3
DOCUMENTATION AND DEMONSTRATIONS	3
<i>Documentation</i>	3
<i>Demonstration Applications</i>	3
LYCIA REQUIREMENTS	4
WHAT IS LYCIA?.....	4
SYSTEM REQUIREMENTS.....	5
INSTALLING LYCIA	6
INSTALLATION FOR WINDOWS.....	6
<i>Installation modules</i>	6
<i>GUI Server Configuration</i>	8
<i>Web Server Configuration</i>	9
<i>Uninstalling Components</i>	11
INSTALLATION FOR UNIX/LINUX.....	11
LICENSING	13
QXLM	13
<i>Getting Hardware ID</i>	13
<i>Activating a License</i>	14
<i>Viewing License Information</i>	15
<i>Deactivating a License</i>	16
<i>Getting a Trial License</i>	17
OVERVIEW OF LYCIASTUDIO	19
RUNNING THE STUDIO.....	19
LAYOUT OF LYCIASTUDIO.....	20
<i>Panels</i>	21
<i>Menus</i>	22
<i>Project Properties Menu</i>	23
<i>Main Toolbar</i>	25



PERSPECTIVES AND VIEWS	27
<i>Managing Views</i>	27
<i>Editors</i>	31
<i>Managing Perspectives</i>	34
USING LYCIASTUDIO	44
IMPORTING A 'HYDRASTUDIO' V4 PROJECT	44
IMPORTING A LYCIA PROJECT	50
<i>Importing Existing Projects into Workspace</i>	50
IMPORTING FILES, FOLDERS, PROGRAMS, AND LIBRARIES INTO AN EXISTING PROJECT	56
<i>Importing 4GL Programs and Libraries</i>	57
<i>Importing File System</i>	62
<i>Importing Archive Files</i>	67
EXPORTING A PROJECT	73
<i>Exporting a project to the file system</i>	73
<i>Exporting a project to an archive file</i>	80
<i>Exporting 4GL programs and libraries</i>	82
CREATING A LYCIA 4GL PROJECT FROM EXISTING SOURCES	84
CREATING A NEW 4GL PROJECT	88
CREATING NEW FILES	93
<i>Creating media files</i>	96
<i>Adding imported files to project requirements</i>	98
BUILDING AND COMPILEATION	100
<i>Build Configurations</i>	112
CHANGING THE DATABASE	119
CHANGING THE GUI SERVER	120
RUNNING A PROGRAM	122
<i>Run Configurations</i>	124
SEARCHING	131
<i>Searching for source files</i>	131
<i>Searching 4GL code</i>	136
<i>Search View</i>	137
COMPARING AND REPLACING SOURCE FILES	138



<i>Comparing source files</i>	139
<i>Replacing a source file</i>	143
USING LIBRARIES	144
<i>Linking an internal library</i>	149
<i>Linking an external 4GL library</i>	151
<i>Linking an external static C library</i>	156
<i>Linking an external dynamic C library</i>	157
INSTALLING NEW FEATURES AND UPDATING.....	159
<i>Installing a new plug-in</i>	159
<i>Updating the installation</i>	163
<i>Uninstalling plug-ins</i>	165
ACTION DEFAULTS.....	168
<i>Action Default Attributes</i>	168
<i>Action Defaults of Global Scope</i>	170
<i>Precedence of Action Defaults Parameters</i>	172
<i>Action Attributes Merging</i>	177
COMMAND LINE TOOLS.....	179
QBUILD	180
QEXPT	181
QFGL.....	181
QFORM.....	182
QLINK.....	182
QMSG.....	183
QXCOMPAT	184
4MAKE.....	186
USING THE FORM EDITOR	187
CONVERTING FORM FILES	187
FORMS.....	189
<i>General Form Properties</i>	189
PALETTE.....	191
<i>Containers</i>	193
<i>Form Widgets</i>	197



THE GRAPHICAL FORM EDITOR USAGE.....	201
CREATING A FORM.....	206
<i>Selecting the Root Container</i>	206
<i>Adding a Grid Container and a Button to the General Tab.....</i>	210
<i>Adding Labels and Text Fields.....</i>	211
<i>Adding RadioButtons</i>	214
<i>Adding Calendar</i>	215
<i>Adding a Picture</i>	215
<i>Creating a Screen Record.....</i>	220
<i>Adding a Stack Panel.....</i>	222
<i>Adding a Table.....</i>	223
<i>Adding Buttons to the Record Tab.....</i>	225
<i>Displaying a Form</i>	226
<i>Adding a Dialog</i>	226
USING THE DEBUGGER.....	227
PREPARING FOR DEBUGGING	227
DEBUGGING.....	229
<i>Setting Breakpoints.....</i>	229
<i>Launching Program in Debug Mode</i>	230
<i>Executing a Program Step by Step.....</i>	232
<i>Terminating and Relaunching a Program.....</i>	235
VARIABLES VIEW.....	238
<i>Monitoring Values of Variables.....</i>	239
<i>Changing the Value of a Variable.....</i>	241
<i>Customising the Variables View</i>	242
BREAKPOINTS VIEW.....	244
<i>Enabling and Disabling Breakpoints</i>	244
<i>Exporting and Importing Breakpoints.....</i>	246
<i>Grouping Breakpoints</i>	249
<i>Breakpoint Properties.....</i>	254
EXPRESSIONS VIEW AND WATCH EXPRESSIONS.....	257
WATCHPOINTS.....	260



CONFIGURATION SETTINGS	263
ENVIRONMENT VARIABLES	264
ENVIRONMENT SETTINGS - FGLPROFILE.....	265
<i>Environment variables</i>	265
<i>Toolbar</i>	265
<i>Tab Controls</i>	266
<i>Grid Settings</i>	266
<i>Miscellaneous</i>	267
<i>SSL Settings</i>	268
<i>System</i>	269
<i>GUI</i>	269
<i>Shadow options for compiler switches</i>	270
LOCALIZATION SETTINGS	271
<i>At Compilation</i>	271
<i>At Runtime</i>	273
<i>Identifiers</i>	273
<i>Numeric and Currency Settings</i>	274
<i>Date and Time Settings</i>	274
<i>Multi-Byte Values</i>	274
USING C API IN LYCIA.....	277
CREATING C SOURCES	277
CREATING A DYNAMIC C LIBRARY.....	278
<i>Using the command line</i>	278
CALLING A C FUNCTION FROM 4GL.....	280
CALLING A 4GL FUNCTION FROM C	282
THE APPLICATION SERVER	284
USING THE GUI SERVER	284
TESTING THE GUI SERVER CONNECTION.....	285
GUI SERVER SECURITY	286
<i>Lycia Encryption</i>	289
AUTHENTICATION	291
<i>Windows</i>	291



<i>Unix/Linux</i>	291
WEB APPLICATION SERVER	295
WEB APPLICATION SERVER INSTALLATION.....	295
<i>Changing the Default Settings</i>	295
USING WEB APPLICATION SERVER	296
<i>LyciaWeb</i>	296
<i>Web Services</i>	297
DATABASE CONNECTIVITY.....	298
CHANGING THE DEFAULT DATABASE.....	298
SETTING UP DATABASE CLIENTS.....	299
<i>Informix Client</i>	299
<i>Oracle Client</i>	301
<i>ODBC Client</i>	301
CONNECTING TO MULTIPLE DATABASES	303
<i>Database Configuration File</i>	303
<i>Source</i>	305
<i>Precedence of Parameters</i>	306
<i>Legacy Database Settings</i>	307
WORKING WITH REPOSITORIES.....	308
CVS REPOSITORY	308
<i>CVS Repositories view</i>	311
<i>History View</i>	316
SHARING PROJECTS IN LYCIA	319
<i>Sharing with the help of CVS</i>	321
MANAGING SHARED PROJECTS.....	324
<i>Checking out</i>	325
<i>Committing</i>	332
<i>Updating</i>	334
<i>Synchronize View</i>	335
WEB SERVICES	337
LYCIASTUDIO TOOLS	337
COMMAND LINE TOOLS	338



<i>The qfgl step</i>	339
<i>The qlink step</i>	340
<i>The wsmdc step</i>	340
<i>The wslink step</i>	341
<i>The wsact step</i>	342
OUTPUT DIRECTORY STRUCTURE	343
WEB SERVICES DEPLOYMENT.....	343
<i>The deployment tool</i>	344
<i>Web service client creation</i>	345
APPENDIX A: PREFERENCES	347
GENERAL PREFERENCES	349
<i>Appearance Preferences</i>	351
<i>Compare/Patch Preferences</i>	355
<i>Content Types Preferences</i>	359
<i>Editors Preferences</i>	360
<i>Keys Preferences</i>	365
<i>Perspectives Preferences</i>	367
<i>Search Preferences</i>	368
<i>Startup and Shutdown Preferences</i>	369
<i>Workspace Preferences</i>	370
4GL PREFERENCES	371
<i>Build Path Preferences</i>	372
<i>Editors Preferences</i>	372
<i>The Run/Debug Preferences</i>	374
THE INSTALL/UPDATE PREFERENCES	379
<i>Automatic Updates</i>	380
<i>Available Software Sites</i>	381
LYCIABI EDITORS CONFIGURATIONS.....	383
RUN/DEBUG CONFIGURATIONS.....	384
<i>Console Preferences</i>	385
<i>Launching Preferences</i>	386
<i>Perspectives Configurations</i>	390



<i>View Management Preferences</i>	390
TEAM PREFERENCES	391
<i>CVS Preferences</i>	392
<i>File Content Preferences</i>	396
<i>Ignored Resources Preferences</i>	396
INDEX.....	398

Introduction

Welcome to the world of Lycia, the essential tool set for 4GL, ESQL/C & New Era developers.

This 'Lycia Developers' Guide' is intended to provide you with sufficient information about creation, compilation and running 4GL applications, along with helpful instructions and examples. This guide contains articles related to the main principles of working with our development suite and LyciaStudio, our Eclipse-based development studio.

LyciaStudio is an integrated Eclipse-based development suite with a built-in graphical form editor, allowing adjustments to be made quickly and easily. With detachable tabs to rearrange your editing environment, LyciaStudio is easily customisable to meet your needs. There is also the introduction of a debugger, which allows developers to explore rich possibilities of coding and find and reduce the number of defects in a program using breakpoints and watchpoints, all easily executed from within the Studio.





About this guide

Downloading Software

Querix software, user guides and other documentation can be downloaded from the Querix Web site at: <http://www.querix.com>.

Documentation and Demonstrations

Documentation

This installation includes a set of documentation, presented in PDF format. It includes:

The 'Lycia Developers` Guide' Guide

This guide provides you the instructions on how to use Querix compiler and Studio. It will also explain the core principles while working with the command line tool, the graphical form editor and other implementations related to 4GL language.

The 'Lycia Getting Started' Guide

This guide shows you the basics of getting Lycia up and running and the wise-steps of building and developing programs and getting them working with our new Studio.

The 'Lycia Upgrade Guide'

This document contains all updates made between the latest version and the previous release.

Demonstration Applications

You can download demonstration applications for Lycia from Querix CVS repository. They can be checked out using the built-in functionality of LyciaStudio. The instructions on how to download and use the demo applications can be found in "Lycia Getting Started" guide, chapter 4.



Lycia Requirements

What is Lycia?

Lycia is a complete business-orientated development suite from Querix which is compatible with Informix-4GL and/or ESQL/C. The suite also provides the ability to present existing and future 4GL applications with GUIs whilst offering debugging facilities to control program developing. Lycia supports most common database systems including:

Informix® (v4.1 or later)

Oracle® (v8i or later)

IBM DB2™ (v8.1 or later)

Microsoft® SQL Server™ (2000 or later)

Pervasive® (8.5 or later)

MySQL (ODBC only)

PostgreSQL (ODBC only)

Other databases accessed via ODBC

Lycia is supported on most of the common OS platforms including:

Windows Server 2003/ 2008/ 7/ 8/ Windows Server 2012 (32 & 64 bit)

Linux (RedHat Fedora 11 and above, SuSE 10 and above, Ubuntu 10.04 and above)
(32 & 64 bit)

Regarding the package installation on platforms different from Linux and Windows, please contact our support team.

Lycia is more than just a standard 4GL compiler; it provides a complete development environment with the LyciaStudio, allowing the rapid deployment of 4GL applications. These applications can be presented to the user in a traditional character-based environment, similar to the Informix standard, or displayed graphically within our thin-clients. Moreover, you can add/modify an existing GUI without making any changes in your existing code, if previously it was compiled with Lycia. It is compiled to be ready for use on any of the Querix thin-client products:

LyciaDesktop – Windows GUI

LyciaWeb – Ajax/HTML5 GUI for platform independence



System Requirements

- JRE 1.7 and above
- C Compiler (only required if you include C or EsqlC files within your 4gl project or want to use web services):
 - Windows: Microsoft Visual Studio 2003, Microsoft Visual Studio .NET, Microsoft Visual C++ Toolkit 2005 and above
 - Linux: gcc



Installing Lycia

The procedure of installing Lycia on Windows and Unix platforms was designed to simplify the very process of installation. This chapter introduces key differences between the installation processes on the indicated operating systems and key issues you may encounter.

Installation for Windows

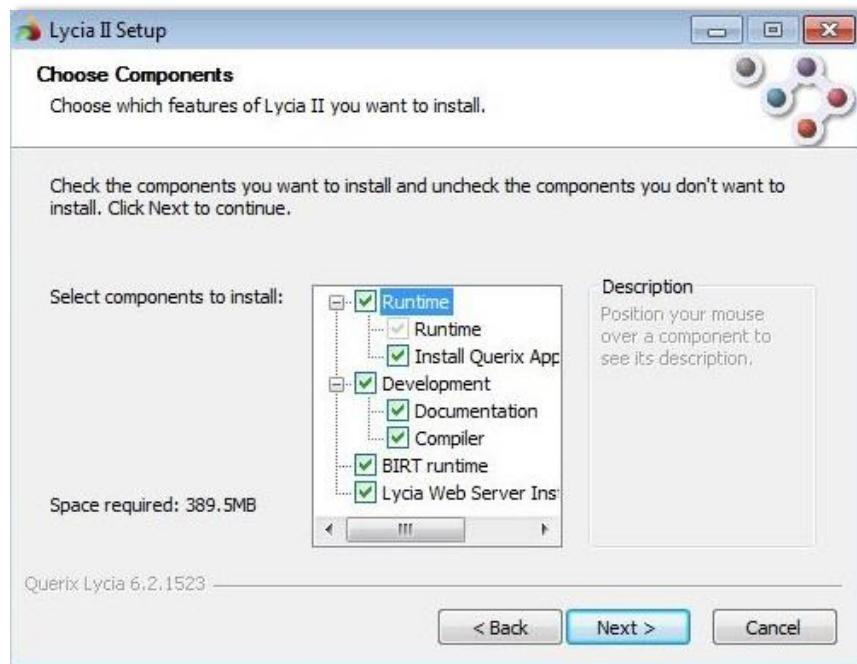
	Note: to install Lycia on Windows, the user must have administrator privileges.
--	--

Basic installation is completed by following each step after running the installation package. We will discuss any extra issues that may be encountered below.

	Note: The installation path for Lycia must not contain special symbols such as @, %, #, <, >, ", !. If it does, Lycia Studio will fail to start.
--	--

Installation modules

Lycia II Development Suite installation package includes a number of optional modules.



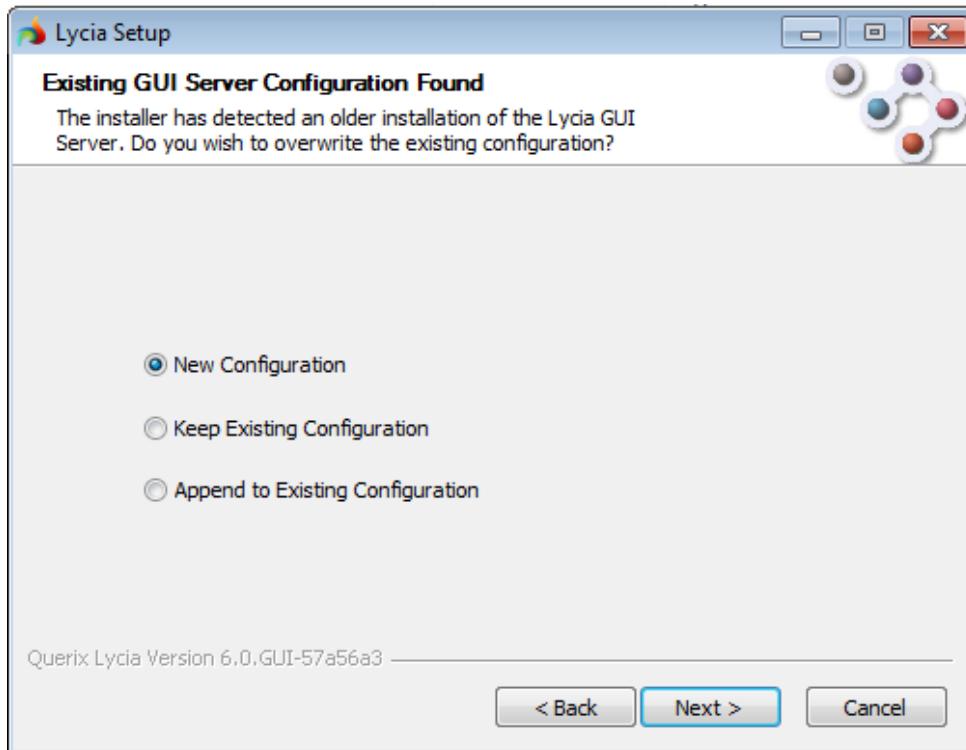
During the Lycia setup, you will see the next component modules:

- Runtime module which includes:
 - Runtime tools and libraries - checked by default as the only required module to be installed
 - Querix Application Server - automatically updates all libraries from the previous Lycia versions if there are any
- Development module:
 - Documentation
 - Compiler - it contains 4GL and Form Compiler
- BIRT Compiler - an Eclipse-based open source reporting system engine
- Lycia Web Server - this module should be checked if you want to use any of the web server related features described later in this section.



GUI Server Configuration

If you installed earlier Querix products and configured AppServer before, you will see this window:

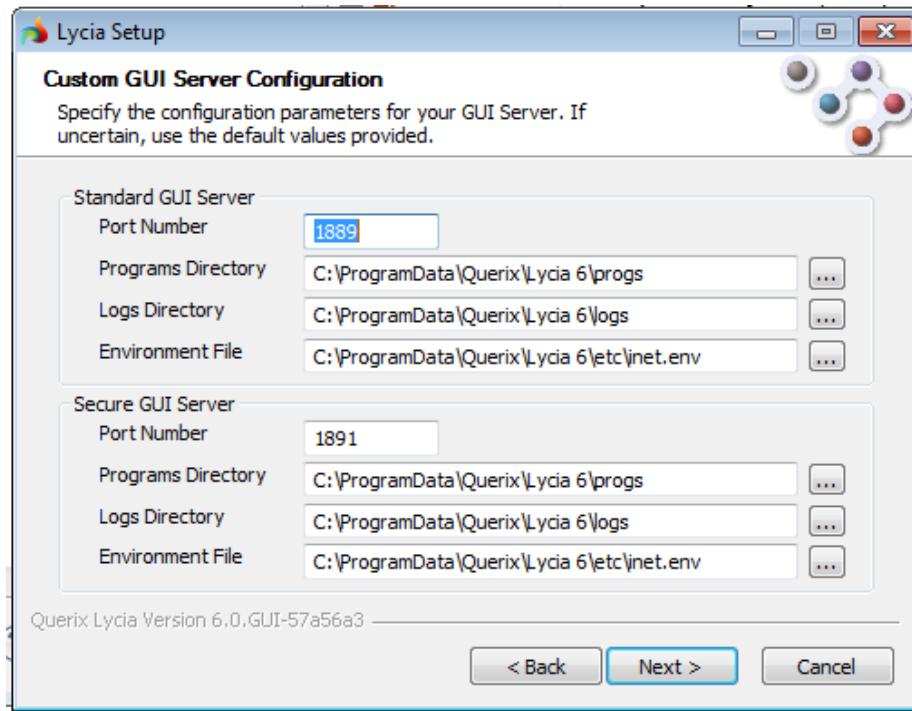


If you wish to keep your existing configuration, select the "Keep Existing Configuration" radio button and continue, else a new configuration will be created and replace any already presented configuration. If you choose "Keep Existing Configuration", the second screen of this process will not appear. There is also an "Append to Existing Configuration" option which allows previous ports to remain active whilst new ones are being added.

A circular icon with a white "i" inside, set against a purple gradient background.	Note: if you choose 'New Configuration', your previous settings will be replaced – if you previously had 'Hydra4GL' or Lycia installed, the AppServer settings will be overwritten.
--	--



If you choose to create a new configuration, then you will see a screen on which you can either keep the default values for AppServer Configuration instances (port 1889 that does not require authentication or 1891, the full authentication port) or set up your own custom instances. If you choose to customize the configuration, you will see the next screen:

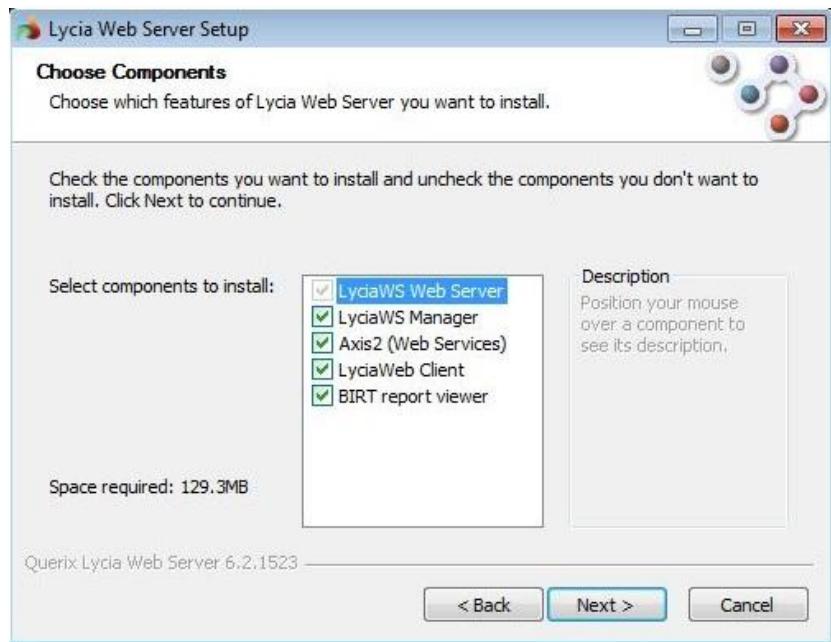


Here, you can specify the port number for both standard and secure ports, the directory programs will be deployed to and where log files will be kept and also indicate the path to the environment file that will be used. Note that the full path must be specified.

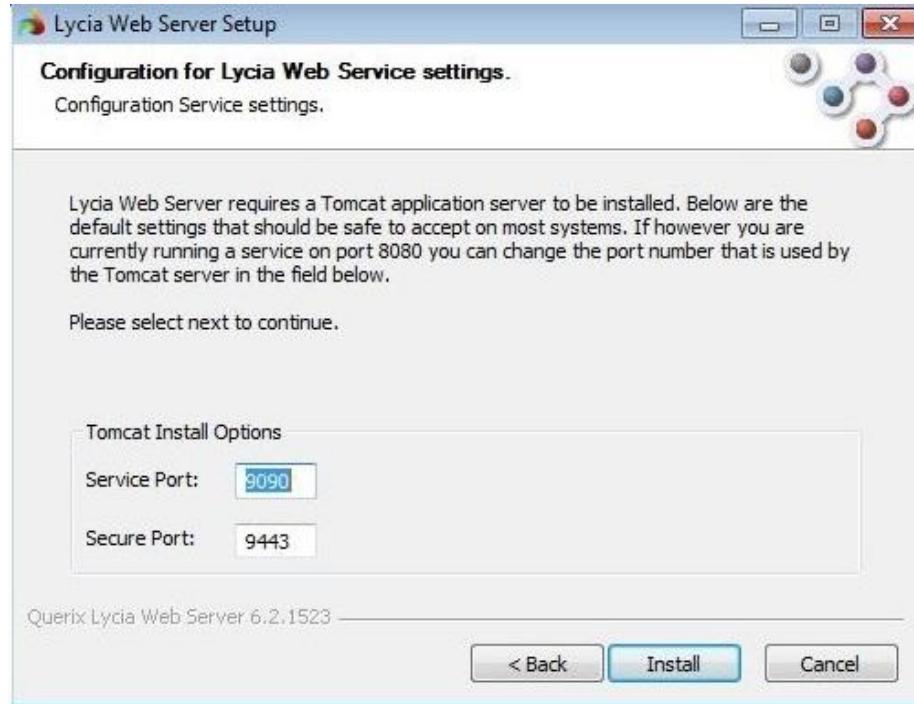
Web Server Configuration

Querix Web Server is a service based on the open source Apache Tomcat that is be used when working with different web applications and other services included to Lycia package. This is an independent installation which should be ticked off, if you want to install any of the following features (they are optional, you can check/uncheck them in the Web Server installation dialog):

- LyciaWS Web Server - an obligatory module
- Axis2 - a third party service used for deploying web services
- LyciaWeb Client - a web client used for running 4GL applications in web environment
- BIRT report viewer - an Eclipse-based plug-in used to generate and render reports



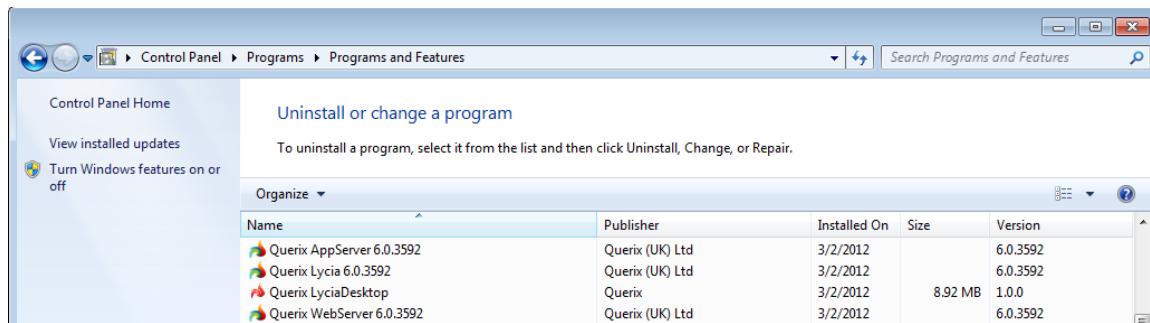
The default server ports are set up during the installation. You can change the default ports or leave them as it is specified:





Uninstalling Components

Lycia II Modules are represented as individual programs in Windows Control Panel -> Programs and Features:



Each module should be uninstalled individually. The uninstallation option in the Start menu provides the uninstalling facilities only for the Development module and does not uninstall Application Server and Web Server.

Installation for UNIX/Linux

 Note: to install Lycia on Unix/Linux, the user **must** be root.
We recommend you create a user called *querix* or *informix* and a group called *querix* or *informix*, which can act as the owner for the Querix installation

To install Lycia on UNIX, the installer should be copied into a temporary location just for the installation and the process run as root. The package should then be unzipped and extracted and then run the install script as follows:

```
gunzip Lycia-<platform>-<buildtype>-<date>.tar.gz  
tar xvf Lycia-<platform>-<buildtype>-<date>.tar  
. ./install.sh
```

After reading and agreeing to the EULA, you will be able to install Lycia into */opt/Querix* by following a number of simple steps which are explained throughout the process. Points to note:

- Step 2 allows you to change the directory where Lycia will be installed. By default, this is */opt/Querix*.



	<p>Note: The installation directory for Lycia must not contain special symbols such as @, %, #, <, >, ", !. If it does, Lycia Studio will fail to start.</p>
---	--

- Step 4 allows you to select and deselect elements of the Lycia package to be installed. You cannot disable the installation of the runtime components, but you can disable installation of the documentation, thin client support through the application server and the compiler
- Step 5 will overwrite any previous environment files if you select 'yes' – note that this cannot be undone. If none is present, then it will simply create one
- Step 6 sets the AppServer directory. If the application server is already installed in the specified location, you will be informed about it, and selecting 'yes' will replace any already established settings
- .environ needs to be run before using Lycia to ensure that the correct environment is set.

After installation, you will find in */opt/Querix* (or the location set in Step 2) a folder called Lycia, where the software has been installed to.

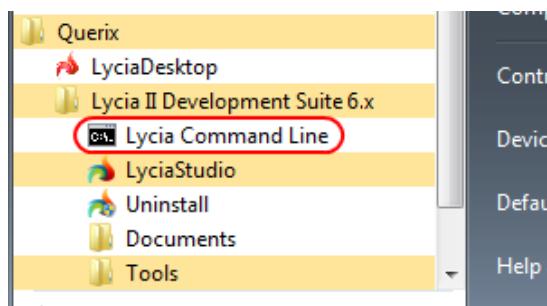
	<p>Note: to run the Studio on Linux, GTK must already be installed on the system. This must include the GTK2 bindings for Java package (for example, the libgtk-java package).</p> <p>This is because LyciaStudio is built within the the Eclipse framework that includes the SWT toolkit which uses GTK+ on Linux. Installation of this package varies across platforms, so please refer to your distribution's manuals regarding the usage of the package tools.</p> <p>Also note that if GTK is not present, the Studio may not start and will produce error messages.</p>
---	---



Licensing

To be able to use Lycia II you need a license. It can be either a full license bought from Querix or our reseller, or a trial license. The last one has a number of critical restrictions described later in this chapter. As to the full licenses, most often they differ in a number of runtime seats and an expiration date. If you are interested in getting a full license, please, contact Querix team or our reseller.

Licenses are installed and managed using the command line tool **qxlm**. This tool can be used from within Lycia Command Line environment, like other command line tools, which can be invoked from **Start -> Querix -> Lycia Development Suite 6.x -> Lycia Command Line**.



qxlm

Syntax: `qxlm [options]`

Where 'options' refer to one or more of:

<code>-?</code> [<code>--usage</code>]	Display usage information
<code>-a</code> [<code>--activate</code>] <code>arg</code>	Activate license
<code>-d</code> [<code>--deactivate</code>] <code>arg</code>	Deactivate license
<code>-t</code> [<code>--get-trial</code>] <code>arg</code>	Get trial license
<code>-l</code> [<code>--list</code>]	List of licenses installed
<code>-i</code> [<code>--hwid</code>]	Getting hardware ID

Getting Hardware ID

For manual activation you should provide Querix team with the Hardware ID of your machine where you want to activate the license. The hardware ID remains stable for the machine whose hardware configuration was not changed. Use `-i` flag without any additional parameters, e.g.:

```
qxlm -i
```



You will receive a code consisting of letters and digits. This hardware ID is sent to the license server automatically during the automatic activation and required only for the manual activation of a license.

A screenshot of a Windows Command Prompt window titled "Lycia Command Line". The window shows the command "C:\Program Files\Querix\Lycia II Development Suite 6.x\bin>qxlm -i" followed by a long activation code: "GNXPK3ZCU1ZSGYXYN2PK3JM6JP7CRMJ5". The activation code is highlighted with a red rectangular box.

Activating a License

To activate a license the option '-a' should be used. This flag should be followed by the license file together with its absolute path and file extension. It can be optionally followed by the activation code received from the license server.

Syntax: `qxlm -a <path to license> [activation code]`

For automatic activation (if the machine has Internet access), the activation code is not required. After you receive a license file from Querix team or your reseller, run this command to activate the license. E.g.:

```
qxlm -a C:\Users\user\Downloads\my_license_file.txt
```

If the license file is correct and you have Internet access, you will get the message about the successful activation and the license will become available.

You may get an error during activation, if the license was already activated from another computer. To move the license to another computer, you need first deactivate it on the machine where it was installed and then activate it on the new computer.

The activation code is required, if the computer where you want to install the license has no connection to the Internet. In this case you need to contact Querix support team and we will send you the license together with the activation code generated on the license server. You need to append it to the end of the activation command to activate the license manually:

```
qxlm -a C:\Users\user\Downloads\license346.txt
N75FKC7QGDPTC1Y22C7H28HGC62WVZMV8FQR46HA0X6KX6MVEMBFQTTJ1H49PMZGE9
```



Viewing License Information

To view all the installed licenses use `-l` flag. This option is not followed by any parameter and displays all the licenses currently installed. Besides providing you with some license details it calculates how many seats are currently in use (In Use column).

A screenshot of a Windows command-line window titled "Lycia Command Line". The window displays the output of the command `C:\>Program Files\Querix\Lycia II Development Suite 6.x\Lycia\bin>qxlm -l`. The output shows license details and a usage table:

Service Name	In Use	Total
Lycia BI Runtime	0	2
Lycia BI Server	0	2
Lycia Batch	0	2
Lycia Compiler	0	1
Lycia DB Firebird	0	2
Lycia DB IBM DB2	0	2
Lycia DB Informix	0	2
Lycia DB Intersystems Cache	0	2
Lycia DB MySQL	0	2
Lycia DB ODBC	0	2
Lycia DB Oracle	0	2
Lycia DB Pervasive	0	2
Lycia DB PostgreSQL	0	2
Lycia DB SQL Server	0	2
Lycia Runtime	0	2
LyciaAjax	0	2
LyciaDesktop	0	2
LyciaText	0	2

License Details

Each license has a number of attributes. They are as follows:

- License Serial Code is a unique identifier of the license.
- License ID stands for the name of the license (in contrast to License Serial Code it does not contain any critical information).
- Status is the current status of the license. It can be set to "Active" or "Inactive" that means that license can or cannot be used correspondingly.
- Expiry Date is the date till which the license is valid. After the specified date the license cannot be used even though it may still have active status.
- NodeLocked if set to "True", the applications compiled with this license can be run only from the application server located on the same machine and cannot be transferred to



other application servers. If set to "False", applications can be copied and transferred without restrictions. The first value is, as a rule, set for the trial license.

- Processors Number stands for the maximum number of processors which can have the machine with this license installed. If set to "Unlimited", the processor number is ignored and the license can be installed to any machine irrespective of the number of processors it has.

If the license contains a numeric value in this field, it means that an unlimited number of runtimes is allowed on the machine where the license was activated, provided that the value in this field is bigger or equal to the actual number of the machine processors. If the Processor Number is smaller than the actual number of the processors, no runtimes can be used. E.g. if the license has "2" for the processor number, it can provide the runtimes for the computers with one and two processors, but not for a machine with 3 or more processors.

- Service Name column lists the services granted by the license. Each process, such as runtime, database connection, compilation, has its corresponding service.
- In Use column shows how many seats of each service are currently used.
- Total column shows how many seats of each service the license grants in total.

Deactivating a License

To deactivate an active license from within Lycia Command Line environment, the option '-d' should be used.

Syntax: qxlm -d <license serial> [deactivation code]

The option declaration should be followed by the license serial number which can be found in the license file, e.g.:

<LicenseSerial>7B62DNHSHRX5AM</LicenseSerial>

It can also be looked up from the list of licenses displayed using the `qxlm -l` command:

```
C:\Program Files\Querix\Lycia II Development Suite 6.x\Lycia\bin>qxlm -l
License Serial Code: U1WJ3QN83CEBN8
LicenseId: Evaluation License 1.5
Status: Active
Expiry Date: 2014-01-03
NodeLocked: True
Processors Number: Unlimited
Service Name      In Use  Total
Lycia BI Runtime    0      2
Lycia BI Server     0      2
Lycia Batch         0      2
Lycia Compiler      0      1
Lycia DB Firebird   0      2
Lycia DB IBM DB2    0      2
```



If your machine is connected to the Internet, you should omit the deactivation code. Here is an example of the automatic deactivation command:

```
qxlm -a 7B62DNHSHRX5AM
```

If the machine is not connected to the Internet, you need contact Querix team and ask to deactivate the license for you on the license server. After that Querix team will send you an e-mail with the deactivation code. You should append the deactivation code to the command for manual deactivation, e.g.:

```
qxlm -a 7B62DNHSHRX5AM  
N75FKC7QGDPTC1Y22C7H28HGC62WVZMV8FQR46HA0X6KX6MVEMBFQTTJ1H49PMZGE9
```

Getting a Trial License

A trial license is a license which is available to anyone who downloaded and installed Lycia for free. To get the trial license you need not contact Querix, if the machine where you want to install it has Internet connection. If the machine is not connected to the Internet, you request the license from Querix team and activate it using -a flag as described above.

To get a trial license automatically (for the machine where Internet connection is available) you should use the '-t' option.

```
Syntax: qxlm -t <username>
```

As it is seen from the syntax the -t flag should be followed by your login used on Querix web site, e.g.:

```
qxlm -t my_querix_login@mail.com
```

Then you will be asked to enter the password you use to access the web site. If the login is correct, the trial license will be downloaded, activated and added to the list of licenses.

```
C:\apps\Lycia Development Suite 6.x\Lycia\bin>qxlm -t my_login@mail.com  
my_login@mail.com's password:  
Trial License added successfully
```

If instead of success an error message appears, make sure that your login and password were put correctly. The possible causes of an error (besides incorrect login/password) may also be the following:

- trying to get the trial license for the second time for the same machine;
- the server is not available;
- the local file system is not available.



If you cannot reach success in getting a trial license on your own, please, contact Querix support team to handle a problem.

Trial License Restrictions

A trial license has a number of restrictions:

- It can be claimed only once for each computer.
- It provides 2 seats for each service which means that you are granted only 2 runtime seats. An exception is the Lycia Compiler service that can be only run on one machine (one runtime seat is provided for this service).
- It is valid for 6 weeks (42 days) starting from the day it was claimed.
- It is node-locked, which means applications compiled with this license can be run only from the application server located on the same machine and cannot be transferred to other application servers.

 A circular icon with a white 'i' inside, set against a light purple background.	<p>Note: Only one trial license can be claimed per computer. If you try to claim a trial license and get the following message: "ERROR: Demo license has been already acquired for this machine", this means that a trial license for the machine with such hardware ID was already claimed. If you did not claim license previously and think this message is shown by mistake, please, contact Querix support team.</p>
---	---



Overview of LyciaStudio

LyciaStudio is a simple-to-use, intelligible and organised development environment for both the development of new 4GL projects and maintenance of already existed 4GL projects. LyciaStudio runs from Java environment, meaning it can be run on any platform that supports Java.

LyciaStudio uses “workspaces” to maintain projects. A workspace is a folder where all projects are maintained and can contain a variety of projects, projects in their turn can contain multiple programs. It is worth noting that files from older projects must be imported using the Import methods within the Studio and not just copied into relevant folders.

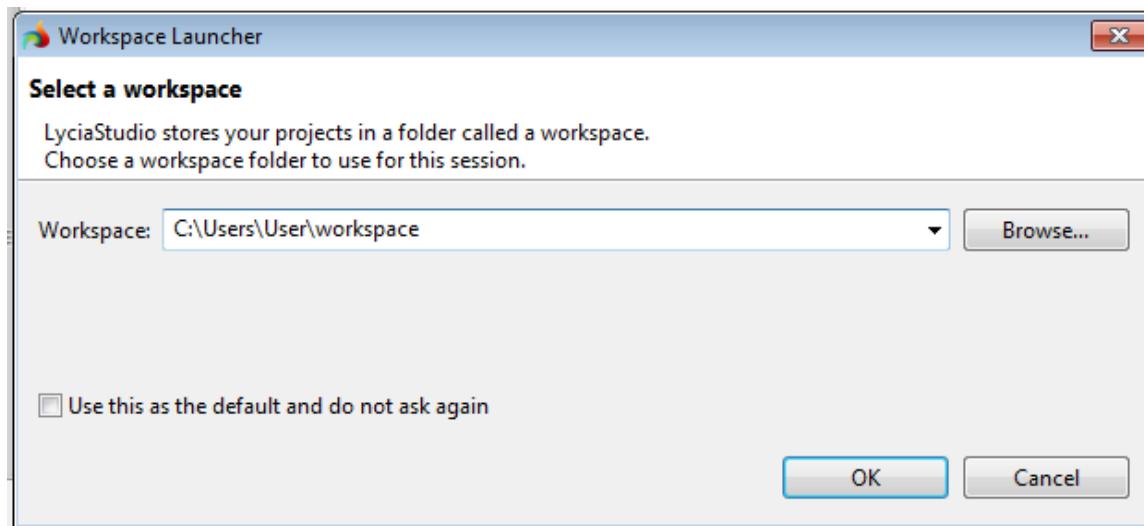
Running the Studio

Once Lycia is installed, on Windows it can be accessed via a link to the Studio under **Start Menu -> Programs -> Querix -> Lycia Development Suite 6.0**. From here, the user can either run “LyciaStudio” or “Lycia Command Line Environment”. LyciaStudio will load the Studio interface, whilst the Command Line Environment will open up a command prompt.

On UNIX, simply navigate to the folder where Lycia was installed to (\$LYCIA_DIR), and run the Studio by executing lyciaide/lycia-ide.

On Mac OSX, go to **Applications -> Querix -> LyciaStudio**.

After running the LyciaStudio program, you will be presented with the following window:



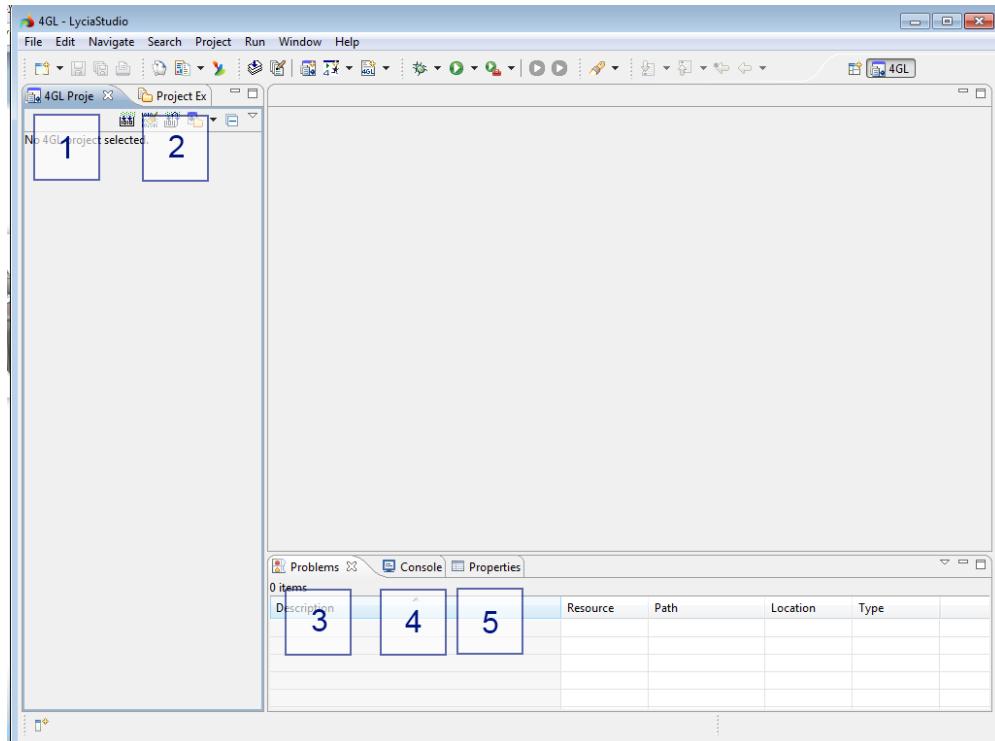
The default path here is where Lycia recommends you to create your ‘workspace’, a place to store the projects and source code a developer will work on. Developers can have more than one



workspace to fit project organisation goals, or all projects can be stored within just one workspace. The number of workspaces that can be used or created and switched between simultaneously is not limited.

Layout of LyciaStudio

Click OK on the previous dialog screen, and the Studio will load the specified workspace, which will look like the following on the initial run:



- 1) 4GL Project – allows the developer to see the contents of each of their programs and libraries
- 2) Project Explorer – allows the developer to manipulate with the projects which are presented in the form of a tree
- 3) Problems – any errors or warnings appearing during compilation
- 4) Console – displays information about actions you perform in the Studio and display expressions
- 5) Properties – lists any properties of a selected item within the Studio. It is mostly used for editing graphical form files.

By default, the **4GL Project** view tab obscures the **Project Explorer** view tab. The **Project Explorer** allows the user to see where files are within each directory and the structure of such



directories, without showing any indication of which file belongs to which program. The 4GL Project View tab shows all files grouped together by their program and requirements.

When first installed, the layout is as above. However, the panels are dockable and interchangeable so a developer can move panels into their own preferred layout (called a perspective). When the developer double clicks on an open panel, such as the currently worked on 4GL file, the panel will be displayed full screen. Another set of double clicks on the same panel will restore the view to its original size.

Below, we will describe the panels, the menus and the toolbars available to developers in the LyciaStudio.

Panels

Panels refer to the various panels available within the LyciaStudio. As you can see from above, there are 6 panels within a default installation – the 5 listed previously and the Main Body panel. All of these panels are “dockable”, meaning the developer can arrange these panels into any layout they desire, or simply remove any they find redundant.

Project Explorer

Initially the Project Explorer panel opens on the left hand side of the screen, overlaid by the 4GL Project panel. It allows the developer to see both the output and source files for a program as well as the executable produced. Many projects can appear here.

4GL Project

Initially the 4GL Project panel is on the left hand side of the screen, overlaying the Project Explorer. The 4GL Project panel allows the developer to see the contents of each of their programs in a tree structure. Many programs can appear here, and will depend upon which project is currently selected in the Project Explorer. Simple project commands such as building and compiling can be executed from this panel. Libraries can also be added here.

Problems

Initially, this is one of the 3 panels along the bottom of the screen, being the first. Any errors found during compilation will appear here. It will inform the developer of the problem found, and where the problem has occurred and any variables involved.

Console

Initially, this is one of the 3 panels along the bottom of the screen, being the second. Any important information messages in relation to the build system are displayed here. Moreover, all the display expressions (indicated after the DISPLAY statement within an application) are displayed to it.

Properties

Initially, this is one of the 3 panels along the bottom of the screen, being the third. This lists any properties that are present in the current program. This panel comes into its own with the



graphical form editor active, as every attribute on the form can be selected and details about it can be seen and edited from here. A developer can change the position, height and width of a field, set colour properties, widget type of a field, whether it is required and the appearance of a field are amongst the settings that can be changed here, and which will be discussed in more detail in the "Using the Form Editor" chapter of this document.

Editor

The main body of the Studio (the Editor) is a set of tabs within a panel in which the developer edits their files. A developer can have open as many files as needed to work from and the panel will adjust to allow access to all of them from a dropdown menu.

Menus

Menus are found in the majority of applications, and this section we will discuss elements that are not standard menu features for LyciaStudio.

File

A standard menu containing New, Open, Import and Save file options amongst others. This menu also contains a Most Recently Used list at the bottom of 4 previous files opened for convenience. Properties about projects can also be accessed from here, as can the current workspace be changed by switching workspaces.

Edit

Another standard menu containing Cut, Copy and Paste exist amongst other Studio-specific tools. Content Assist provides developers with a tool which contains all available functions and an encoding specification tool, where the developer can change which encoding they are using. Delete will remove a file from the file system, after confirmation from the user.

Navigate

Navigate has navigation tools and allows the developers to navigate through the source code.

Search

Search is an Eclipse function which will search or replace within individual and groups of files for specific text, throughout an entire workspace. When a search is completed, a new tab appears alongside the Console tab with the search results inside it. The search functionality will be explained fully in a later section. A right-hand click (or Control-F) within the editor however will start a "simple search", which is a search within just that one open file, similar to a "Find" option.

Project

Project contains all of the features available in relation to a project, such as building all open programs or selected programs, setting a default option of autobuilding a program, or changing the properties of a program, such as source folder locations, ports to run programs against and deployment paths. Properties can also be edited from this menu. The properties menu is explained later in this chapter.



Run

Run encapsulates the breakpoints, running and debugging options under one menu. The developer can toggle on and off any breakpoints, as well as define run and debug configurations and launch a debugger for their application from here. Running and debugging will be explained fully in a later section of this document.

Window

Window covers most of the customisation of a user's perspective, such as adding favourite toolbar buttons to a particular perspective. A perspective is the layout that each developer has by default for the arrangement of their panels within the Studio. Also customisation of the Studio in general is available, where such options as auto-building a project, change start-up settings and which database driver to compile against by default. At any stage, the perspective created by the developer can be reset to a default set-up so all layouts return to their original positions. Preferences are accessed from this menu to, and they have their own section later on in this document, listing what each option's functionality is. The Preferences option is explained later in this document in more detail.

Help

The help menu will provide access to any help topics, and also the build version of your LyciaStudio. Here, the developer will be able to look up the syntax of functions for 4GL and SQL. Also Key Assist can be activated, which is a list of all available shortcut key combinations for functions. A list of all plug-ins and any software updates can be obtained through the Studio itself from the Querix servers.

Project Properties Menu

The Project preferences menu can be found under **Project->Properties** and is where project specific variables are adjusted.

Resource

States folders and paths, as well as text file encodings and line delimiters:

- Path
 - Where the project is in relation to the Studio installation
- Type
 - The type of project currently open
- Location
 - Location of the current working workspace
- Text File Encoding



- Can specify which encoding to use (default from container or specified by user)
- Text File Line Delimiter
 - Specify which format to use and which platform standard (Windows, Unix, Mac OS)

4GL Project

4GL related preferences:

- Source and Output Folders
 - Specify source folders and output folders for the project or use just the project alone.
- Database
 - The user can choose between using the workspace default database, or specifying one for a particular project. This means that one workspace can have many projects, all with different database connections

Build Configuration

The list of the build configurations available for the selected project

GUI Server

The GUI Server settings for this project:

- Default GUI Server
 - Use either the workspace default port or set one up for the project
- Deployment Path
 - Choose which folder to deploy a project to (can be the root or to a subdirectory)

Project References

References to other projects in the same workspace

Run/Debug Settings

Managing launch configurations for this project



Main Toolbar

The main toolbar of LyciaStudio looks like the following:



The main toolbar consists of the toolbar buttons and a tab with the buttons that switch perspectives.

We will now explain what each element of the toolbar's functionality below:

- The 'New' wizard (Alt-Shift-N) – the same as clicking on File -> New
- Save (Ctrl-S) – a shortcut to the 'save' functionality
- Save All... (Ctrl+Shift+S) - a shortcut which saves all the files open in the editor area which were changed
- Print (Ctrl-P) – a shortcut to the 'print' functionality
- Build All (Ctrl-B) – this builds all programs in an open project
- Documents Wizard - Create a new LyciaBI document
- New LyciaBI project
- Compile the selected source file – compiles the file open in the editor
- Build Automatically – enables automatic building
- Create a new 4GL Project
- Create a new 4GL Program – with options for files within this such as 4GL library, C Library and Web Service
- Create a new source file – can be 4GL, C, form file, message file etc
- Debug As (F11) – activates the debugger, which will be explained in its own section



 - Run As (Ctrl-F11) – ‘run as’ settings can be changed here. Has a MRU top 5 list to choose from, explained more in its own section

 - Run with LyciaDesktop – runs the current program in Lycia Desktop Client

 - Run with LyciaWeb – runs the current program in LyciaWeb Client

 - Next Annotation (Ctrl+.+) - takes you to the next annotation. The annotations can be selected from the drop-down menu of this button, these can be: breakpoints, bookmarks, tasks, errors, warnings, etc.

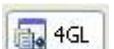
 - Previous Annotation (Ctrl+,-) - takes you to the previous annotation. The annotations can be selected from the drop-down menu of this button, these can be: breakpoints, bookmarks, tasks, errors, warnings, etc.

 - Last edit location – takes you to the last place an edit was made

 - Back to <file name> (Alt+Left) – opens and brings forward a previously viewed file

 - Forward to <file name> (Alt+Right) – opens and brings forward the next file in the list of recently viewed files. It is active only if you have previously pressed the Back to <file name> button

 - Open Perspective – opens a list of available perspectives

 - The button which brings forward the 4GL perspective when there is more than one perspective opened. The buttons of other open perspectives can be located next to it.



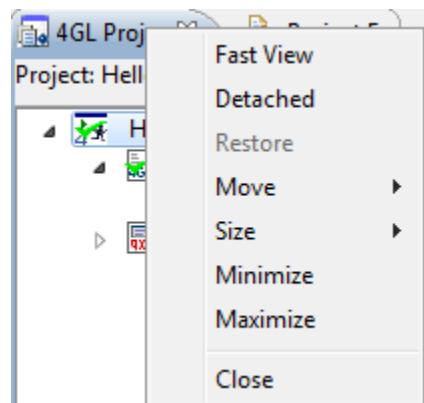
Perspectives and Views

Each LyciaStudio runtime window contains one or more perspectives. A perspective is a combination of views and options. Different perspectives share the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task. For example, the 4GL perspective combines views that are typically used while editing 4GL source files, and the Debug perspective contains the views that are used for debugging 4GL programs. Perspectives control what appears in certain menus and toolbars. They can be easily switched, opened, closed, or customized.

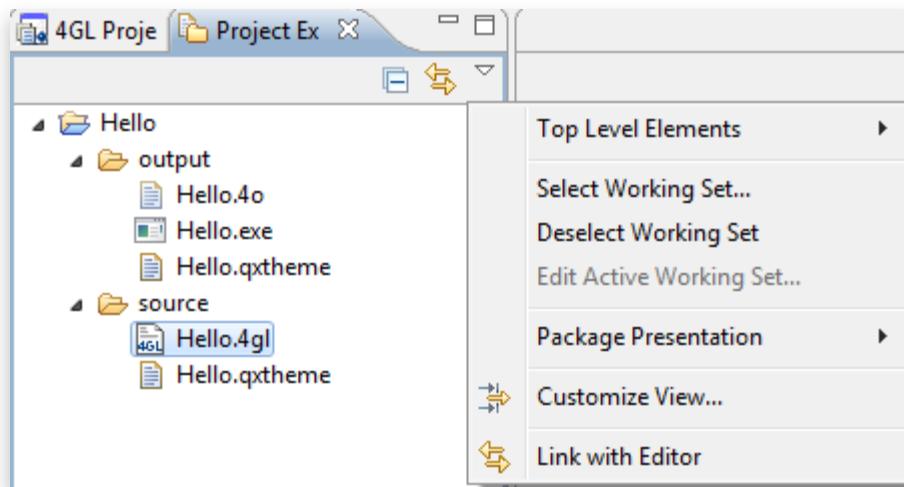
A view is a panel that has its specific functions: it supports editors or provides alternative presentations as well as ways to navigate the information in your Studio. For example, the 4GL Project View displays the currently selected project and its source files. Some views have their own toolbars and menus. A view can appear by itself or it can be stacked together with other views.

Managing Views

Typically a view has two menus. The first, which is accessed by right-clicking on the view's title bar, contains general manipulation options.



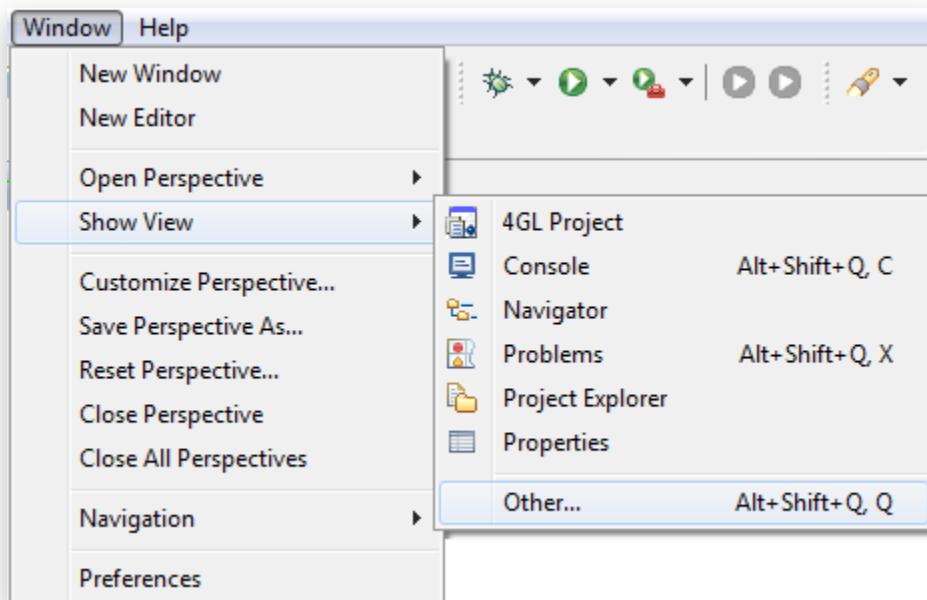
The second menu, called the “pull-down menu”, is accessed by clicking the down arrow (▼). The view pull-down menu typically contains operations that apply to the entire contents of the view, but not to a specific item shown in the view. Operations for sorting and filtering are commonly found in the pull-down menu.



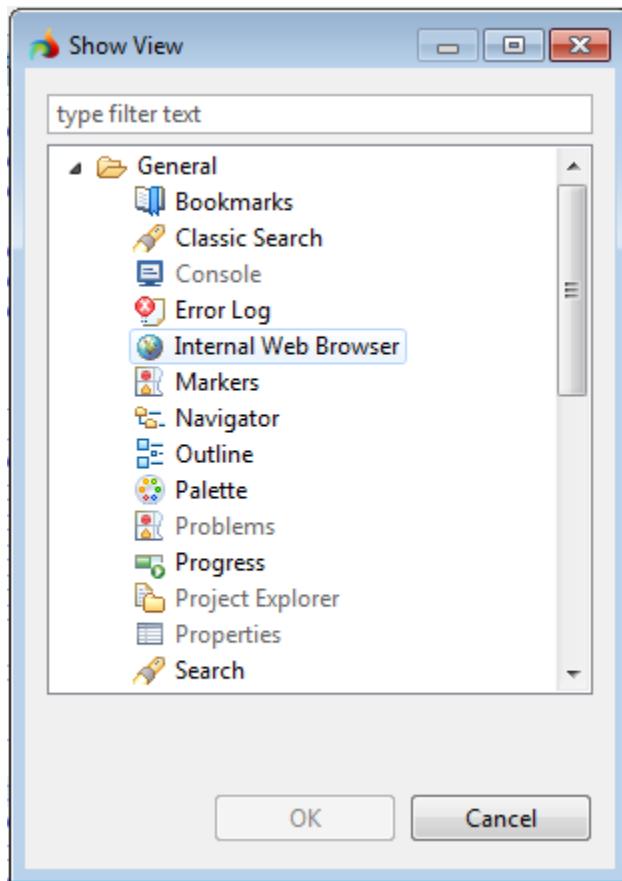
To maximize a view or editor, double-click its title bar. To restore the size of the view double-click its title bar one more time.

Opening Views

The views which are associated with a certain perspective are opened automatically when the perspective is opened. Some views (e.g Properties view, Search view) are opened when LyciaStudio undertakes certain actions. To open any view manually, select the **Window -> Show View** main menu option:



There will be several views available. To see all the views, click the **Other...** option. You will see the list of views as follows:



Select a view and press the **OK** button to open it.

Rearranging Views

The position of the 4GL Project view or any other view in the Studio window can be changed.

1. Click on the title bar of the 4GL Project view and drag the view across the Studio window.
2. While still dragging the view around the Studio window, note that various drop cursors and empty frames appear. These drop cursors can look like:
 - - Stack: If the mouse button is released when a stack cursor is displayed, the view will appear as a tab in the same pane as the view underneath the cursor.
 - - Window: If the mouse button is released when a window cursor is displayed, the view will be placed in a separate detached window. This option cannot be used with the editors, it can be applied only to views.
 - The rectangular highlight indicates where the view will dock in relation to the view or editor area underneath the cursor when the mouse button is released.



Creating Fast Views

Fast views are hidden views, which can be quickly made visible. There are two ways to create a fast view:

- Using drag and drop.
- Using a menu operation available from the view context menu.

To create a fast view using drag and drop, do the following.

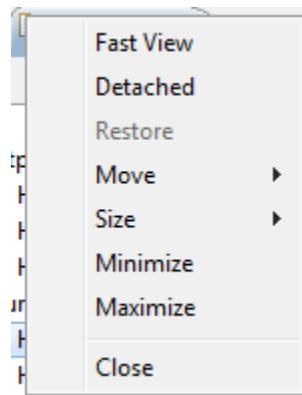
1. In the 4GL Project view, click on the title bar and drag it to the shortcut bar at the bottom left corner of the window.
2. Once over the shortcut bar, the cursor will change to a "fast view" cursor. Release the mouse button to drop the view onto the shortcut bar:



3. The shortcut bar now includes a button for the 4GL Project fast view:



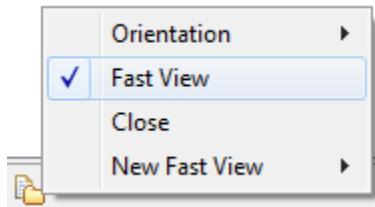
To create a fast view using the second method, start by popping up the context menu over the view's tab. From this menu, select the **Fast View** option.



To display a fast view, click on the icon on the shortcut bar. The view will be hidden, if you click on any other view within the Studio.



To restore the position of the view, deselect the **Fast View** option:



Editors

To open a source code editor, double-click a source file in the 4GL Project view. Depending on the file type, the appropriate editor will be displayed in the editor area. For example, if a .4gl file is being edited, the 4GL Source File Editor is displayed in the editor area, a .4fm file is opened in the 4GL Graphical Form Editor. The name of the file appears in the title bar of the editor.

```
*Hello.4gl X
MAIN
DISPLAY "Hello World!" AT 2,2
CALL fgl_getkey()
END MAIN
```

A screenshot of the 4GL Source File Editor window. The title bar says '*Hello.4gl'. The editor area contains the following 4GL code:

```
MAIN
DISPLAY "Hello World!" AT 2,2
CALL fgl_getkey()
END MAIN
```

An asterisk (*) appearing at the left side of the title bar indicates that the open file has unsaved changes. If an attempt is made to close the editor or exit the Studio with unsaved changes, a prompt to save the editor's changes will appear.

The editors can be stacked in the editor area and individual editors can be activated by clicking on the tab of the editor. Editors can also be tiled side-by-side in the editor area in the same way as other views can be moved around the Studio, so their content can be viewed simultaneously.

If a resource does not have an associated editor, the Studio will attempt to launch an external editor registered with the platform. These external editors are not tightly integrated with LyciaStudio and are not embedded in the editor area.

Editors can be cycled through using the back and forward arrow buttons () on the main toolbar. These arrows move through the last mouse selection points and permit moving through several points in a file before moving to another one. Additionally, editors can be cycled by using the **Ctrl+F6** key combination. **Ctrl+F6** pops up a list of currently open editors. By default, the

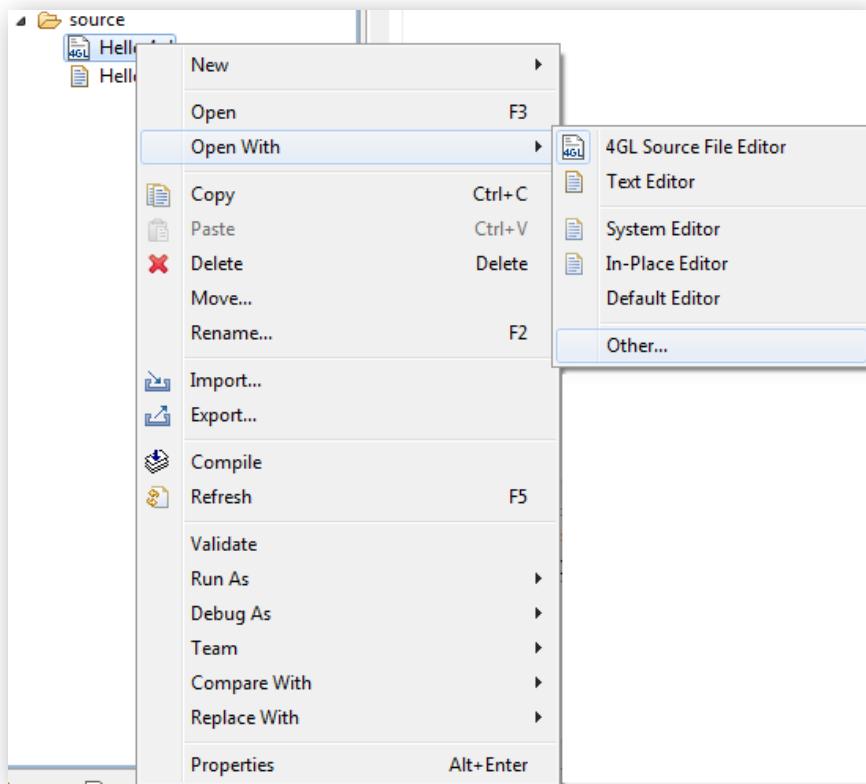


list contains the editor used before the current one, allowing you to go back to the previous editor.

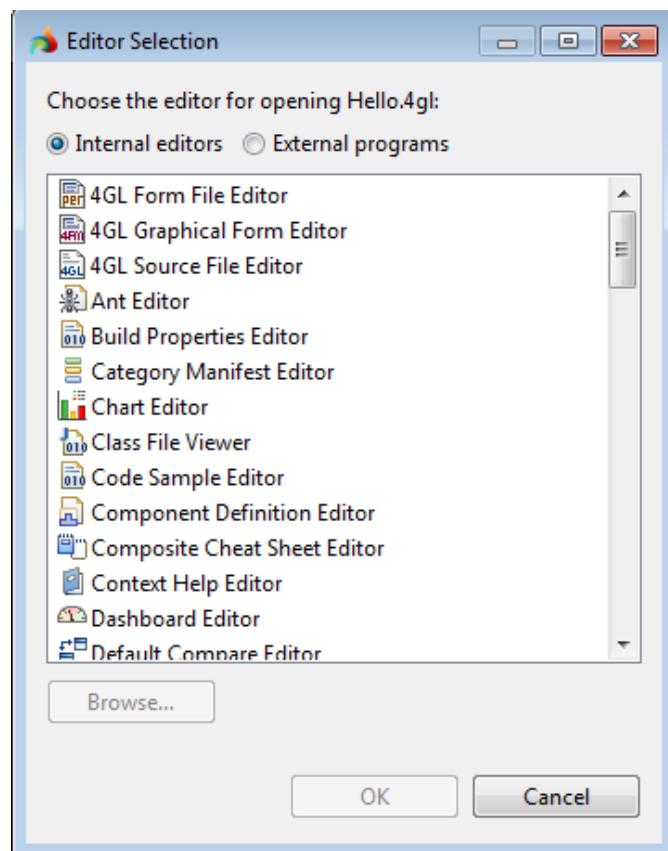
On Windows™, if the associated editor is an external editor, the Studio may attempt to launch the editor in-place as an OLE document editor. For example, editing a .doc file will cause Microsoft Word to be opened in-place within the Studio if Microsoft Word is installed on the system. If Microsoft Word has not been installed, Word Pad would open instead.

Choosing an Editor

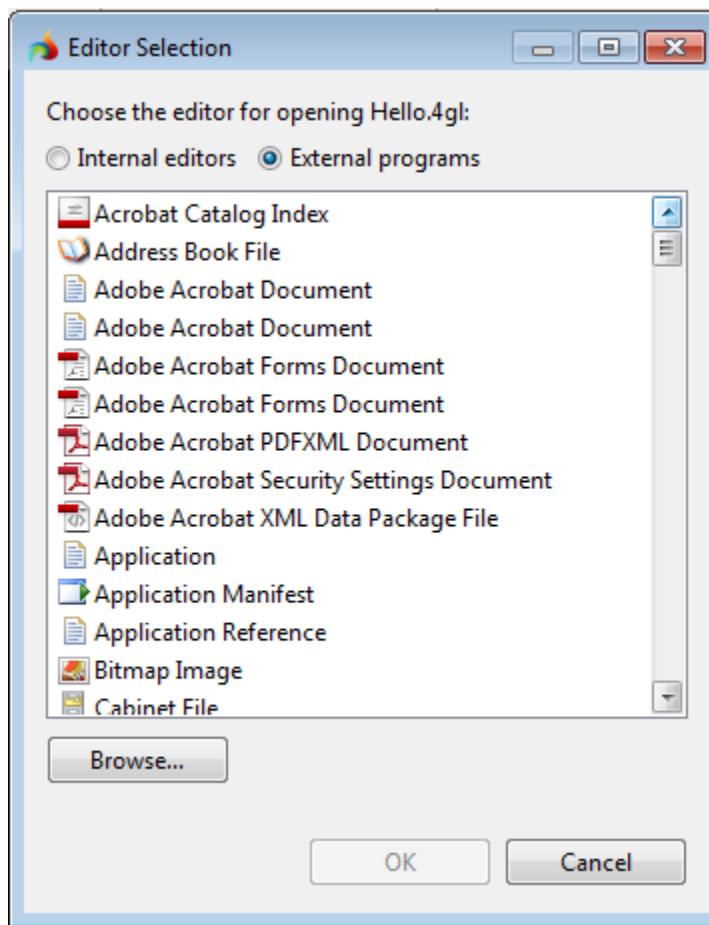
A source file is typically opened with the default editor. LyciaStudio supports both external and internal editors. If you want to choose an editor different from the default one, click the right mouse button on the file in the Project Explorer view and choose the **Open With** option.



If you choose the **Other...** option, you will be presented with the list of internal editors:



If you want to use an external editor, select the "External programs" option to see the list of editors available on your system:



Managing Perspectives

LyciaStudio contains a number of built-in perspectives. The 4GL perspective is the default perspective. Some perspectives are opened automatically when LyciaStudio performs certain actions (e.g. Debug perspective is opened when a program is run in debug mode).

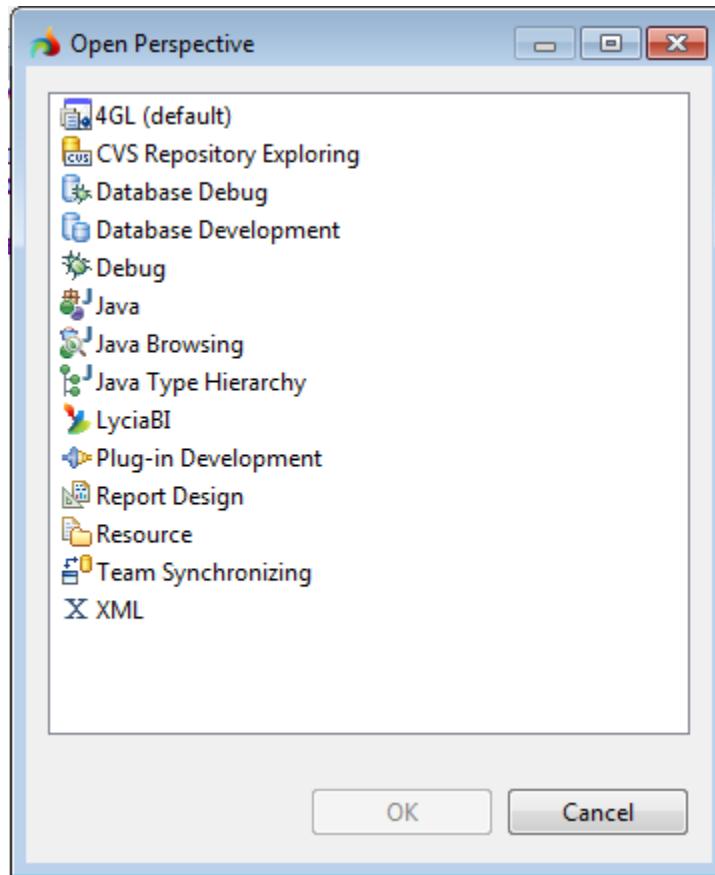
To open a perspective manually, follow these steps:

1. Press the Open Perspective button on the shortcut bar or select the **Window -> Open Perspective** main menu option.
2. A menu appears showing the available perspectives. Choose the **Other** option from the list:



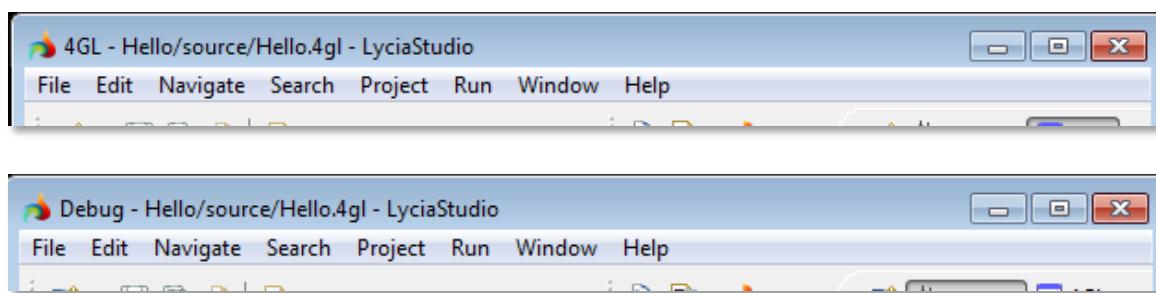


3. In the Open Perspective dialog, choose any of the available perspectives and press the **OK** button:



The title of the window will indicate which perspective is currently in use. The shortcut bar will now contain two perspectives, the original 4GL perspective and the one you have opened.

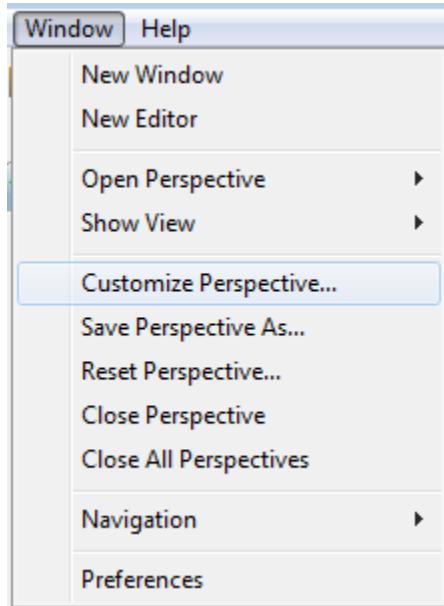
Below are given screenshots of LyciaStudio window titlebar with 4GL and Debug perspectives active:





Customizing Perspectives

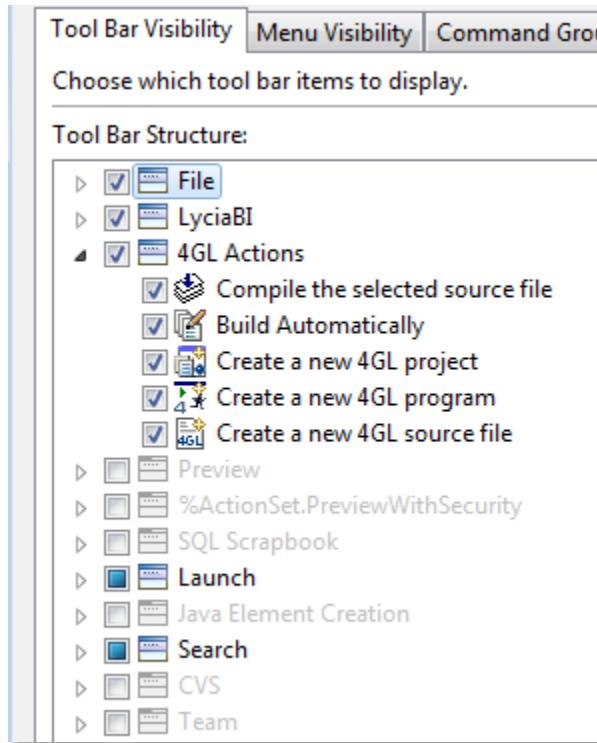
You can customize a perspective by adding, removing and rearranging views as described above in the “Managing Views” section. To modify the toolbar and the menu of a perspective, select the **Window ->Customize Perspective...** main menu option:



The Customize Perspectives dialog consists of four tabs.



The “Toolbar Visibility” tab is used to specify which buttons should be visible on the main toolbar of the Studio:



You can select the “Filter by command group” option to switch to other display mode, where the command groups are placed in the left pane and their contents in the right pane:



Tool Bar Visibility Menu Visibility Command Groups Availability Shortcuts

Choose which tool bar items to display.

Command Groups:

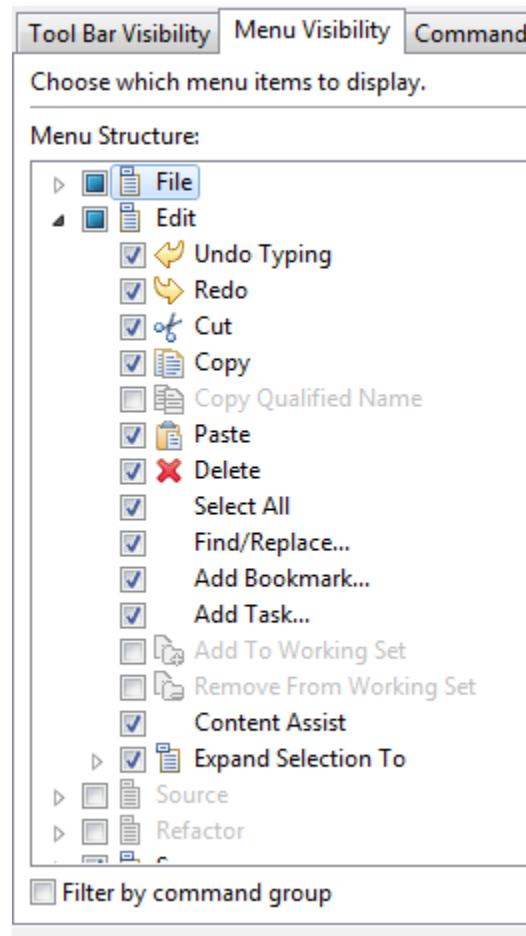
- %ActionSet.PreviewWithSecurity
- 4GL Actions
- 4GL Launching Actions
- Annotation Navigation
- Ant Editor Presentation
- CVS
- Editor Navigation
- Editor Presentation
- External Tools
- Java Editor Presentation
- Java Element Creation
- Java Navigation
- Launch
- LyciaBI
- Preview
- Profile
- Search
- SQL Scrapbook

Tool Bar Structure:

- ▲ Launch
- Run with LyciaAjax
- Run with LyciaDesktop

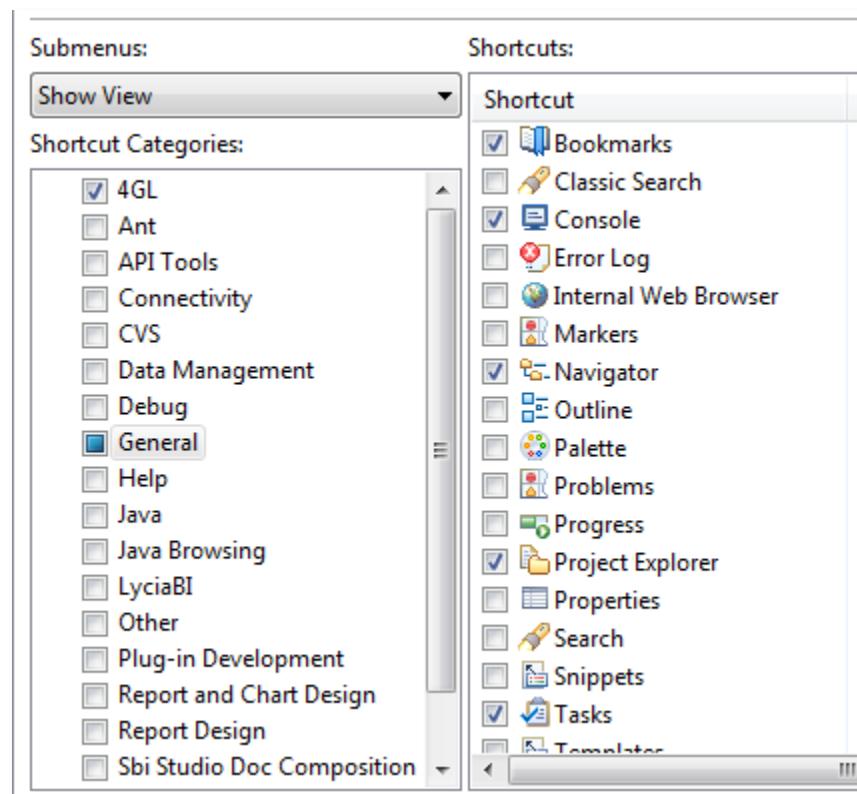
Filter by command group

The “Menu Visibility” tab is used to specify which options should be visible on the main menu of the Studio. It can also be filtered by groups:

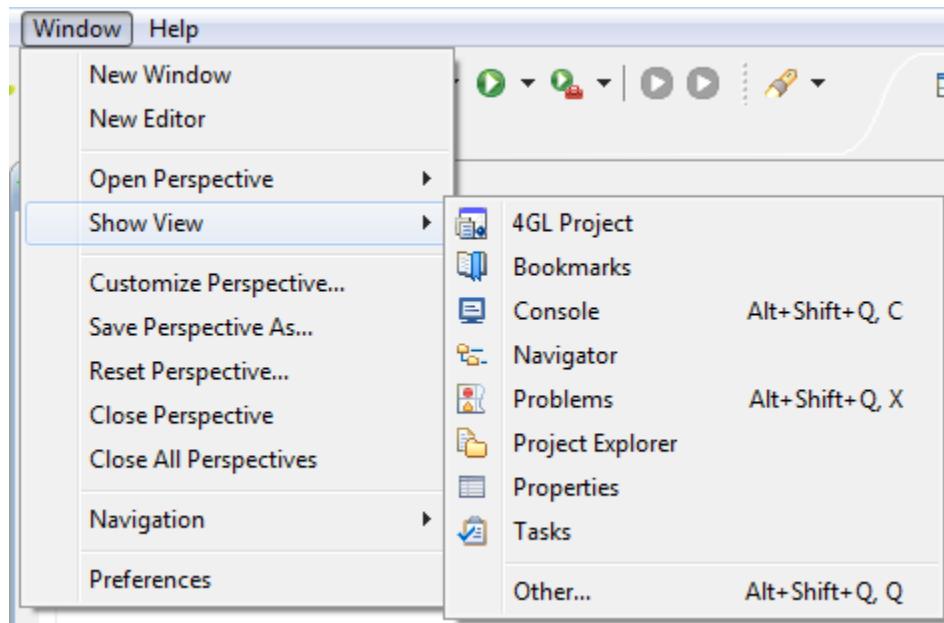


The "Shortcuts" tab allows you to modify the **Show View**, **New**, and **Open Perspective** submenus of the main menu. Select the submenu from the combo list and select the objects you want to be displayed in this menu. You can also deselect the items that you do not want to be displayed in the submenu.

E.g. select "Bookmarks" and "Tasks" in the **General** option of the **Show View** submenu as shown below and press **OK**:



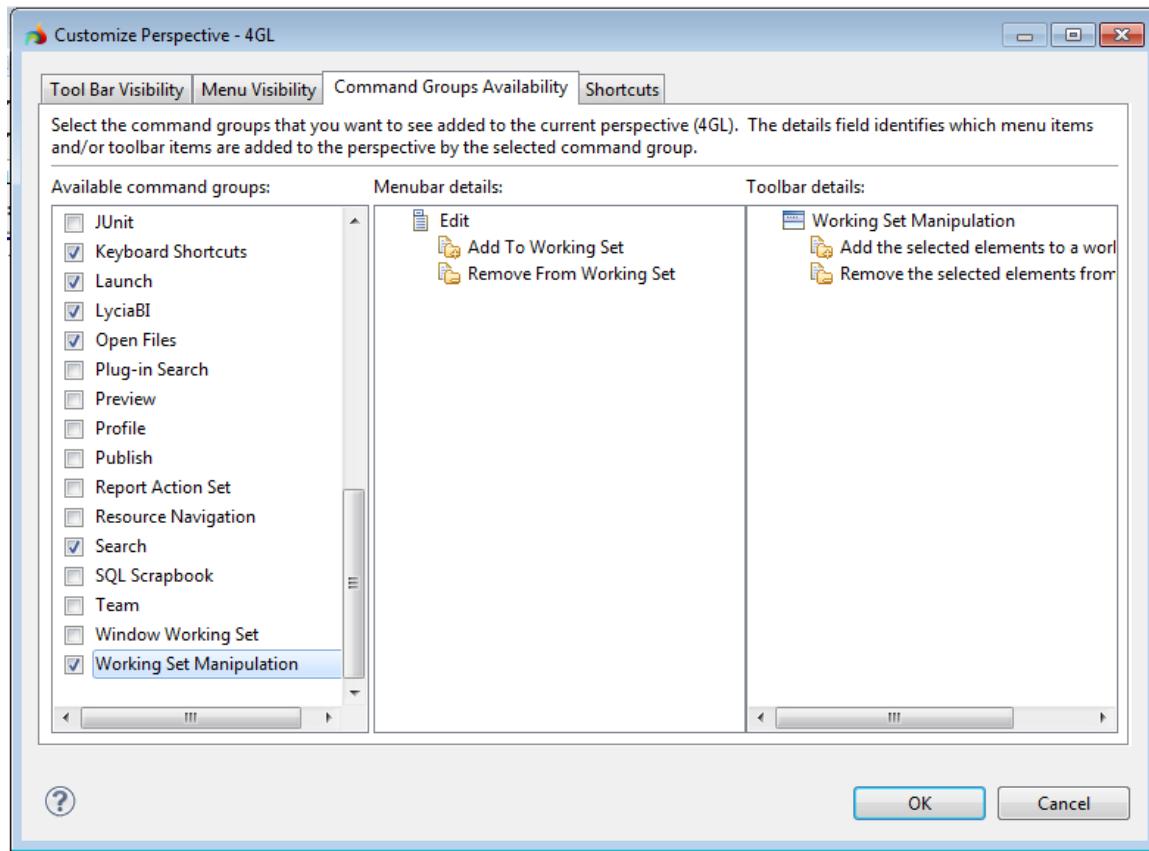
There will appear in the list of available views in the **Window ->Show View** main menu option:



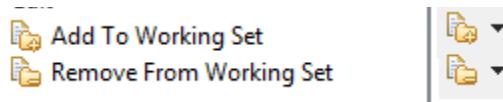


The “Commands” tab allows you to modify the content of the toolbar and the main menu of the current perspective.

The list in the left pane contains the command groups that include menu options, toolbar options or both. If you deselect a command group in the left pane, the menu and toolbar options associated with it will be removed from the current perspective. E.g. select the “Working Set Manipulation” option and press **OK**:



The **Edit** menu option and the main toolbar will contain additional options:

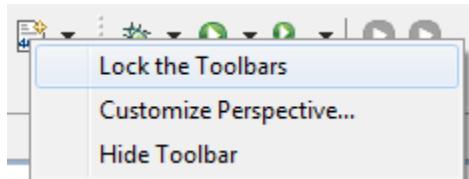


This is how they look on the toolbar:





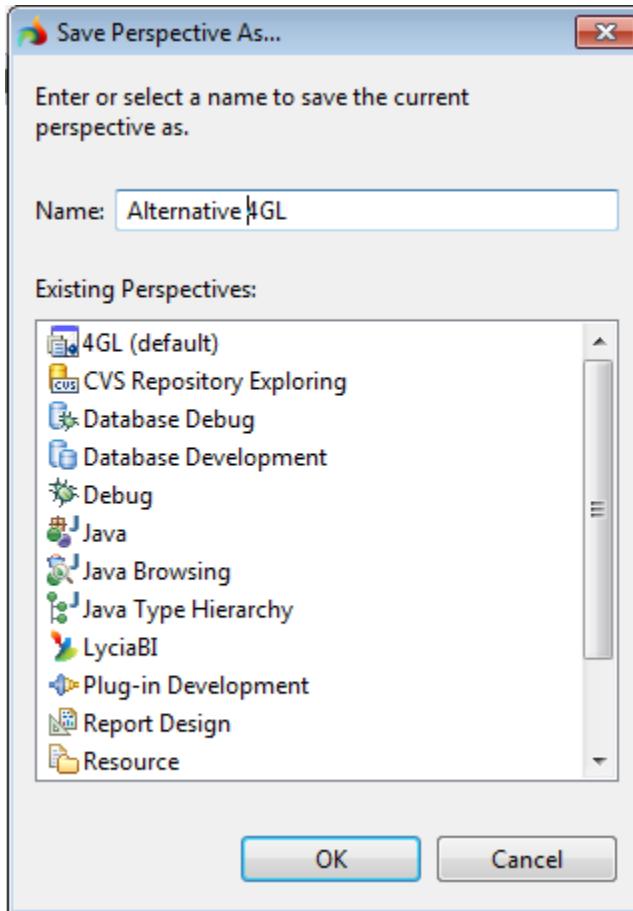
The toolbar buttons can also be moved around the toolbar so that you can adjust them to your needs, just click and drag the sets of buttons separated by vertical lines. When you have arranged the buttons as to your needs, you can lock the toolbar to prevent the buttons from being moved accidentally by right-clicking the toolbar and selecting the **Lock the Toolbars** option:



Choose the **Hide Toolbar** option in the toolbar context menu to hide the toolbar. Use the **Window -> Show Toolbar** option to restore it.

Saving Perspectives

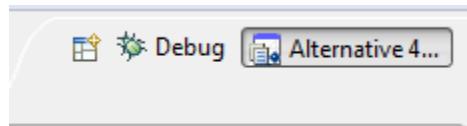
To save a customized perspective for future use, select the **Window -> Save Perspective As...** main menu option. Enter the name of your customized perspective in the Save As dialog:





	<p>Note: it is not recommended to use the name of an existing perspective for the new one, because in this case you will be unable to restore the default state of a built-in perspective.</p>
--	--

The icon of the current perspective on the shortcut bar will change:



In case, if you have customized a perspective but do not want to save the changes and want to go back to the default state of the perspective, use the **Window -> Reset Perspective...** menu option. This option will restore the state of a user-customized perspective in which it has been saved for the last time.

To delete a perspective, use the **Window -> Preferences -> General -> Perspectives** preferences page. Only user-created perspectives can be deleted.



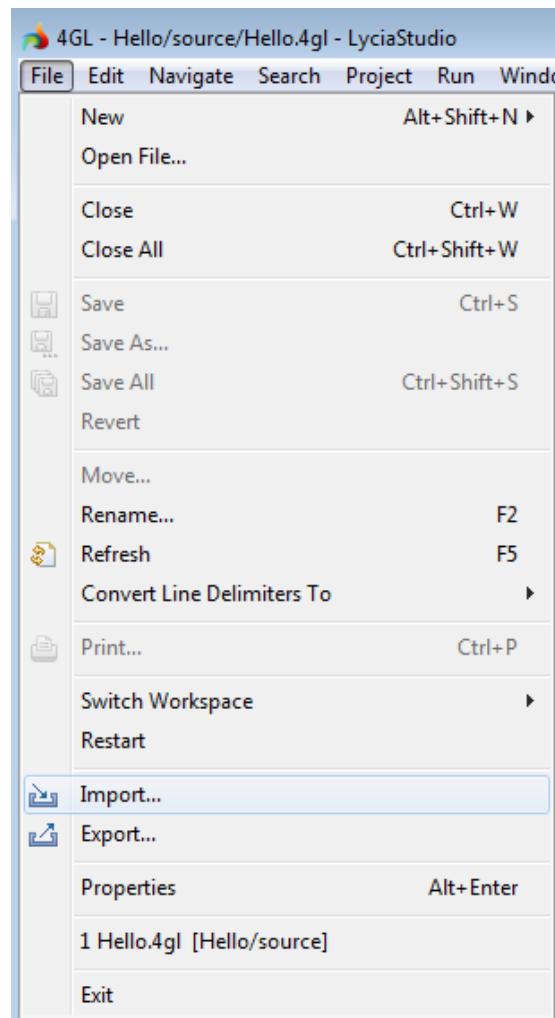
Using LyciaStudio

This section is designed to help the developer get used to any changes from previous versions of the development suite and to guide them through, step by step, common operations. This will include importing legacy projects, creating new projects from scratch and creating non 4GL-specific projects.

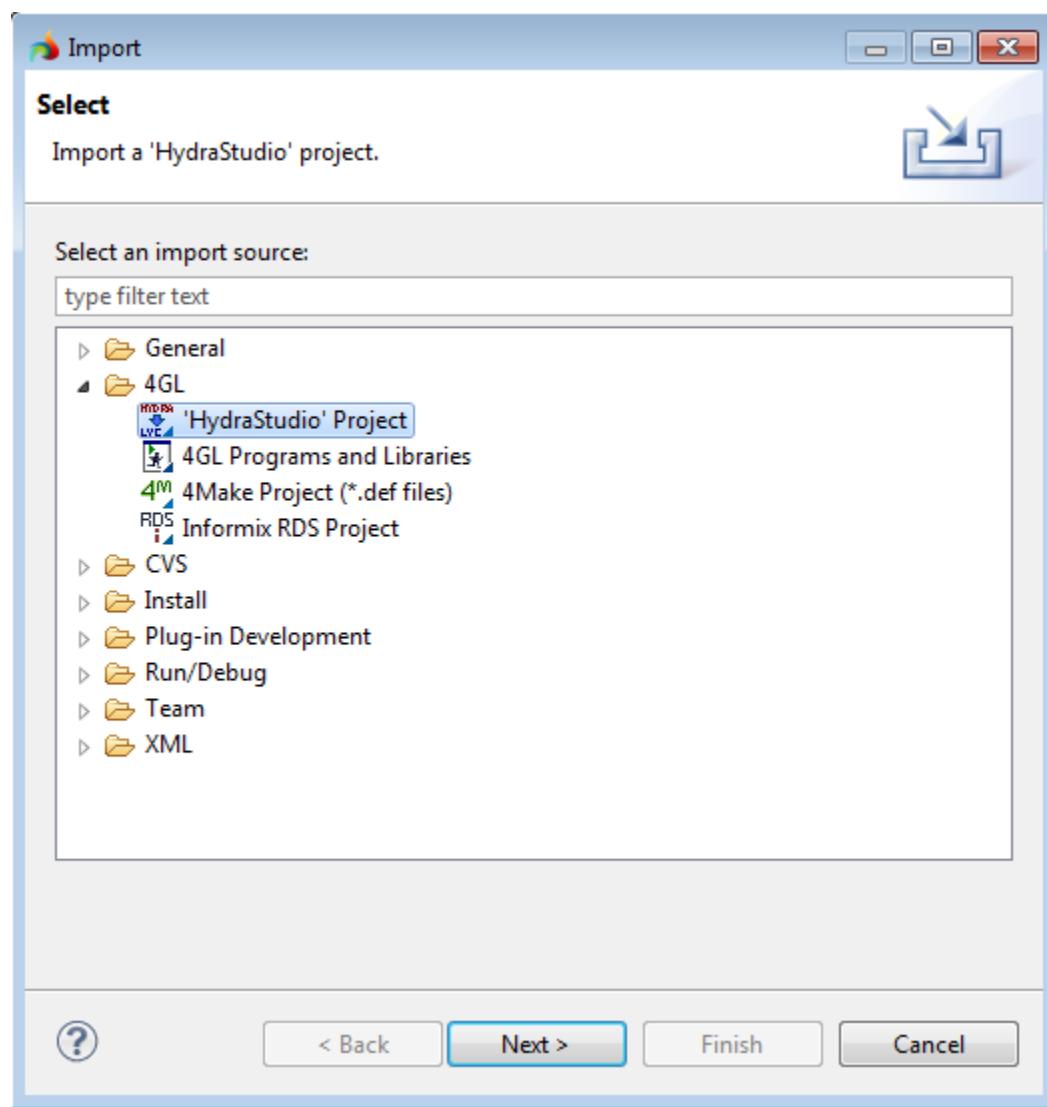
Importing a ‘HydraStudio’ V4 project

To work with ‘HydraStudio’ V4 projects, a developer needs to import the project into the LyciaStudio. This is a simple process, as follows:

1. Start the Studio
2. Click on **File -> Import**

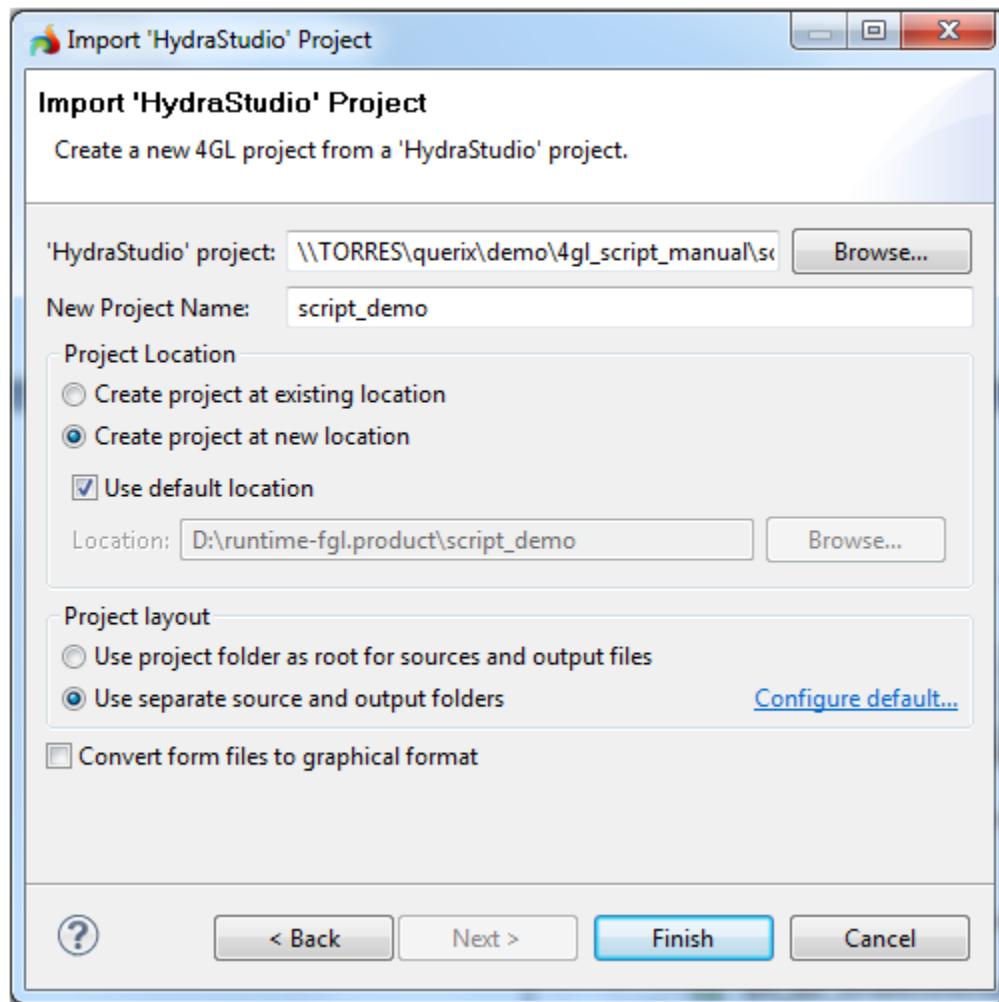


3. Select "4GL/Legacy HydraStudio Project" and click Next





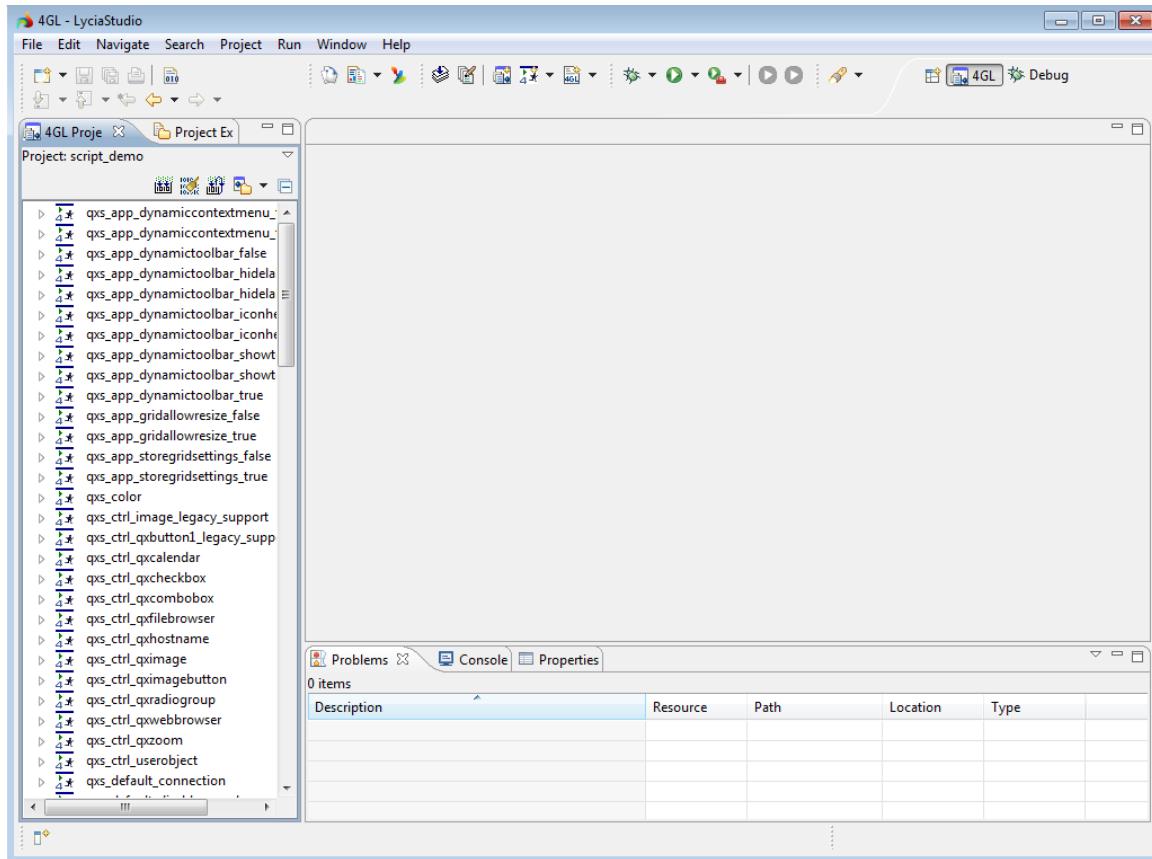
4. Browse for the project you wish to import (for this example, we will pick the script_demo shipped with 'Hydra4GL')



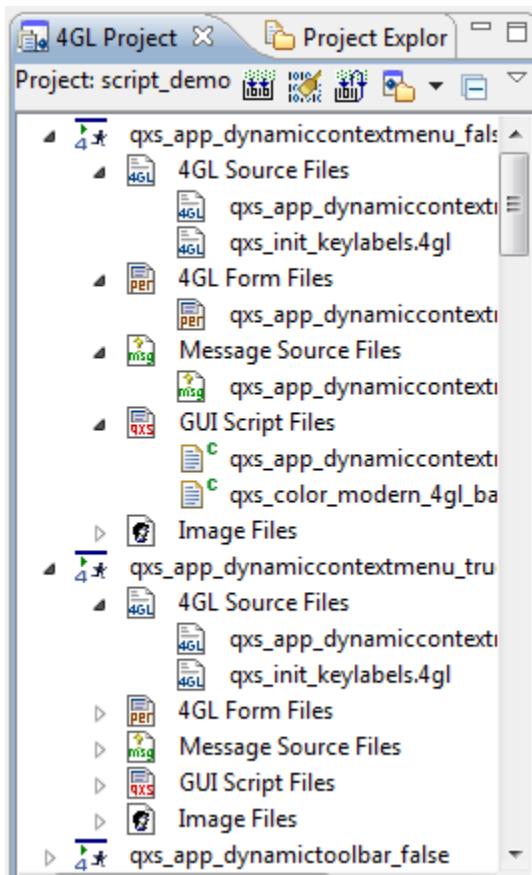
	<p>Note: form files are NOT automatically converted into a graphically editable format (.4fm). If you wish to edit them graphically, you will need to convert them.</p> <p>To do this, either select the option "Convert form files to graphical format" as seen above, or individually within the 4GL Project viewer by right-clicking a form file and selecting from its menu "Convert to graphical format" at any point.</p>
--	---



5. Click Finish



You can now work with your previous projects as you wish. The structure of an imported project will remain the same as it was within 'HydraStudio', as shown below. Expanding a program in the '4GL Project' view will allow you to edit the files as you desire.



	<p>Note: external library requirements are not automatically imported on legacy projects. An external library is defined as a required library, but not built as part of the same project.</p> <p>With Lycia, 4GL targets cannot have a dependency on a static C library. It is also unknown whether the library is a C library or a 4GL library during import. Lycia also allows the linking to .dll files as well as .lib files, and cannot tell if the .lib is a static library or an imports library.</p> <p>To import them, the developer must import them manually, so the correct dependencies and libraries can be added.</p>
--	---



Importing a Lycia Project

You cannot use the "Open file..." option to open a project previously created in Lycia and to work on it. To work on a Lycia project that is not located in the current workspace and is not displayed in the Project Explorer View, you must import this project. A project that has been previously exported by means of LyciaStudio must be imported with the help of 'Import' option. Note that if you export a project, it will not be deleted from the Project Explorer View and you will be able to work on it. Importing is required only if you have deleted the exported project from the Project Explorer View, or if you need to work with a project which has been created and exported using another computer.

Importing Existing Projects into Workspace

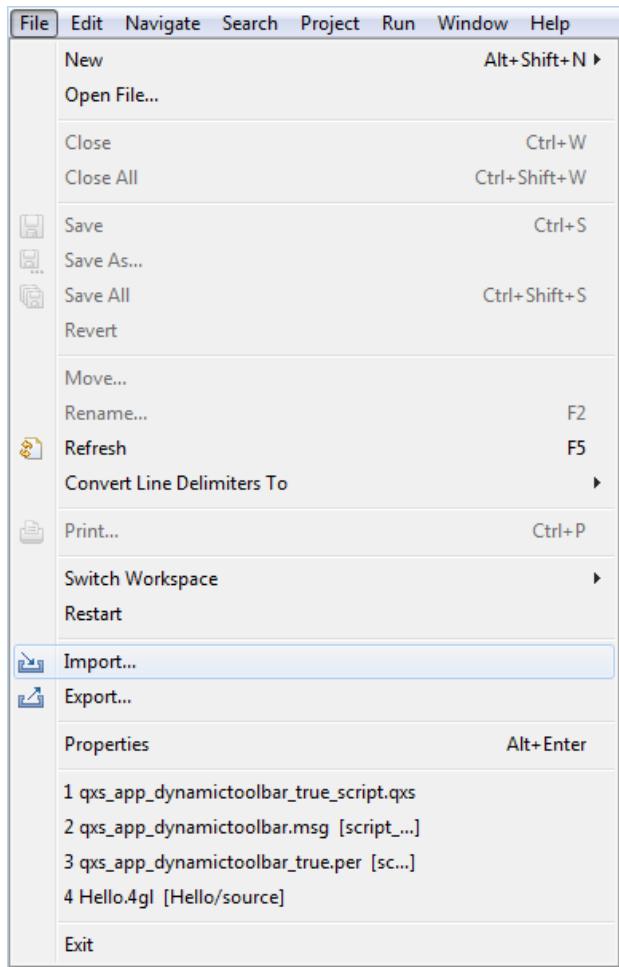
There are two ways to import a native Lycia project into LyciaStudio using the "4GL -> Existing Projects into Workspace" option. Which import method to use depends on the method used to export the project.

The "4GL -> Existing Project into Workspace" option is used to import whole projects previously exported by either "General -> Archive file" export option or "General -> File System" export option. The "4GL -> 4GL Programs and Libraries" option can be used to export and import individual programs and libraries, it is advisable not to use it to export and import complete projects.

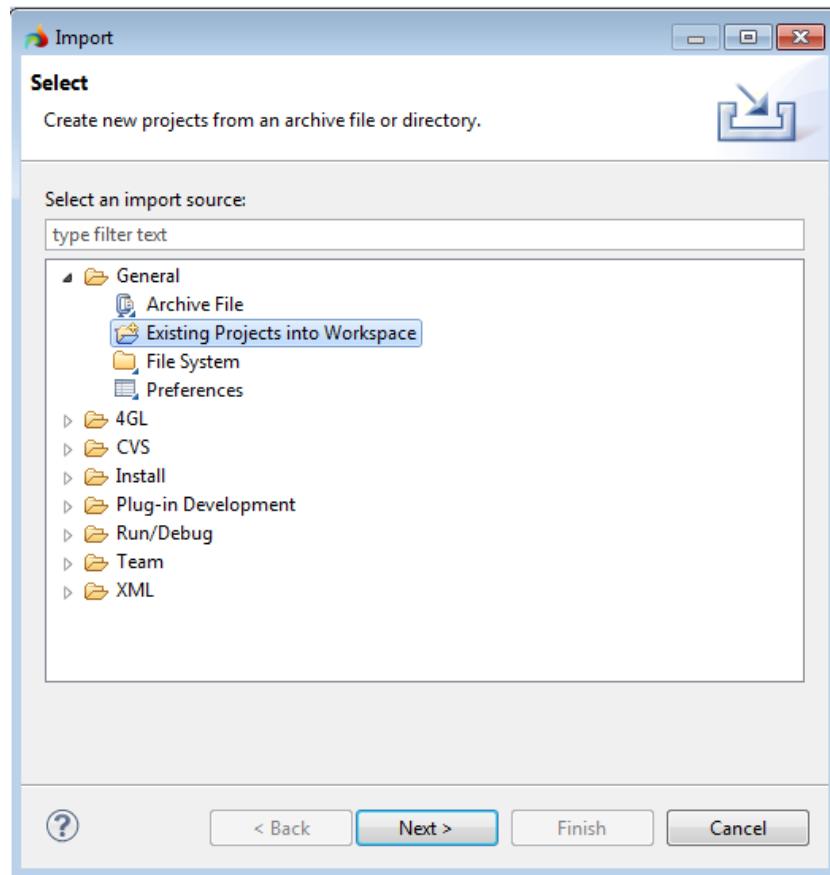
Importing a Non-Archived Project

If you have a project created in Lycia and exported by means of the "4GL > Folder System" export option, do the following to import this project:

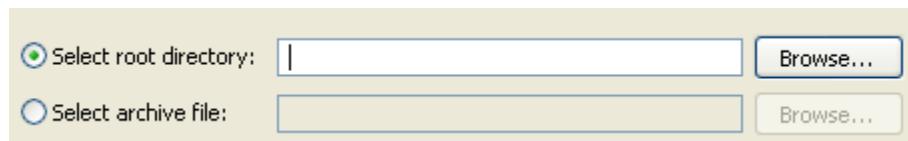
1. Start the Studio
2. Click on **File -> Import**



3. Select the **General -> Existing Projects into Workspace** option and press **Next** button:

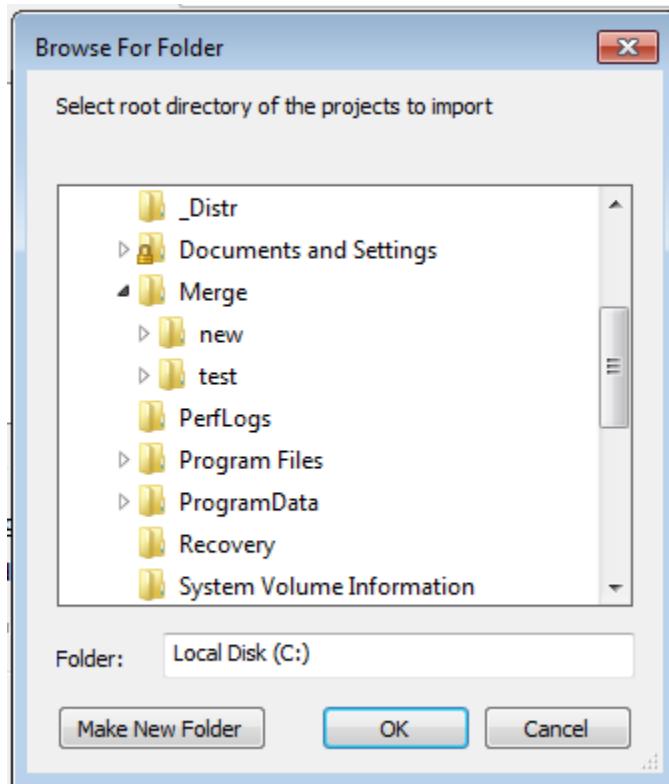


4. Select the "Select root directory" radio button and press the **Browse...** button in the dialog box that will appear:



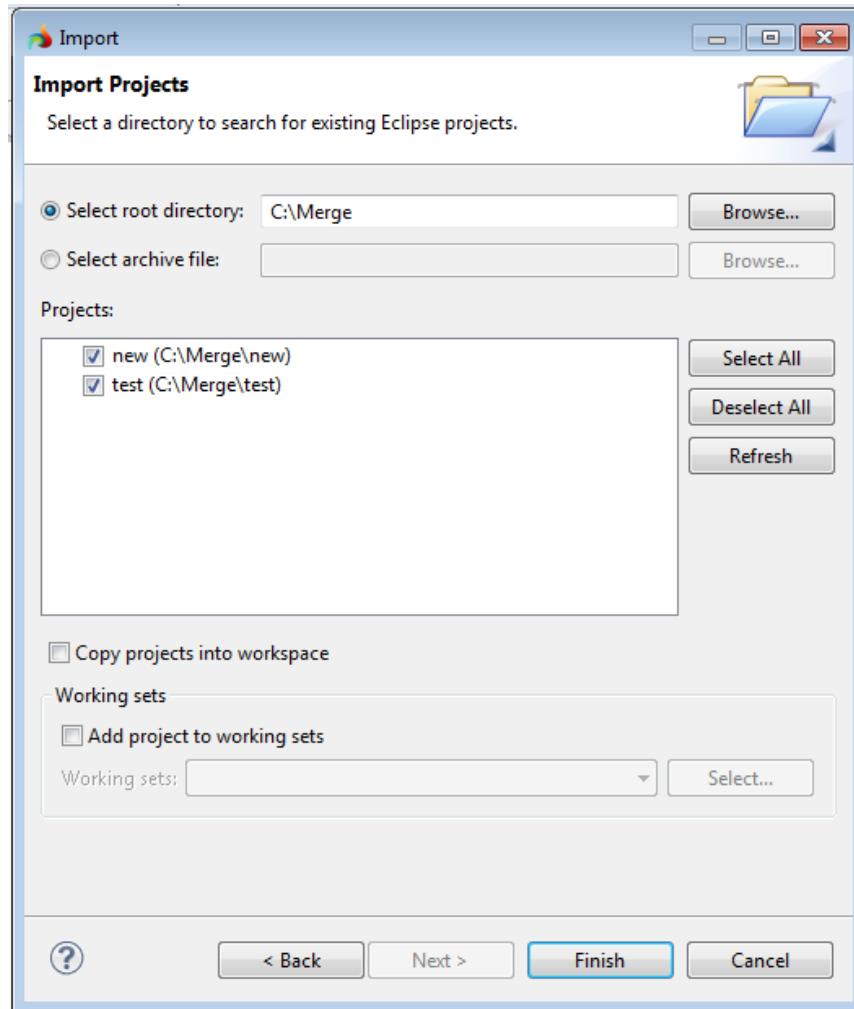


5. Select the folder of the project you want to import in the pop-up window and press **OK**:

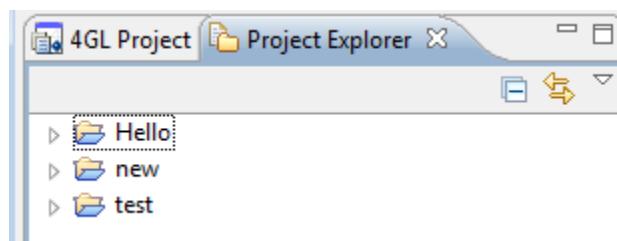




6. You will see the project which can be imported in the Projects field. You cannot import a project which is already present in the current workspace. If a folder contains several projects, the projects that do not exist in the current workspace will be displayed in the Projects field. Select the projects you want to import.



7. The imported project will appear in the Project Explorer View and in the 4GL Project View, if you have selected more than one project, they will be imported and displayed as independent projects:





Importing an Archived Project

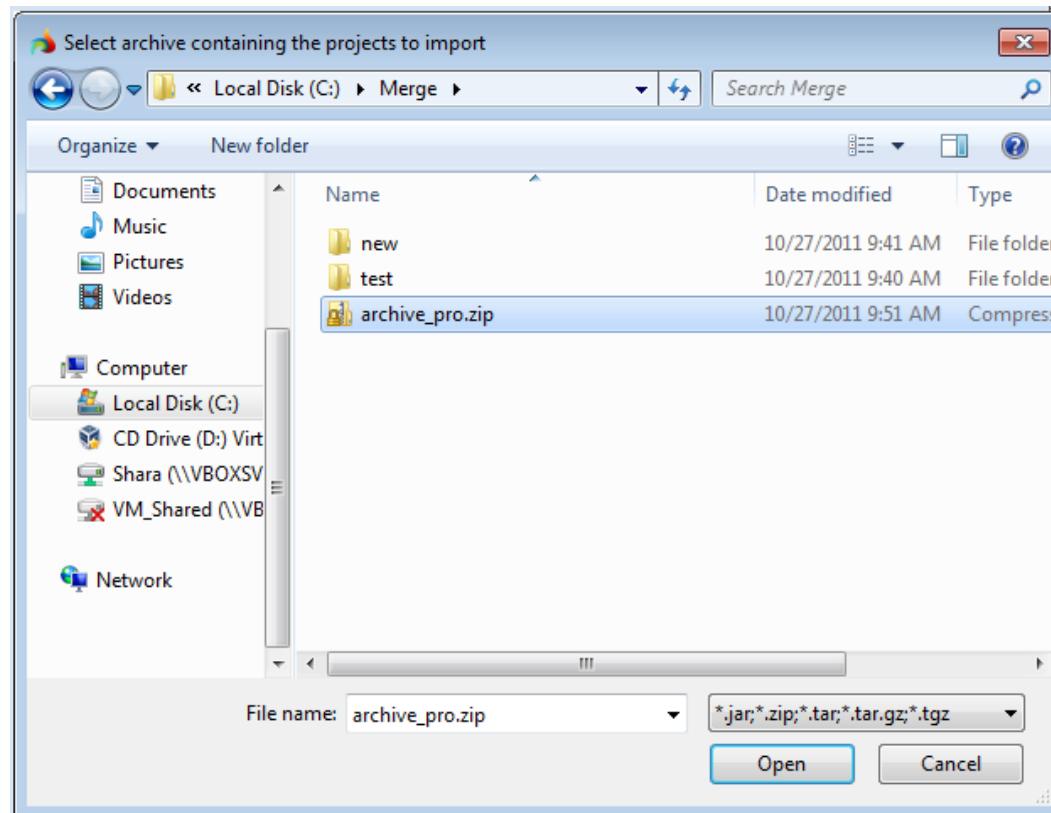
If you have a project created in Lycia and exported by means of the "4GL > Archive File" export option, do the following to import this project:

Repeat the steps 1-3 described in the "Importing a Non-Archived Project" section above.

Select the "Select archive file" radio button and press the **Browse...** button in the dialog box that will appear:

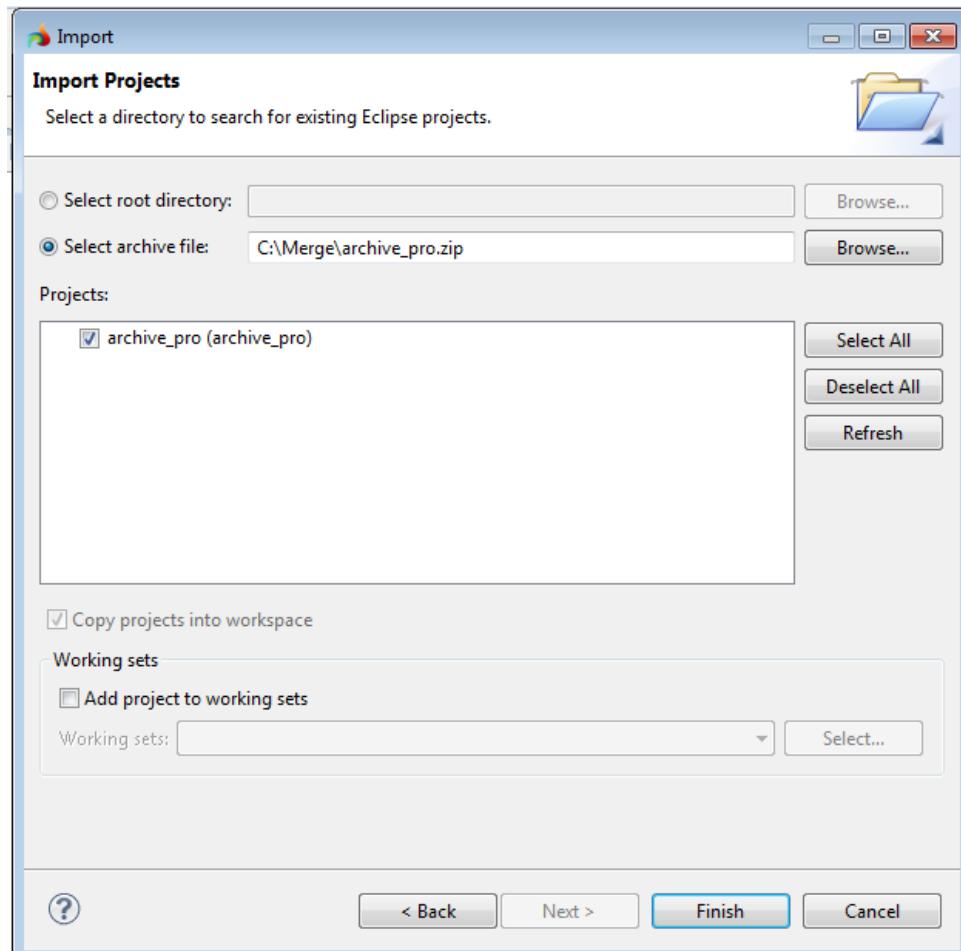


Select the archive file which has been previously created by the "4GL > Archive File" export option and press **Open** button:

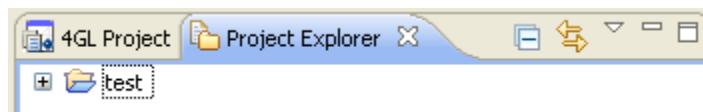




You will see the name of the project which the archive file contains, select and press the **Finish** button:



The imported project will be displayed in the Project Explorer View and in the 4GL Project View:



Importing Files, Folders, Programs, and Libraries into an Existing Project

If you want your current project to use source files which are not at the moment an integer part of the project, you can import source files, a set of folders with the source files, 4GL programs, and libraries into the current project.

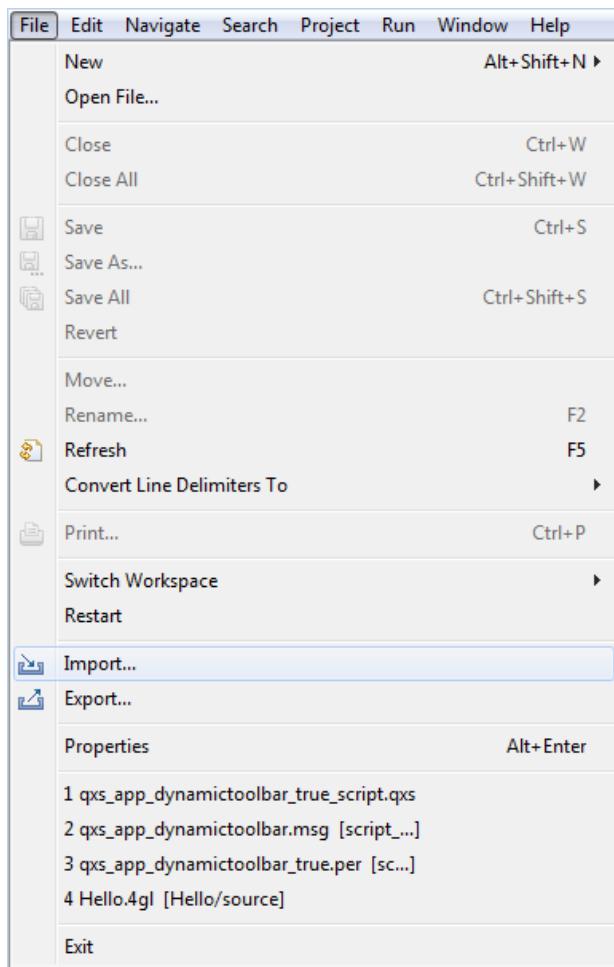


When you import individual source files of any kind or a library, they are imported into the workspace but are not included into the project. To include them into the project and to make them visible in the 4GL Project view you must add them to the project requirements after you have imported them. This can be done by right-clicking a project in the 4GL Project view and selecting the **Add Requirements** option from the context menu. For more details see the "Creating new files" and "Using libraries" sections below.

Importing 4GL Programs and Libraries

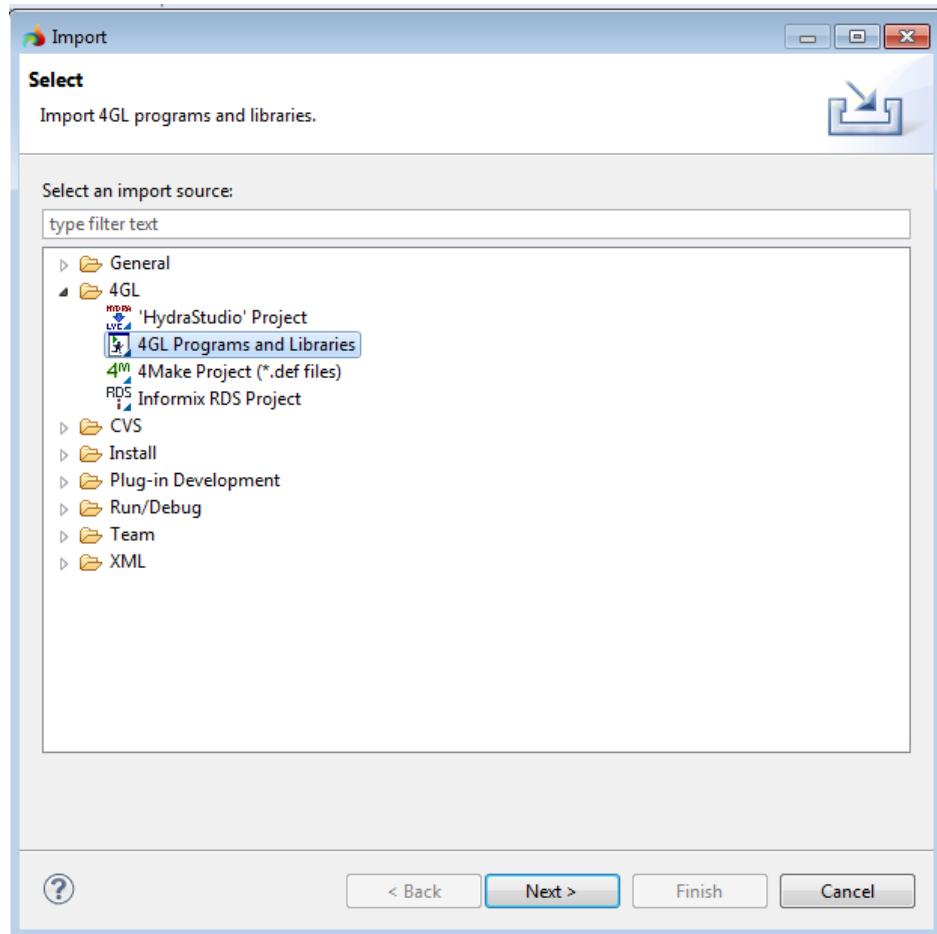
The "4GL -> 4GL Programs and Libraries" option is used to import programs or libraries into an existing project. It should not be used to import a complete project. This option can import programs and libraries from the archive files exported by means of the "4GL -> 4GL Programs and Libraries" export option. To import programs and libraries, do the following:

6. Start the Studio
7. Click on **File -> Import**





8. Select the **4GL -> 4GL Programs and Libraries** option and press **Next**:

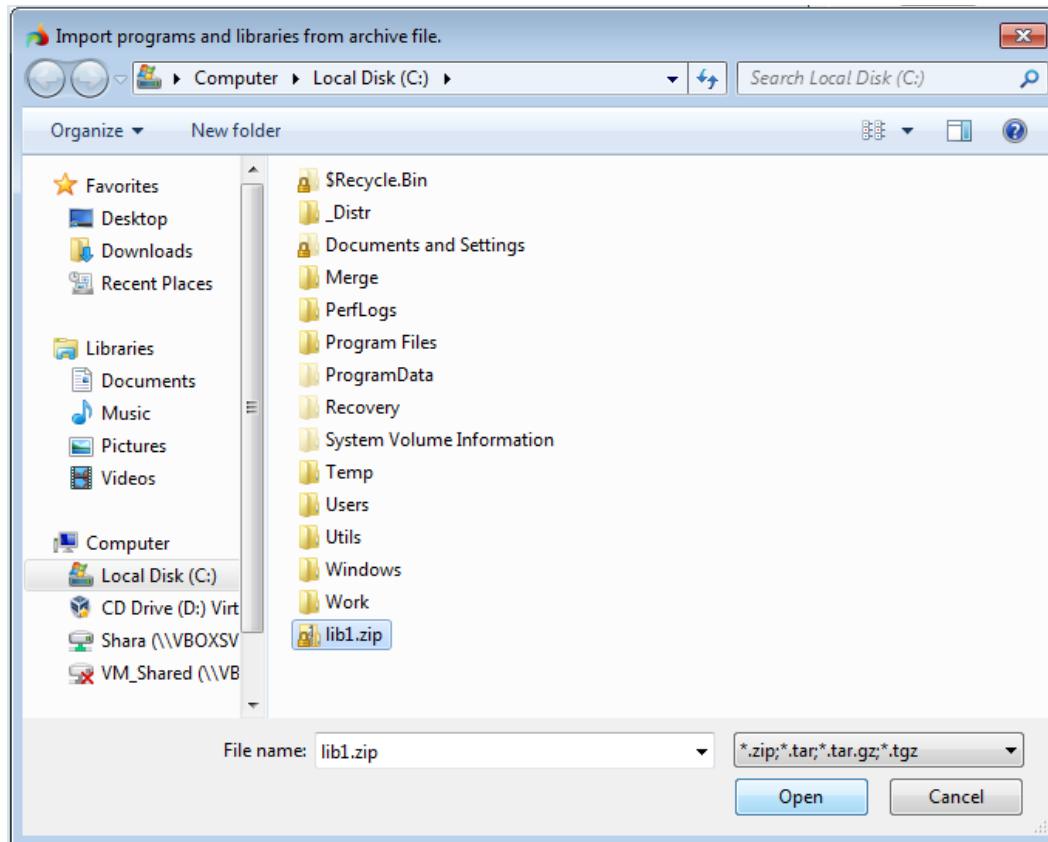


9. Press **Browse...** button in the dialogue box that will appear:



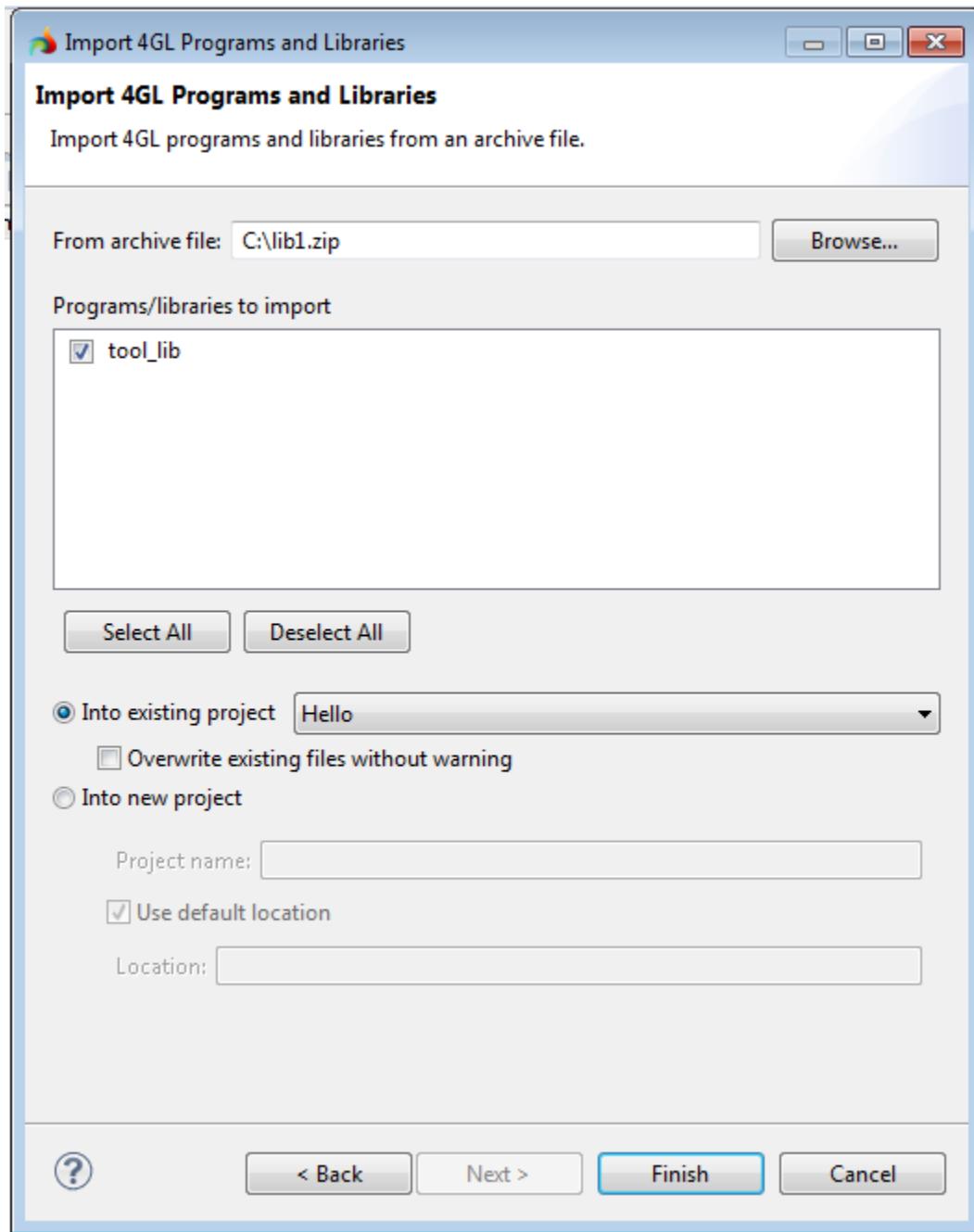


10. Choose the archive file to which the project has been exported and press **Open**:

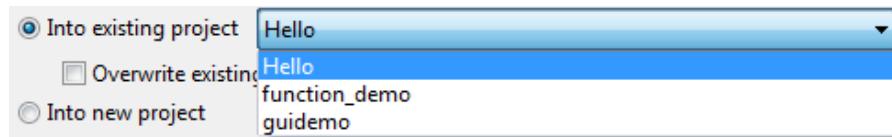




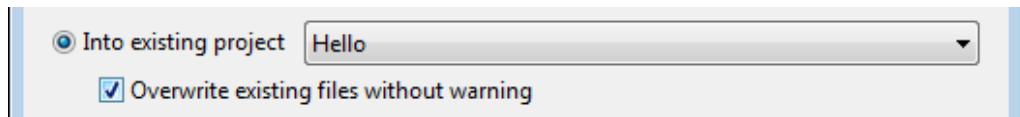
11. You will see the programs and libraries of the selected project in the Project field. Tick off the programs and libraries you want to import:



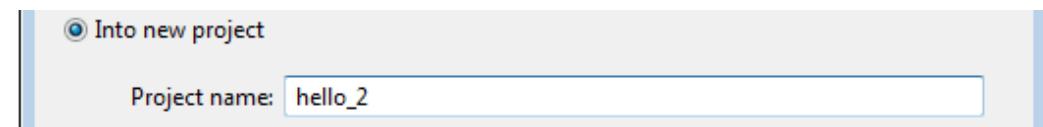
- a. If you want to merge the project with the existing one, choose the "Into existing project" option and select the desired project from the combo box list:



- b. If you want to overwrite the already existing files, tick off the "Overwrite the existing project" option:

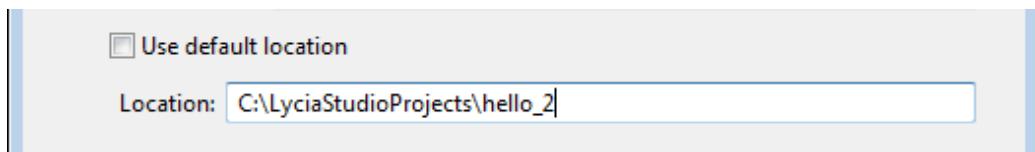


- c. If you want to import a program or a library into a new project, select the "Into new project" option and enter the name for this project:

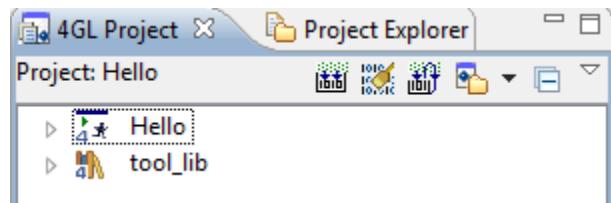


	Note: The "4GL -> 4GL Programs and Libraries" option (both in import and export) can be used to import and export one or more programs and libraries. It is not the preferred way to import or export a whole project. To import or export the whole project use "4GL -> Existing Project into Workspace".
--	---

- d. You can change the default location to which the project will be imported by removing the tick from the "Use default location" option. The specified location cannot overlap the workspace location.



12. After you have selected the programs which must be imported and the import method, press **Finish** button. The selected programs and libraries will become an integer part of the existing project (or a new project) and they will be displayed in the Project Explorer View and in the 4GL Project View:

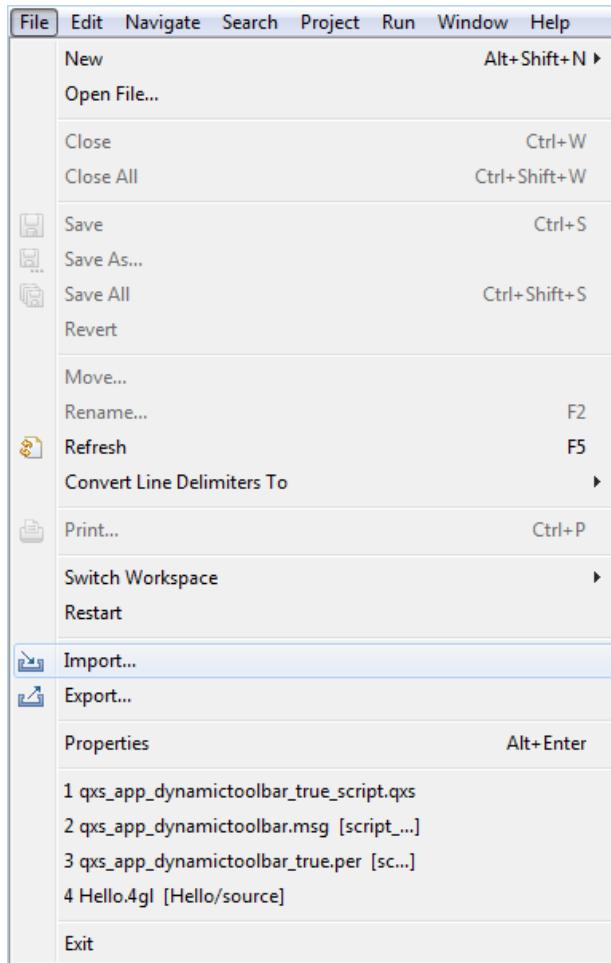


Importing File System

You can import a complete file system or a part of it from a project previously exported by means of the "General -> File System" export option into an existing project. You cannot import files and folders which have not been exported, it will may cause unpredictable results.

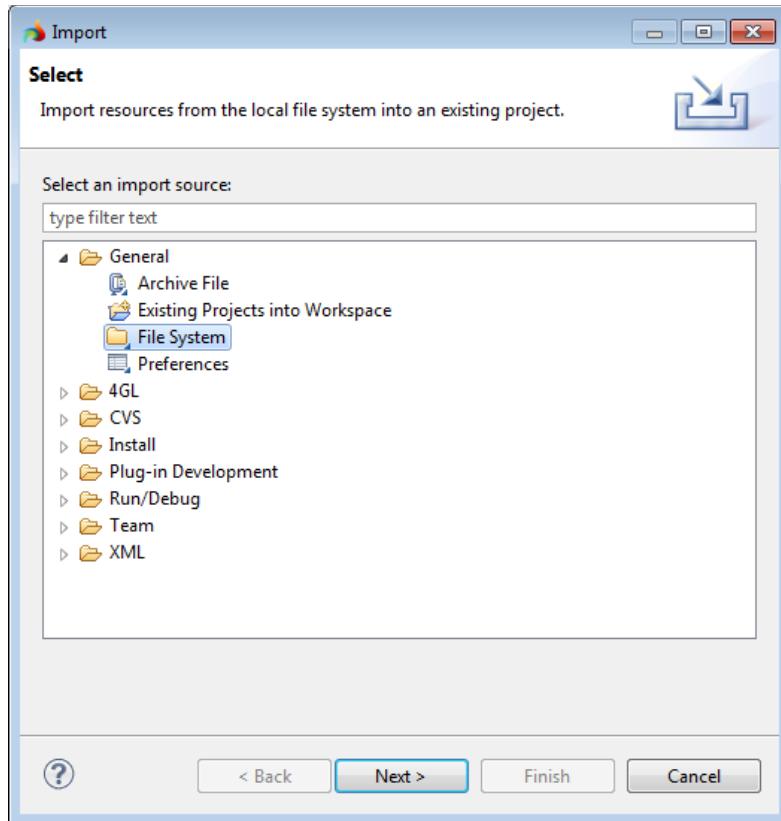
To do it, follow these steps:

1. Click on **File -> Import**

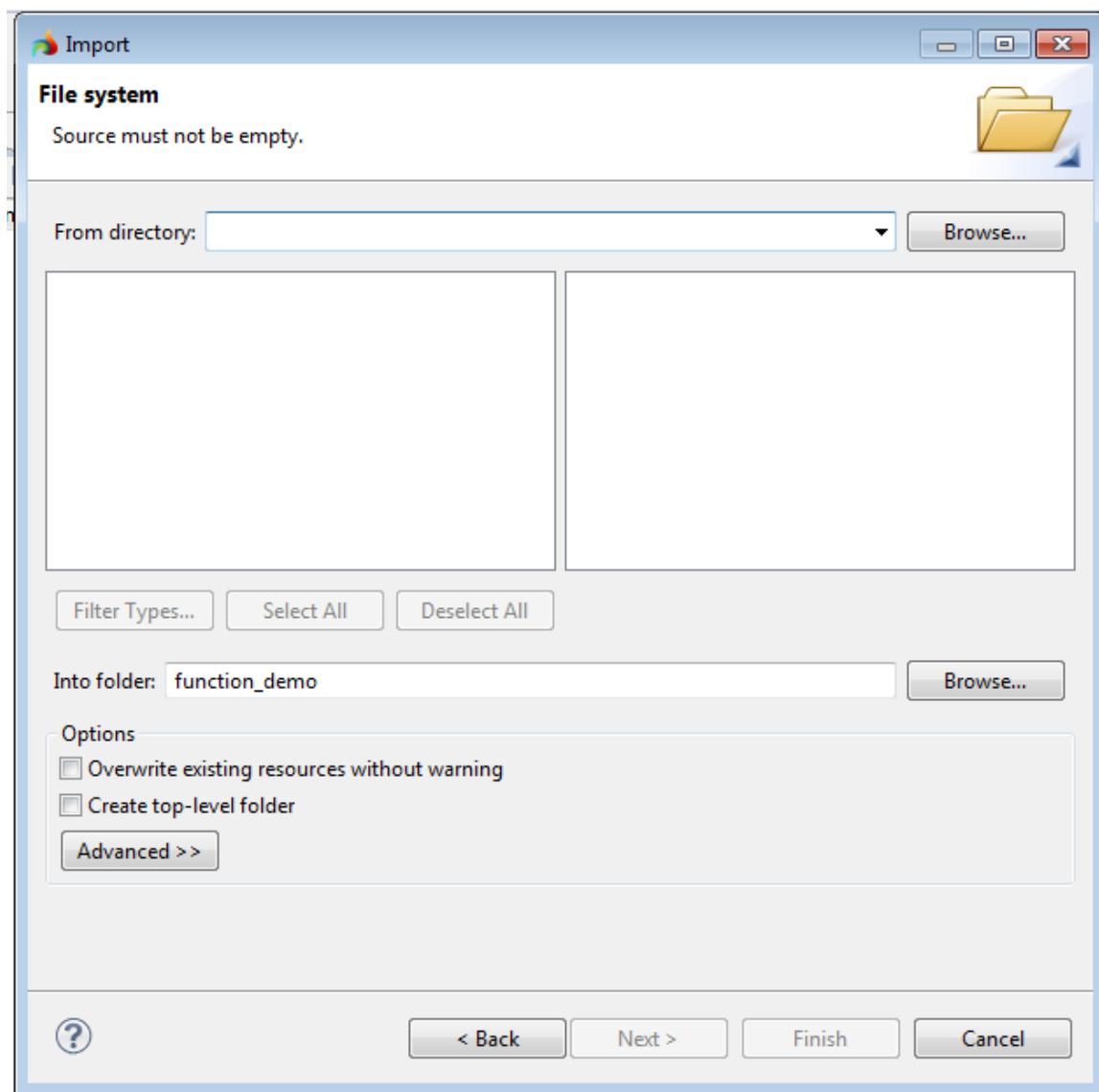




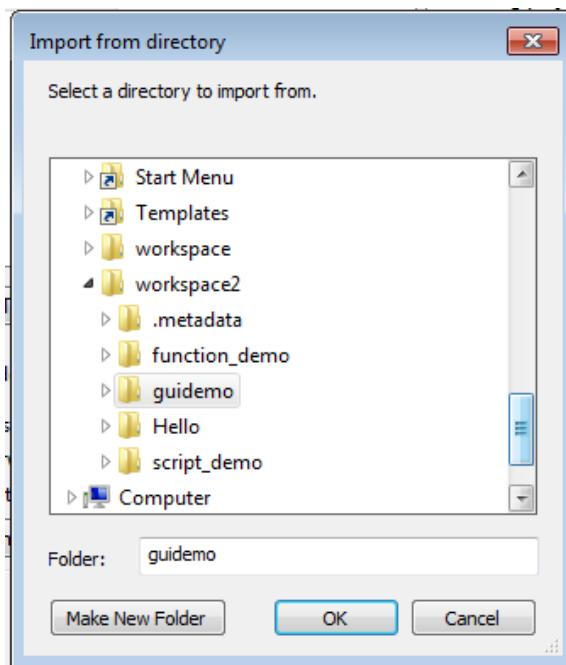
2. Select the **General -> File System** option and press **Next**:



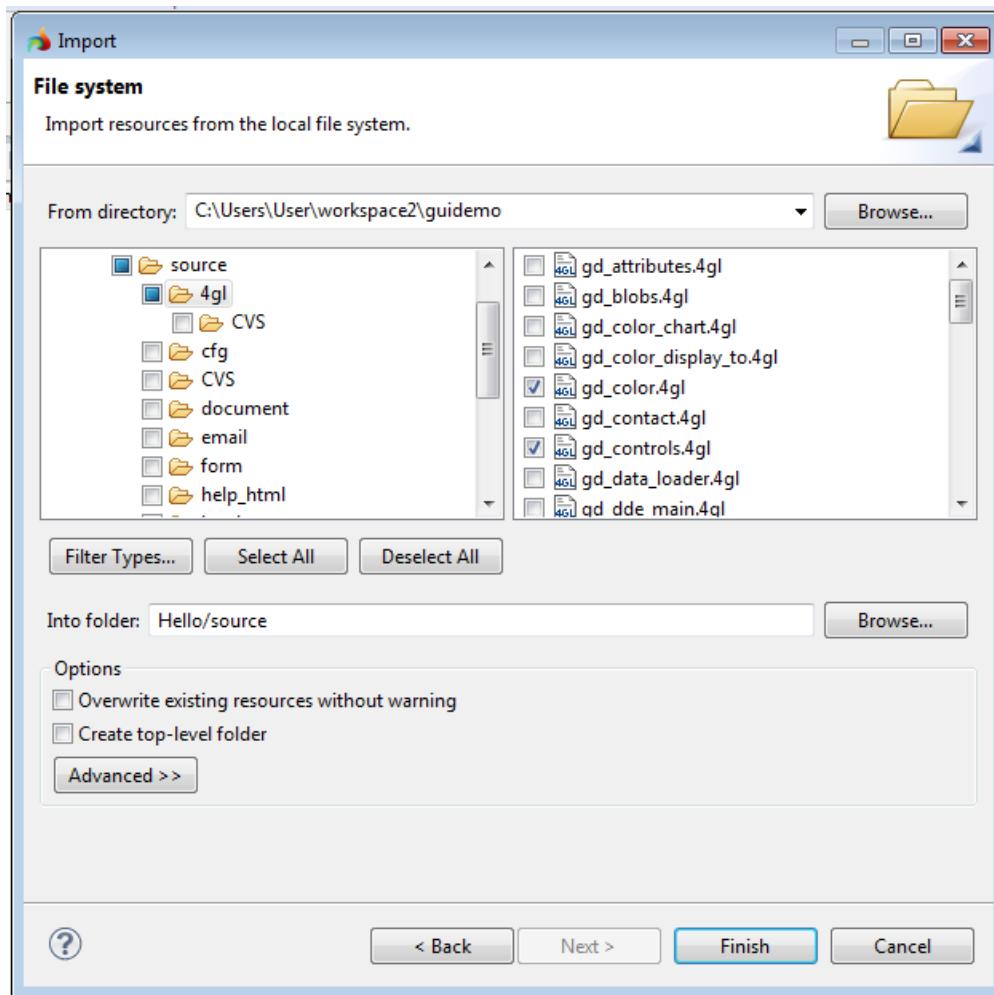
3. Press **Browse** button in the dialogue box that will appear:



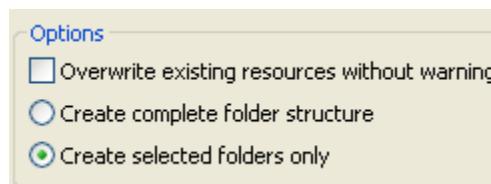
4. Choose a project folder from which you want to import the files and press **OK**:



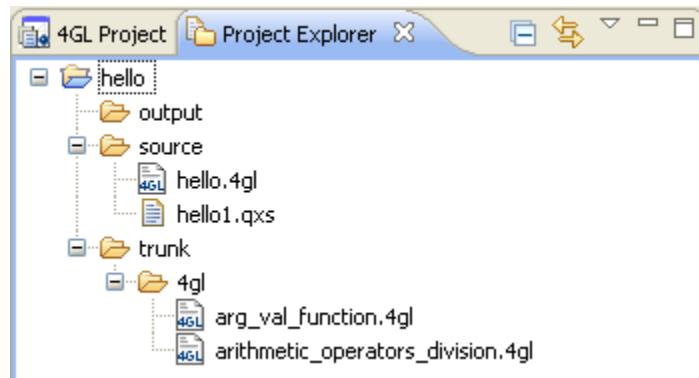
5. Select the subfolder which contains the required files from the list on the left, the files contained in the folder will be displayed in the list on the right. Tick off the files you want to import.



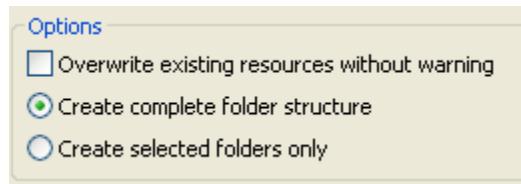
6. You can import only those folders which are included in the path from the selected folder to the selected files, or import all the folders that contain the selected files up to the root directory:
 - a. To import only related folders select "Create selected folders only" option and press **Finish**:



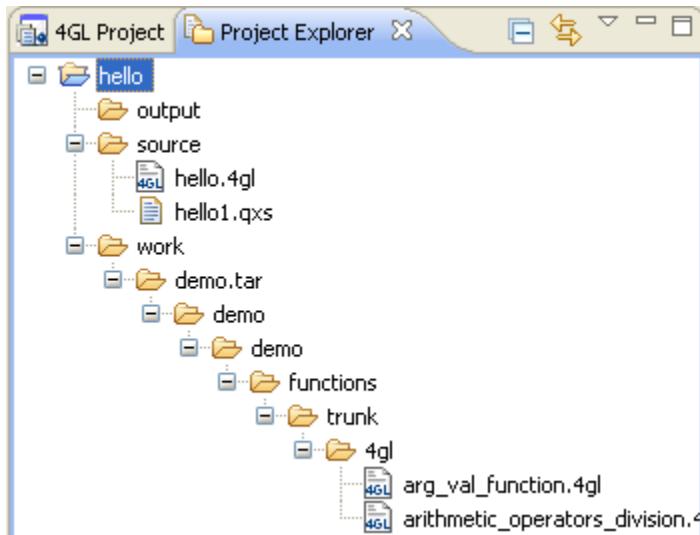
The Project Explorer View will contain all the folders from the root folder in the list on the left to the folder containing the imported file or files:



- b. To import all the folders up to the root directory select the “Create complete folder structure” and press **Finish**:



The Project Explorer View will contain all the folders from the root directory (in this case "work" folder in C:\ root directory) to the folder containing the imported file or files (4gl):

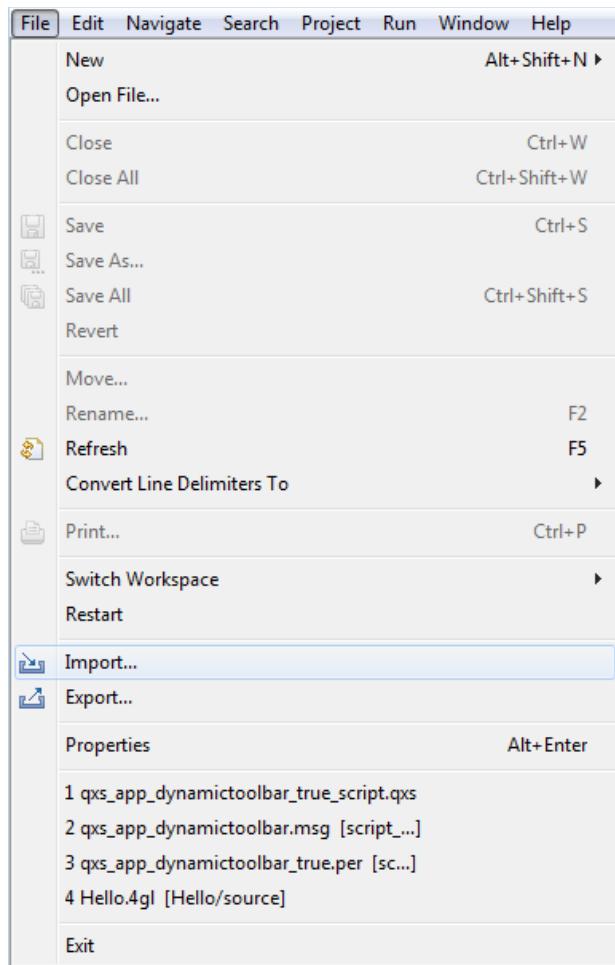


Importing Archive Files

You can import resources from an archive file into an existing project. The archive file must be a project previously exported by means of the “4GL -> Archive File” export option. To import an archive file, do the following:

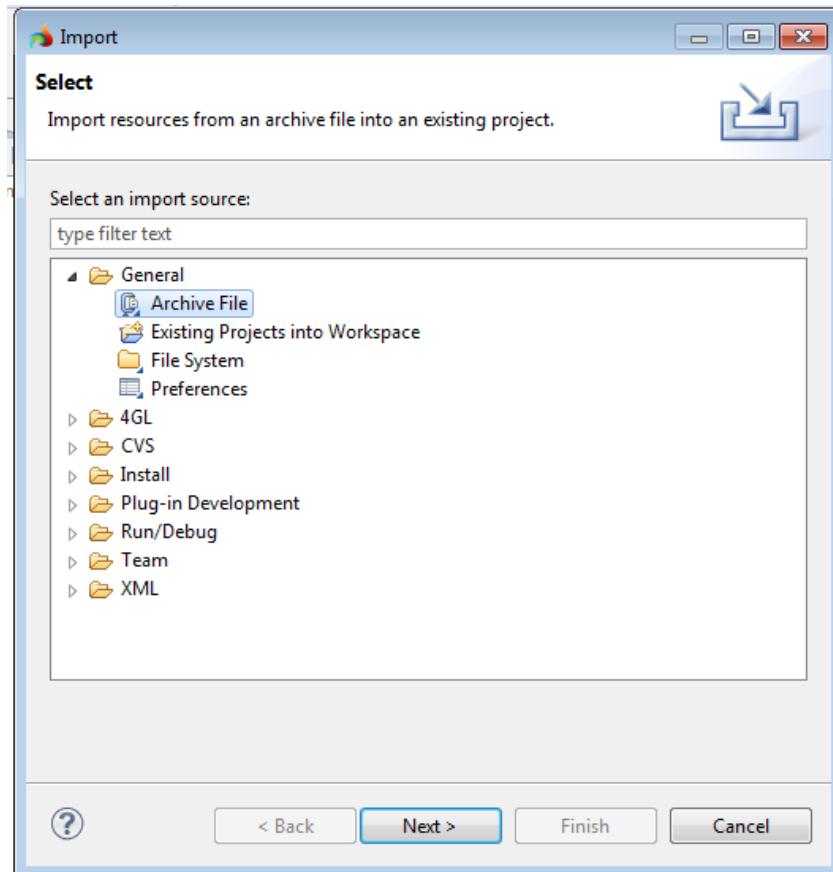


1. Click on **File -> Import**



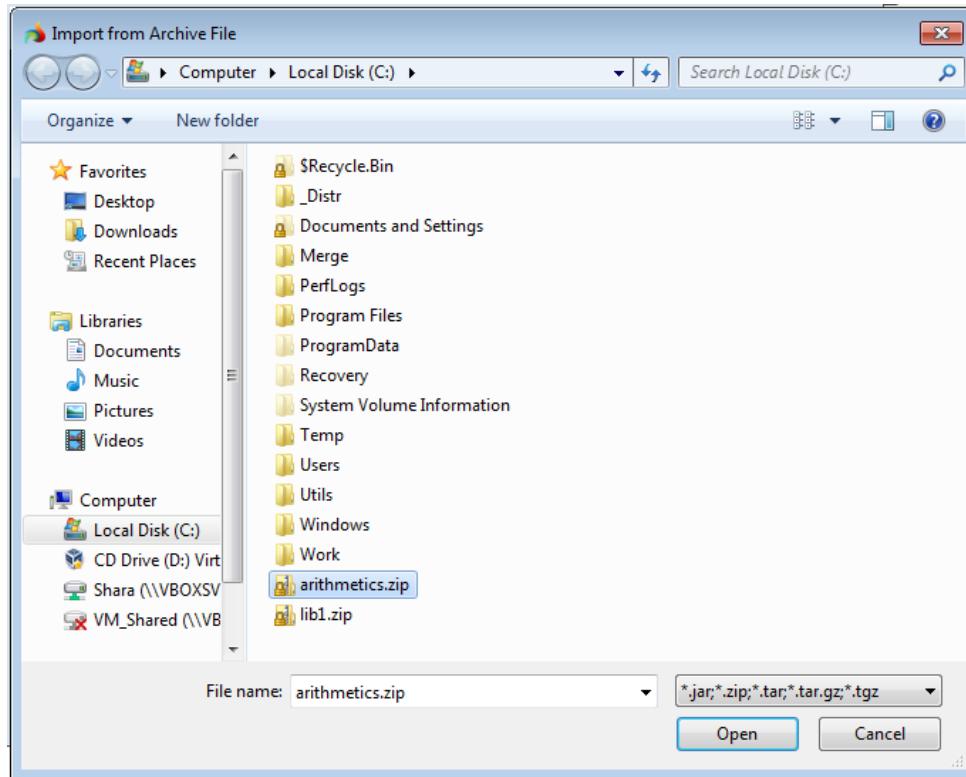


2. Select the **General -> Archive File** option and press **Next**:



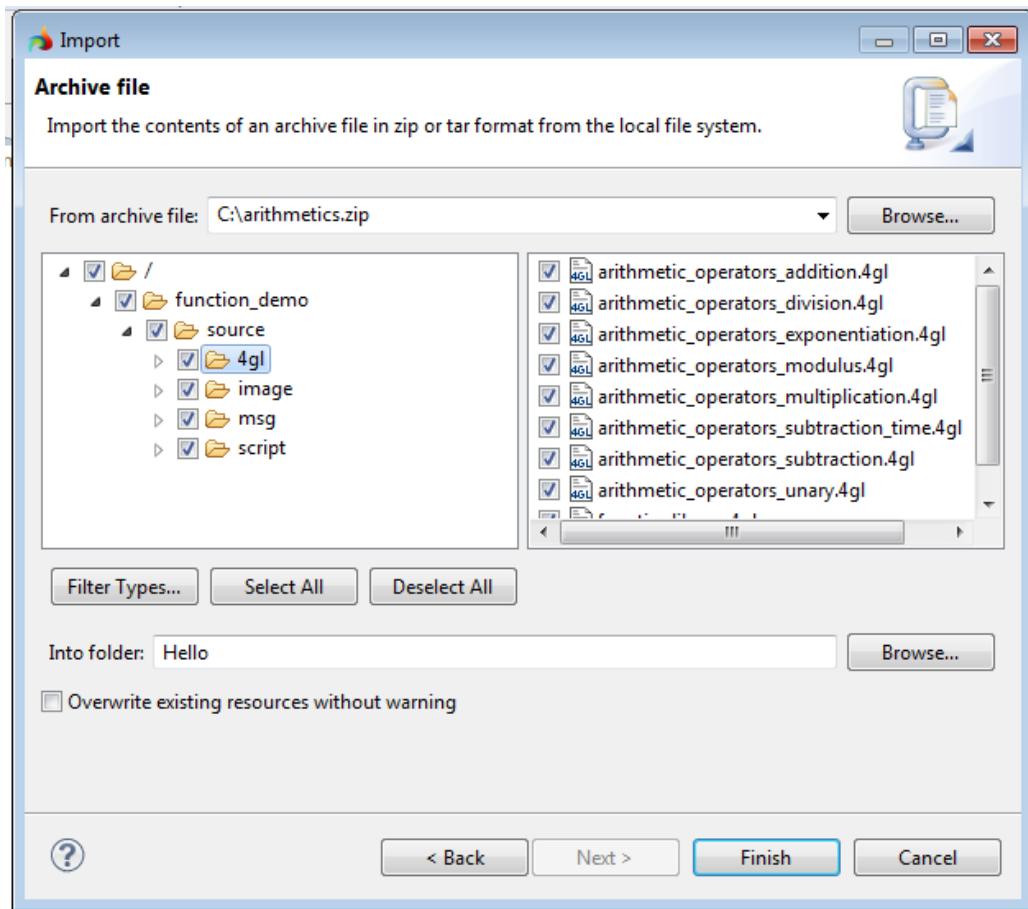


3. Press **Browse** button next to the "From archive file" field in the dialogue box that will open.
4. Select an archive file the content of which you want to import:



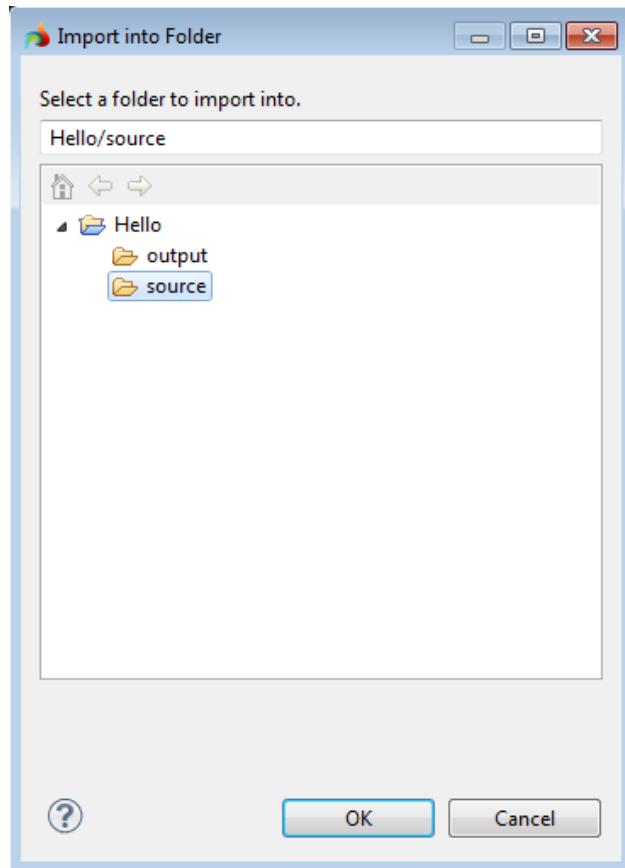


5. The content of the archive file will be displayed in the field on the left, tick off the files which you want to add into the selected project:





6. You can select the project and the folder for the imported files, if you press **Browse...** button next to the "Into folder" field:



7. Press **Finish** button.



Exporting a Project

To pass a project developed in LyciaStudio from one PC to another, the project needs to be exported. After it has been exported, it can be imported into any other LyciaStudio. If you try to import a Lycia project that has not been exported previously, it may result in unpredictable behaviour.

There are three methods of export:

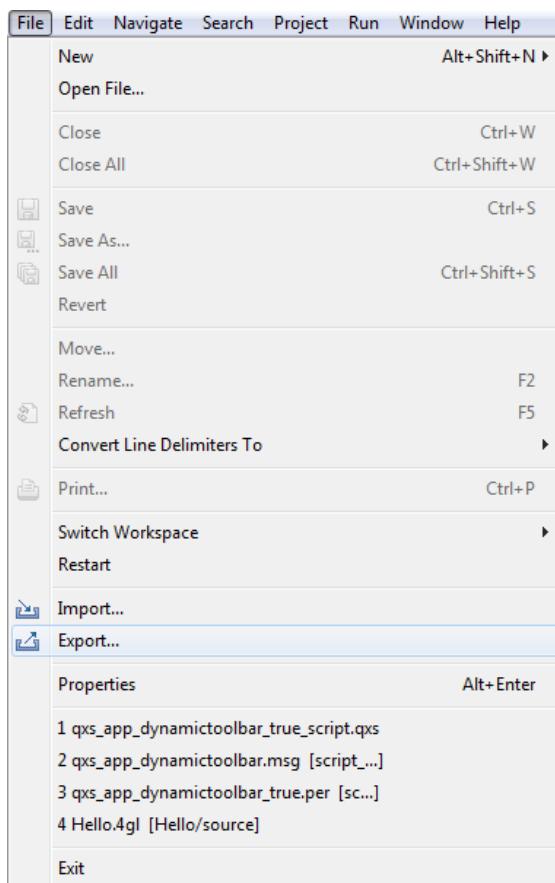
- export to an archive file
- export directly to the file system
- export separate programs and libraries.

The first two methods can be used to export a complete project. To import complete projects exported by means of these two options, use the **Import -> General -> Existing Projects into Workspace** option. The programs and libraries exported by means of the **Export -> 4GL ->4GL Programs or Libraries** option cannot be imported using this option. For more information see the Importing section of this document.

Exporting a project to the file system

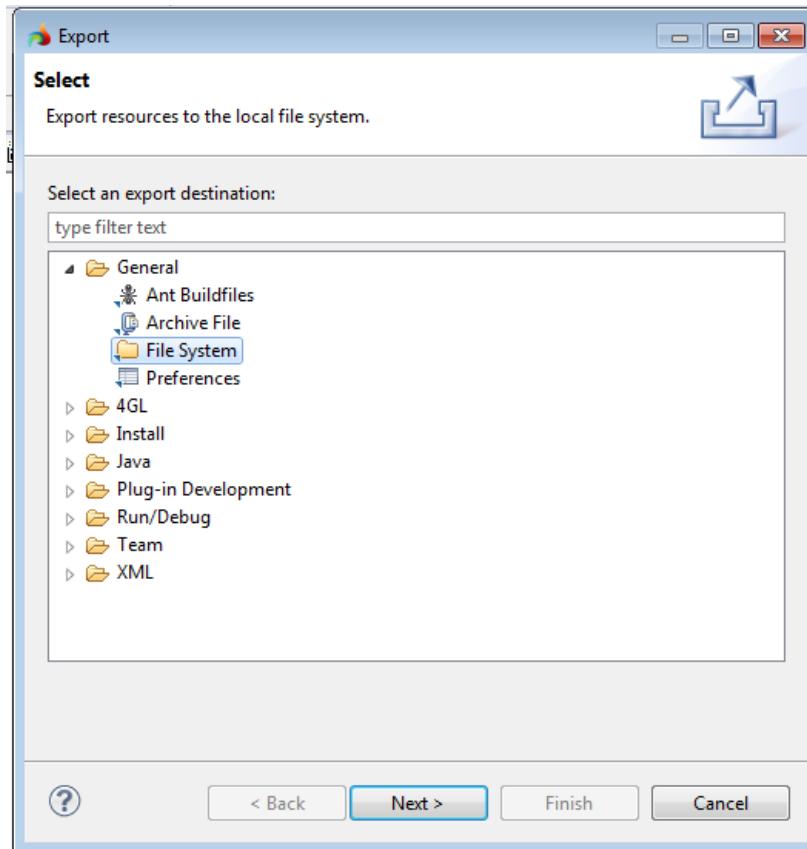
To export a project that is currently located in your Workbench to the file system, follow these steps:

1. Start the Studio
2. Select the project you want to export and click on **File -> Export**



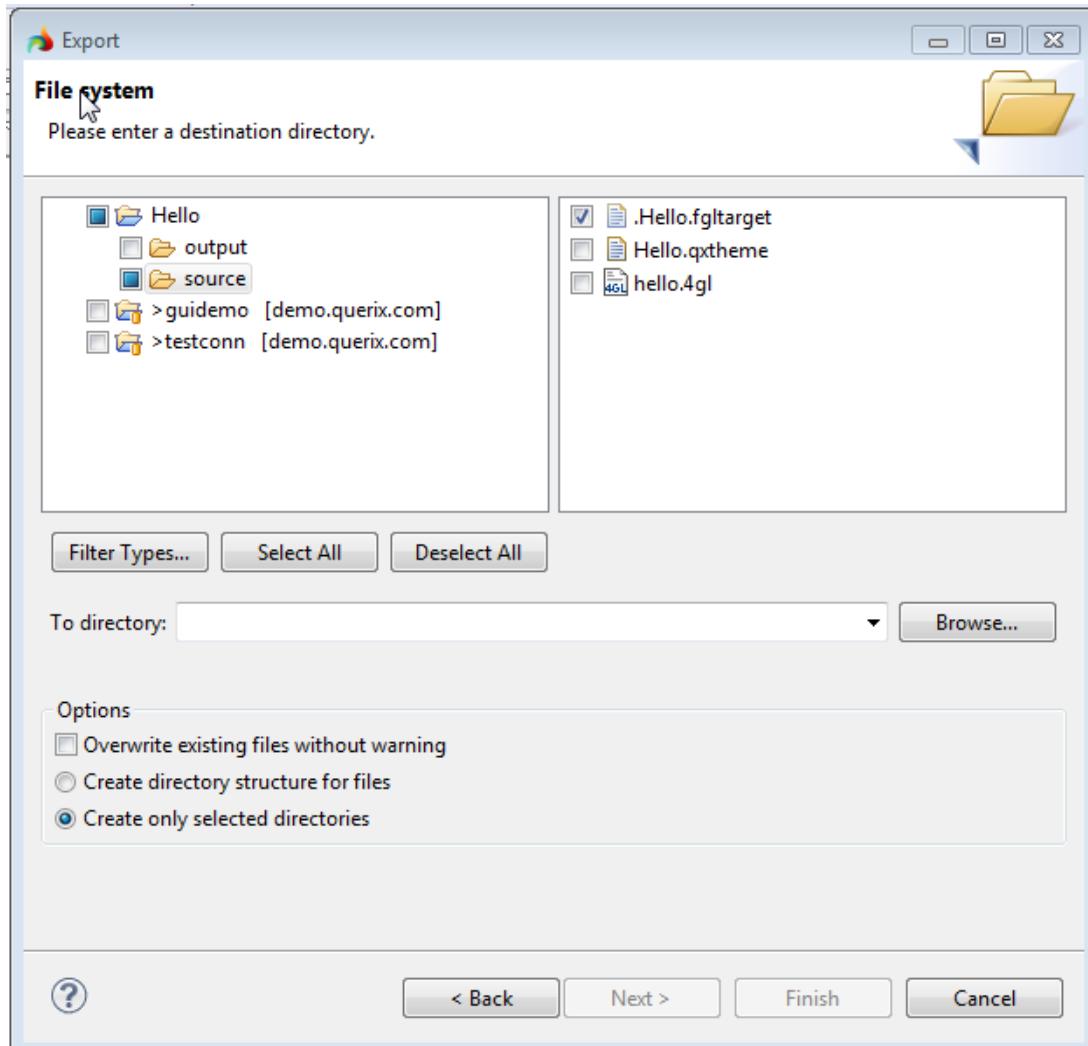


3. Select the **General -> File System** option and press **Next** button:





4. You will see the project folders in the left pane. The files that are contained in the selected project folder are displayed in the right pane. By default, source files are not automatically selected:



5. To be sure that you export all the files there are in the project, press the **Select All** button. Remove the tick from the output folder if you do not need to export the result of the compilation:

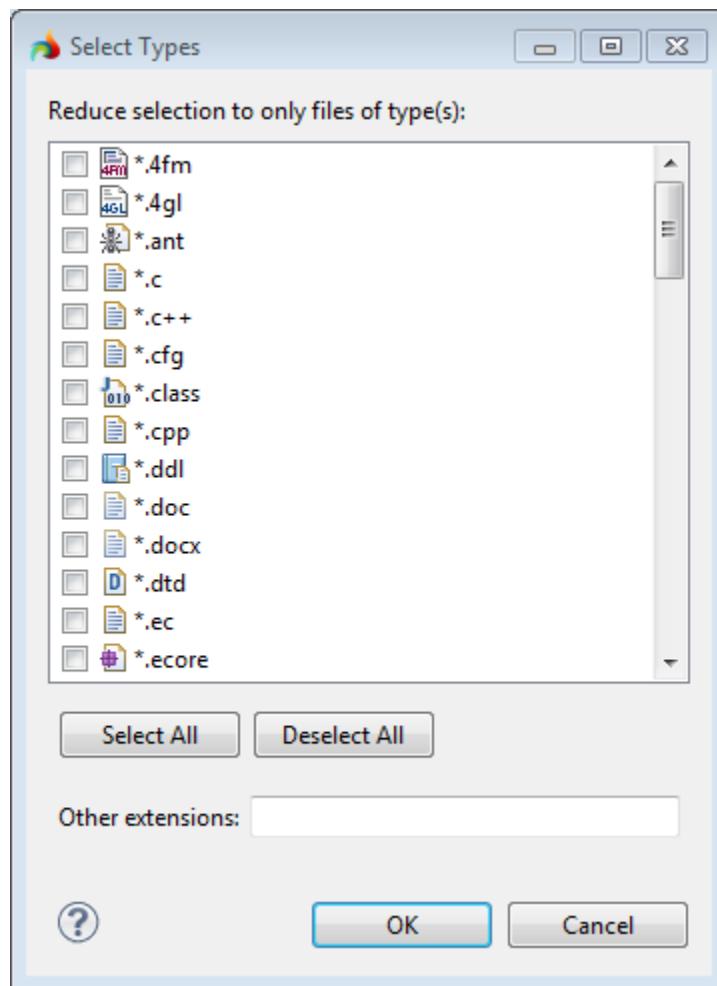




6. Make sure that the .project and .fglproject files are ticked off. These files are located in the root folder of the project and are essential if you want to import this project as a standalone project later. They must be selected automatically when you press the **Select All** button or when you select the project root folder:



7. If you want to export not a complete project, but a certain type of source files you can select these files manually. If the project you are exporting is large, then press the **Filter Types...** button and select the filetypes you want to export to quicken the process:



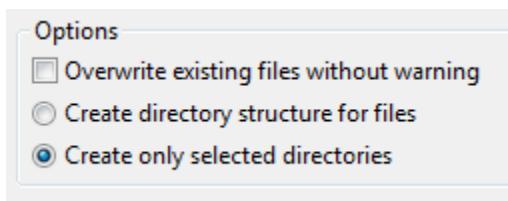
E.g. if you select the *.4fm files, all the graphical form files within the project will be selected.



Note: It is worth noting that the list of file types does not include the special file types such as .fgltarget, .project and .fglproject which are not source files as such, but which need to be exported if you export a complete project and can be manually selected. If you plan to export only a set of source files which will then be imported into another project, these files are not required.

8. Type the path to which you want to export the project or press the **Browse** button to select it:

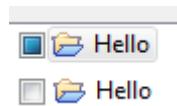
9. The Export dialog contains a number of options. If this is not the first time you are exporting the same project to the same folder, you can select the "Overwrite existing files without warning" option to replace the old files with the new ones.
10. You can now export either only the selected directories or the complete directory structure:



- a. By default, the "Create only selected directories" option is selected and only the directories selected in the left pane of the export dialog are created during the export. Those folders which are not selected are not created. If you do not select the main folder that contains the whole project, it is advisable to specify an empty folder in the "To directory" field, otherwise the files and folders of the project will be mixed with other files on your system.
- b. A folder is considered selected when its check box contains a check mark:



A folder is not selected either when it has no mark in its check box, or if it has a green square mark:



- c. If you select “Create directory structure for files” option, Lycia will export the complete folder structure of the project. However, only those files which are selected will be exported and some folders may remain empty.

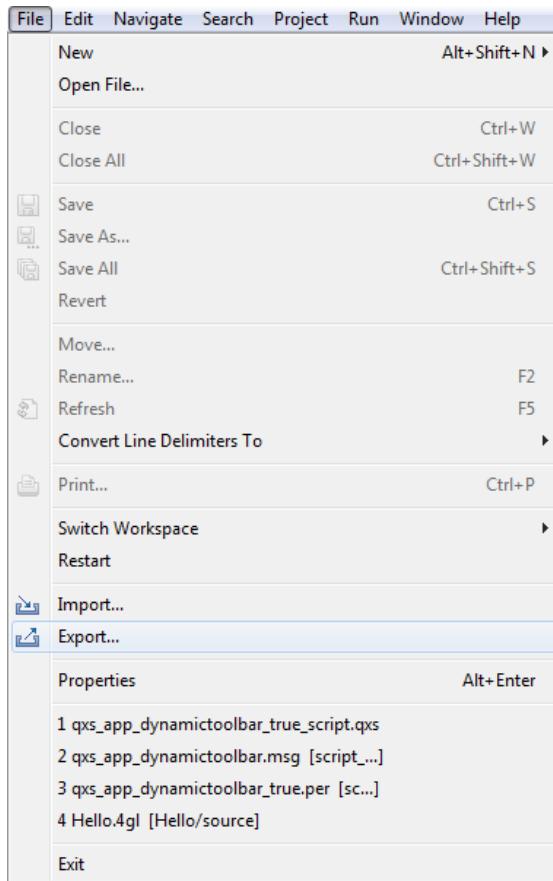
11. Press the **Finish** button to complete the export.



Exporting a project to an archive file

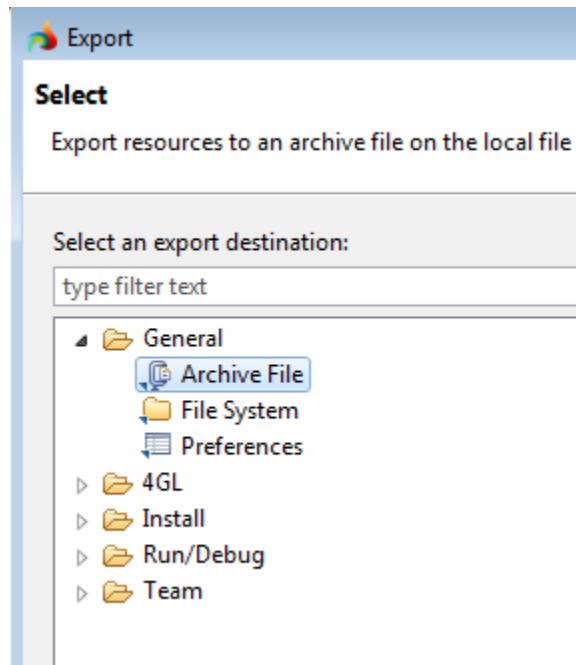
To export a project that is currently located in your Workspace to the file system, follow these steps:

1. Start the Studio
2. Select the project you wish to export and click on **File -> Export**

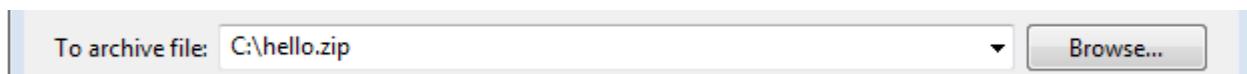




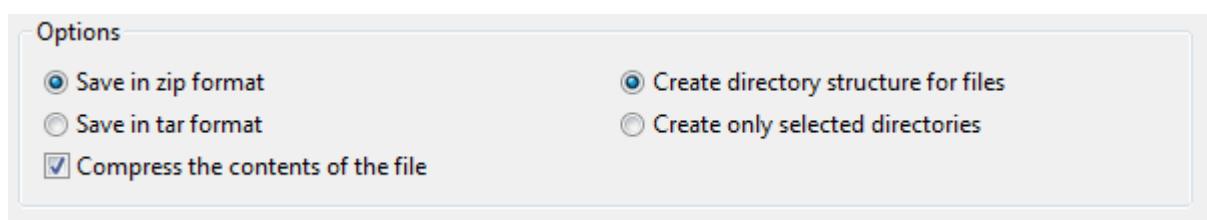
3. Select the **General -> Archive File** option and press the **Next** button:



4. Repeat steps 4-7 described in the "Exporting a project to the file system" section above.
5. Press the **Browse** button to select the export directory and type the name of the archive file to which the project will be exported:



6. You can choose the format (either .zip or .tar) of the archive file in the "Options" section of the Export dialog:



7. As you can see from the screenshot above, the contents of the archive file is compressed by default. You can uncheck the option if needed.
8. You can create either only selected folders or the complete directory structure of the project. For more details see the "Exporting a project to the file system" section, step 9:



- Create directory structure for files
- Create only selected directories

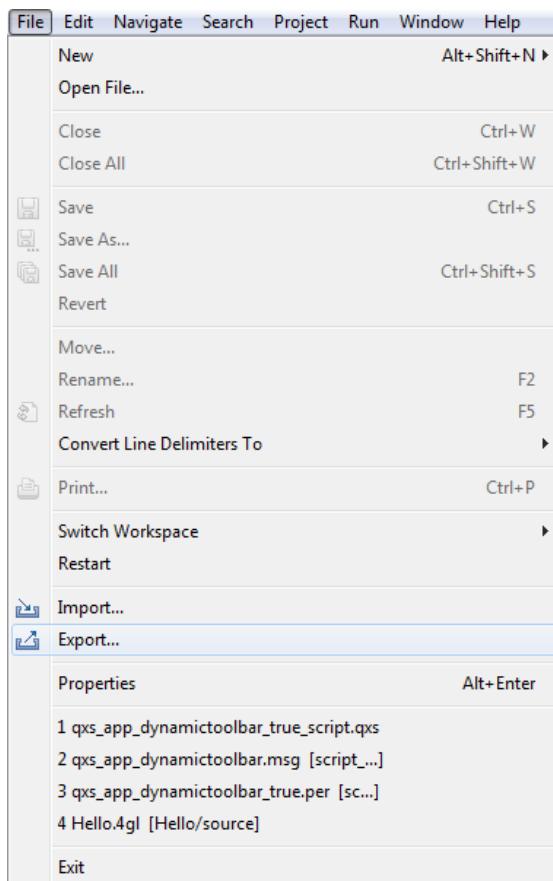
9. Press the **Finish** button to complete the export.

Exporting 4GL programs and libraries

To export a program or a library from a project into an archive file use the “4GL Programs and Libraries” export option. This option should **not** be used to export a complete project. The programs exported by means of this option must be imported into an existing project, or into a new project.

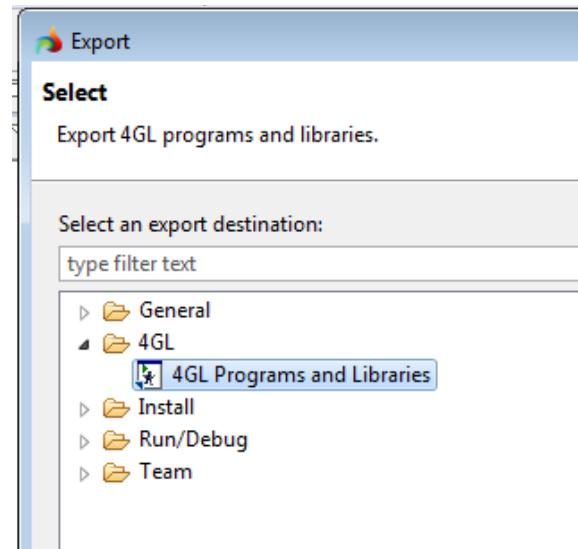
To export a program or a library, follow these steps:

1. Start the Studio
2. Select the project you want to export and click on **File -> Export**

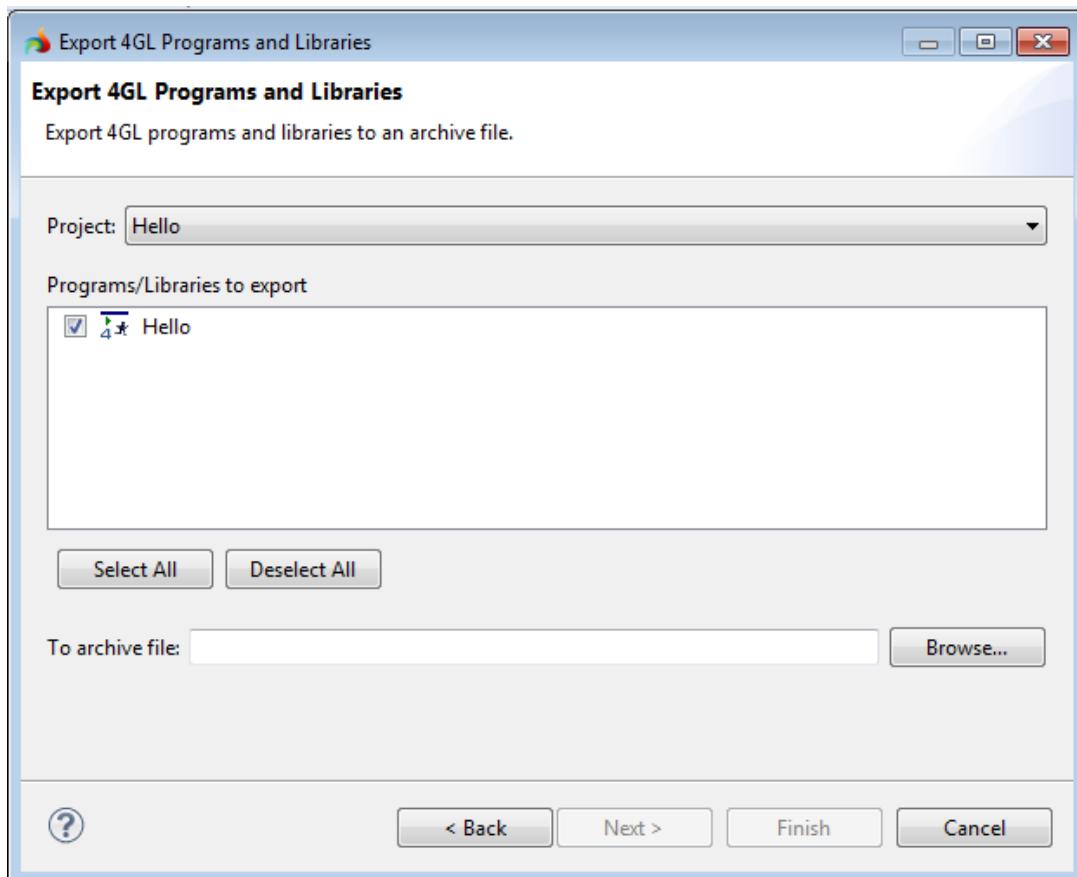




3. Select the **4GL -> 4GL Programs and Libraries** option and press the **Next** button:

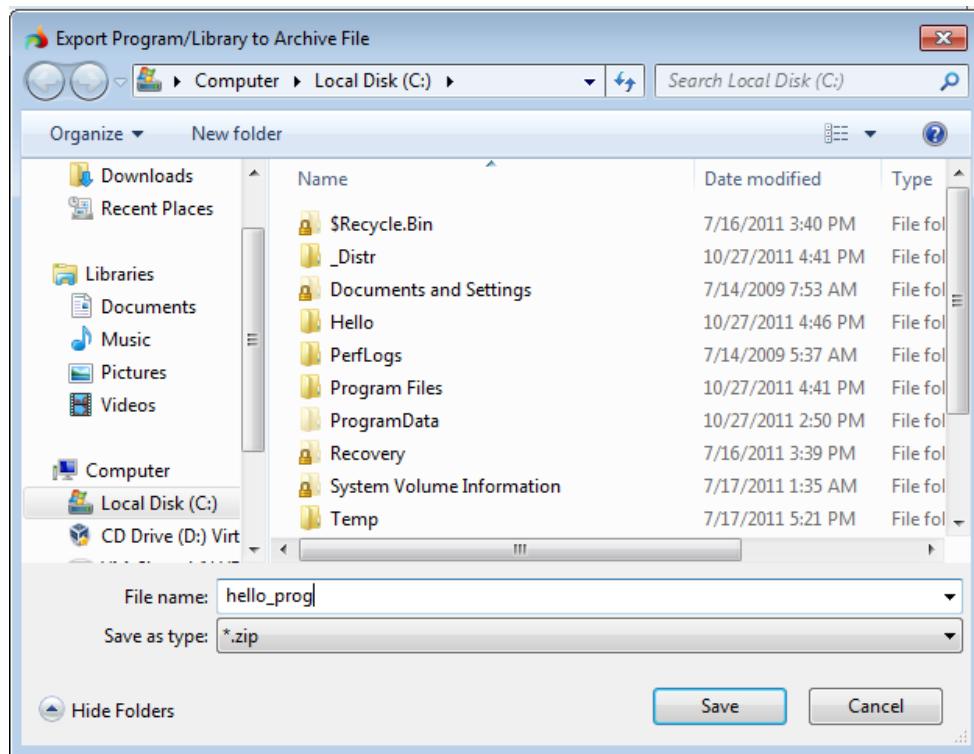


4. Select the programs and libraries you want to export:





Press the **Browse** button to select an archive file for export. You can create a new archive file or rewrite the existing one. To create a new archive file, enter the name for it in the “File name” field and press the **Save** button:



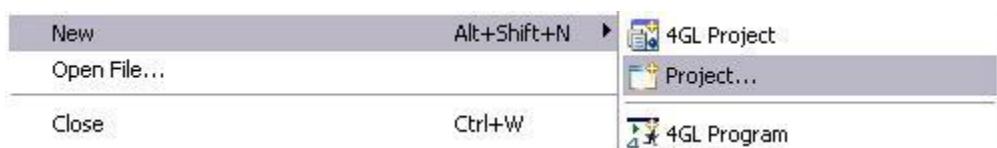
The created archive file has the .zip format by default, enter the file extension explicitly to specify another format from the formats listed in the “Save as type” field.

5. Press the **Finish** button to complete the export.

Creating a Lycia 4GL project from existing sources

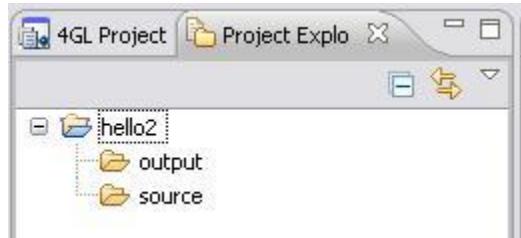
LyciaStudio allows a developer to import a project which has not been originally created in ‘HydraStudio’. If the developer has 4GL sources, the original format is not important as they can import them regardless.

1. Start the Studio
2. Create a new project from **File -> New -> 4GL Project**

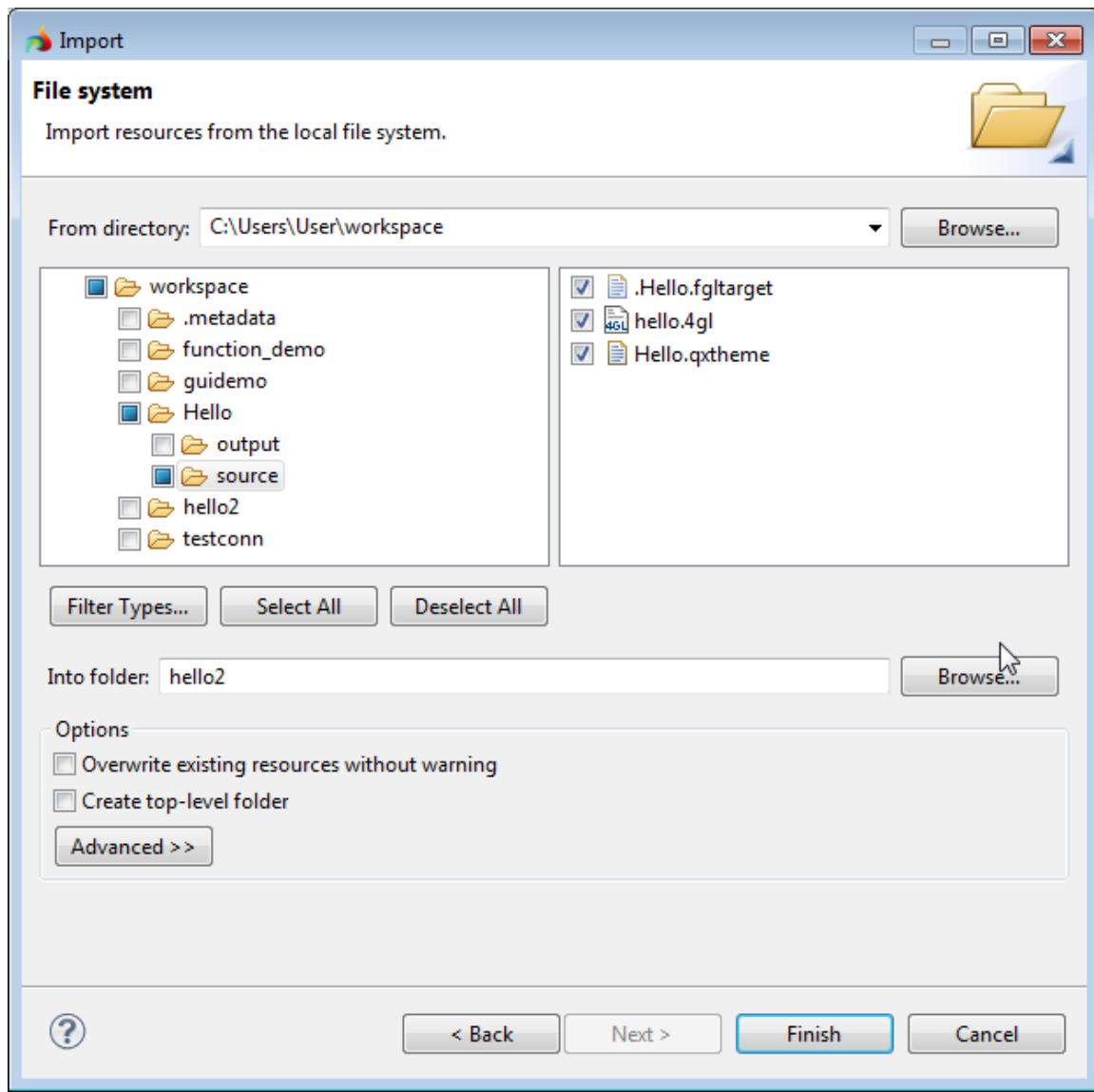




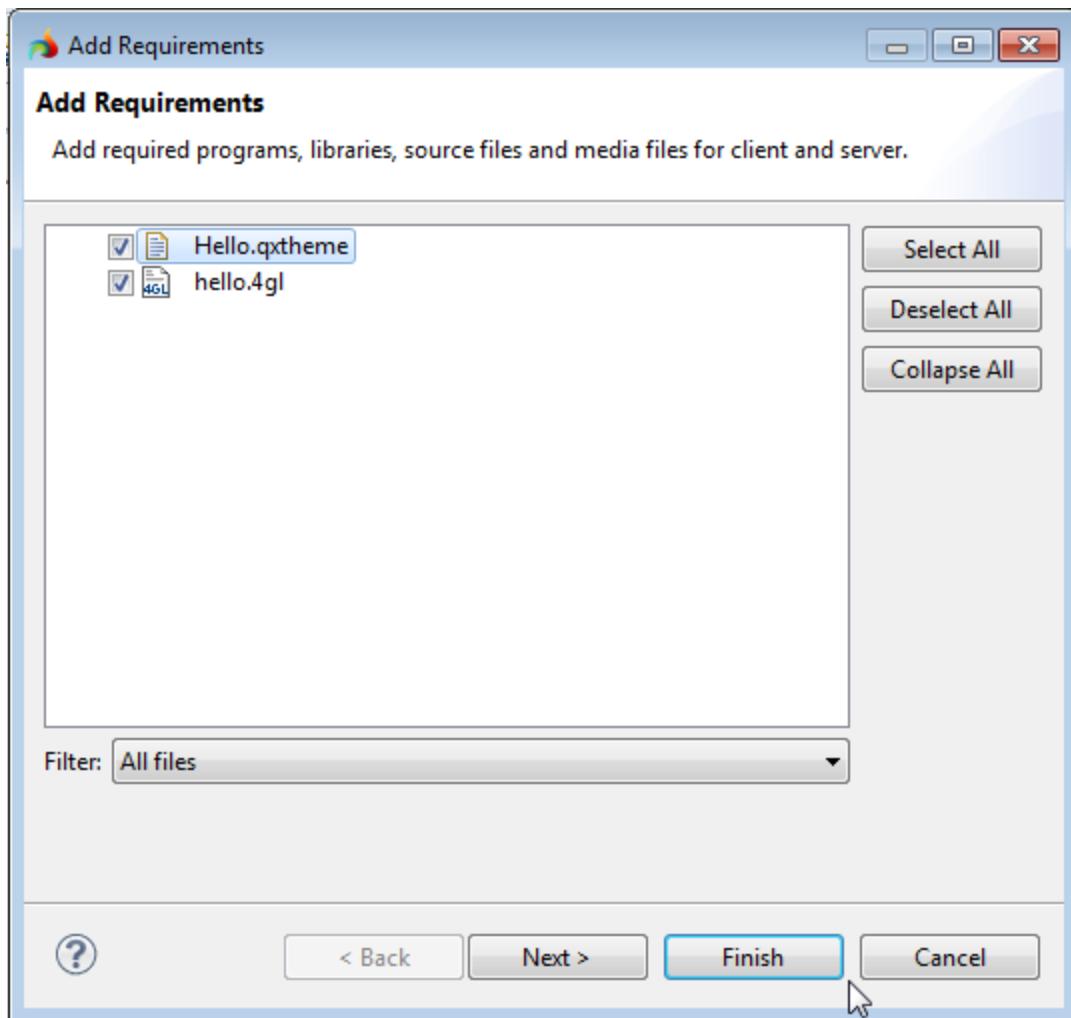
-
3. This will create a project which can be seen under the Project Explorer tab. Change views so this panel is now on top of the 4GL Project tab.



4. Right-click on the project name (*hello2* in our example)
5. Select **Import**, and then **General -> File System**
6. Add all files required. Select which files are required for the import from the right hand list, ticking each required. Next specify the "Into Directory" (if not already set). Click on Finish.



7. The Studio then takes these files and creates a project, maintaining the structure from the previous project.
8. Switch back to the 4GL Project View tab. Create a new program within this project with **File -> New -> 4GL Program**
9. Name your program, choose the same location as before for the source folder and click Next
10. Under the Add Requirements section, select all files which will be needed by this project.



As you can see from above, we have selected a spreadsheet and an icon file to be added. This will then set a dependency between this project and the files selected.

11. Add any External Libraries if required on the following screen.

12. Click Finish.



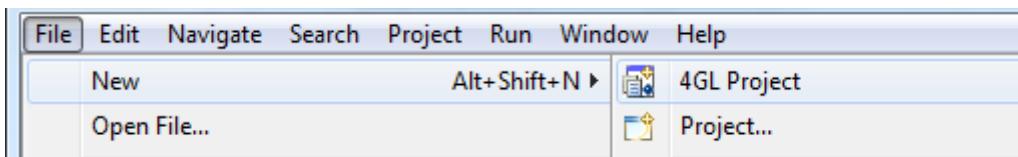
Creating a new 4GL project

To demonstrate how to make a new 4GL project, we are going to use a simple example called *hello.4gl*, which will print to the screen "Hello World" and run using our thin client, LyciaDesktop.

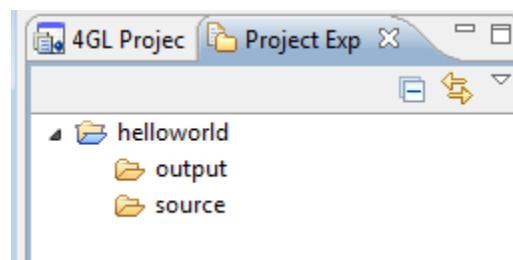
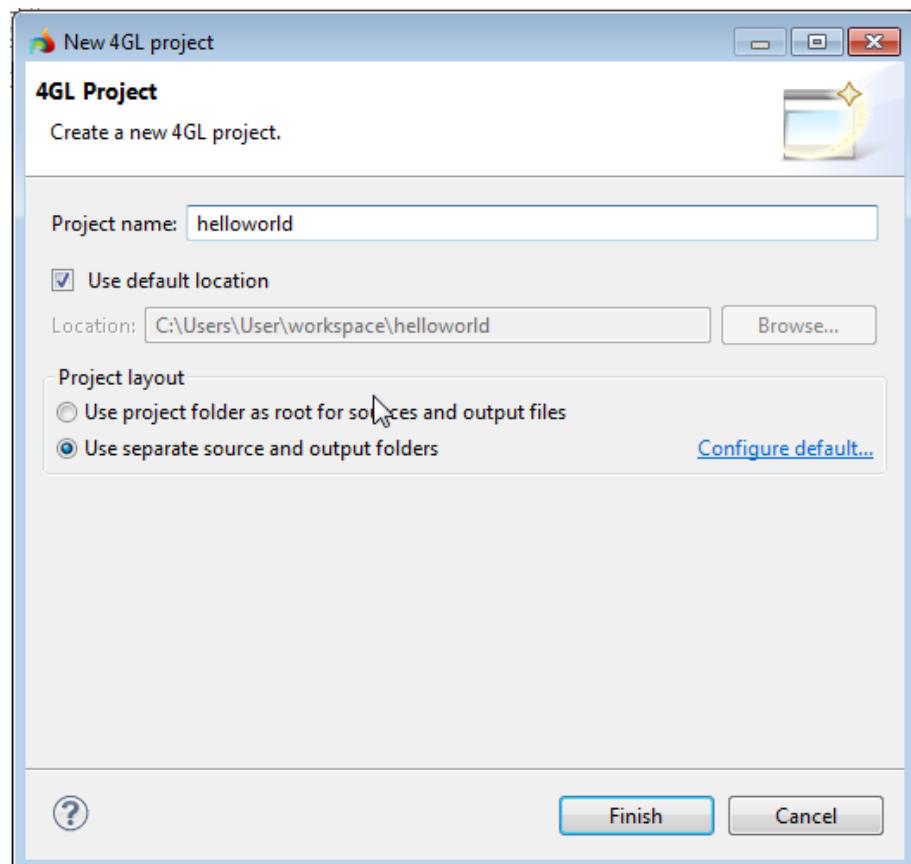


Please note, within the Studio there is more than one method to adding new programs/files to a project. **This method takes you through the main menu system, but simply right-clicking on a program will also offer the same functionality.** This demonstration will show you both screen shots, first the menu-based option, followed by the right-click option. Please use whichever you feel most comfortable with.

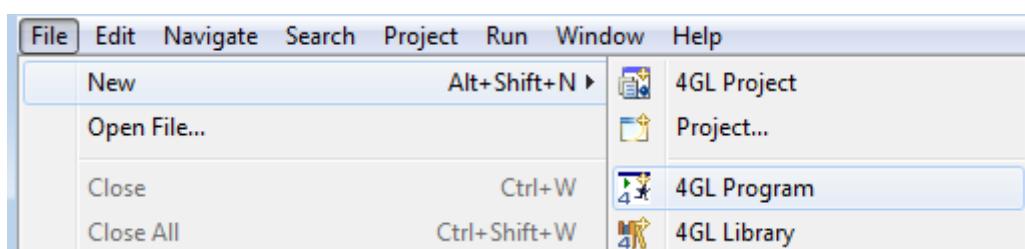
1. Start LyciaStudio
2. Click on **File -> New -> 4GL Project**



3. Give the project the name *helloworld*, leave the default options and click Finish

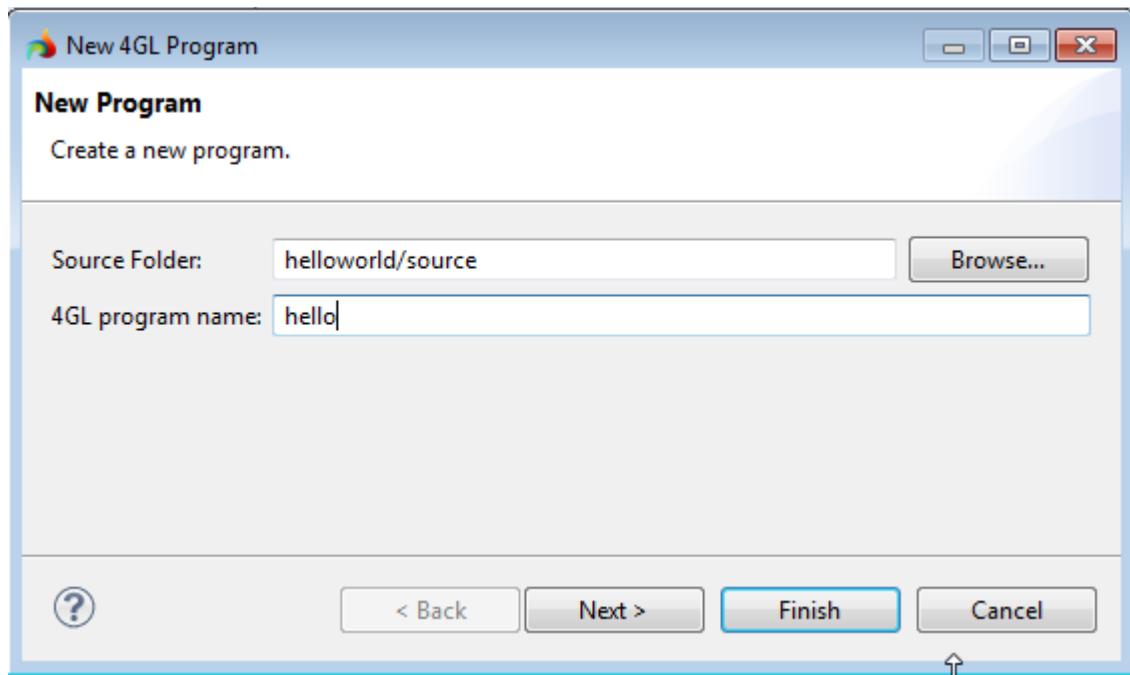


4. Click on **File -> New -> 4GL Program**





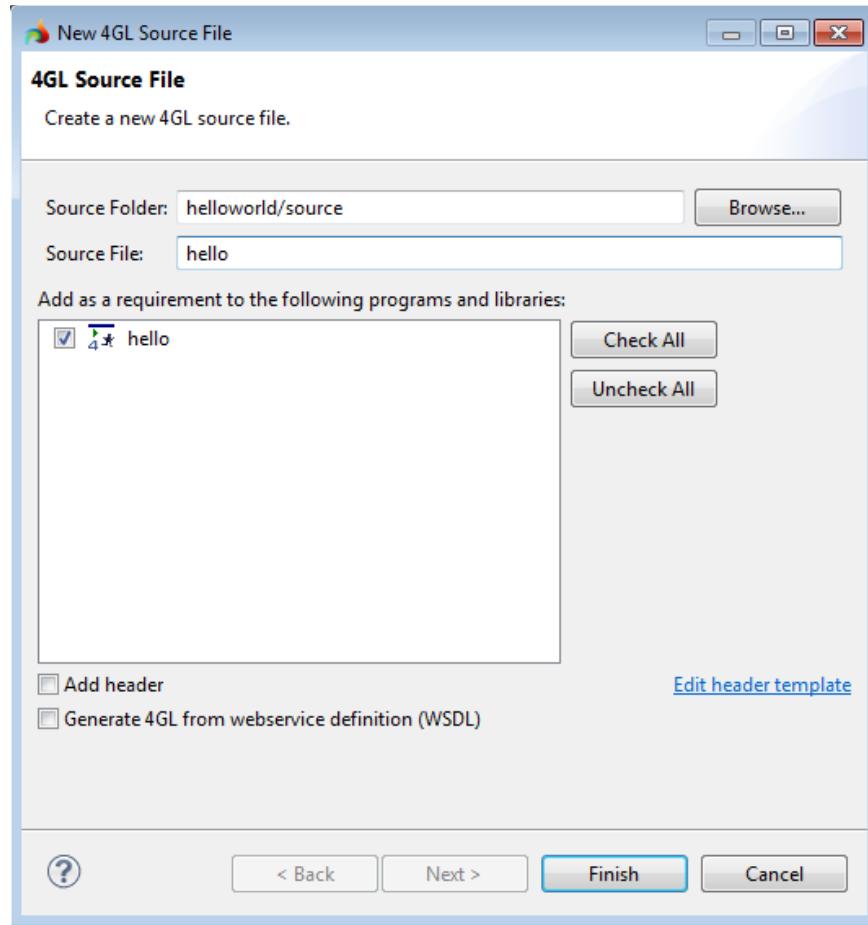
5. Give the program the name *Hello*, leave the default options as they are and click Finish



6. Click **File -> New -> 4GL Source File**



7. Give the source file the name *Hello*, leave the default options and click Finish

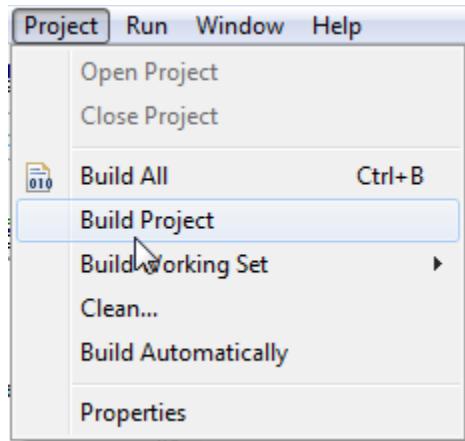


8. Add the following code to the source file and save:

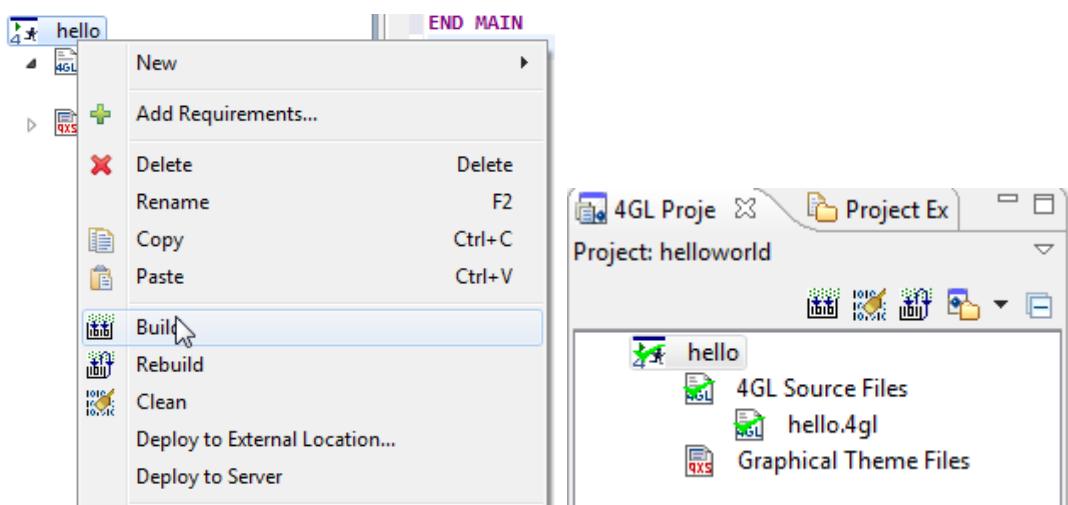
```
MAIN
DISPLAY "Hello World" AT 2,2
SLEEP 3
END MAIN
```



9. Select **Project -> Build Project**



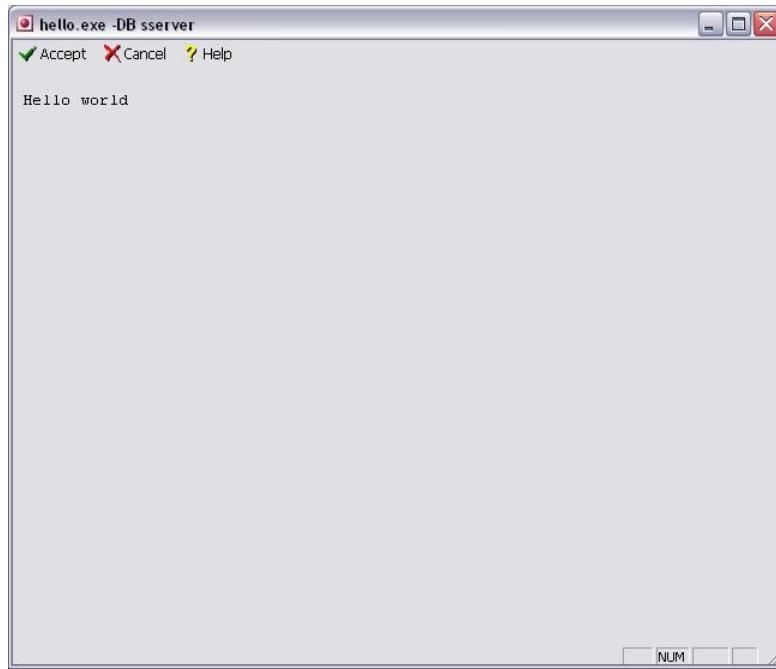
Or left-click the application:



10. Click on the program *Hello* and select **Run -> Run As -> with LyciaDesktop** from the menu or from the right-click menu of the program

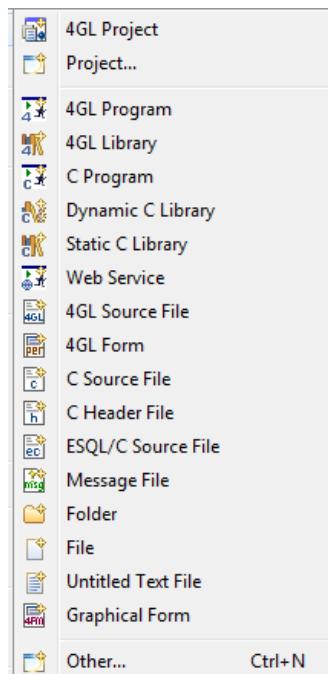


Running results in the following LyciaDesktop window:



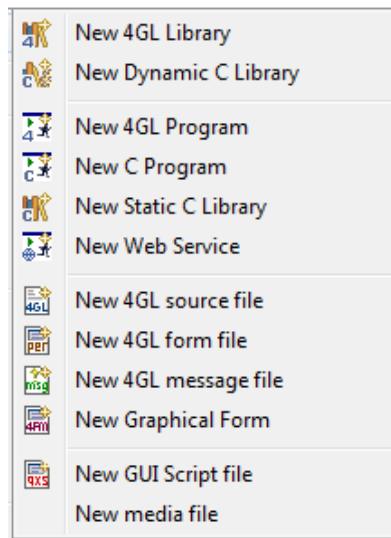
Creating new files

New files can be added to existing projects or to new ones using the menu system. There are many different types of files which can be created, as shown below, which can be accessed either from the main **File->New** menu, or by right-clicking on a project.





Main New File Menu

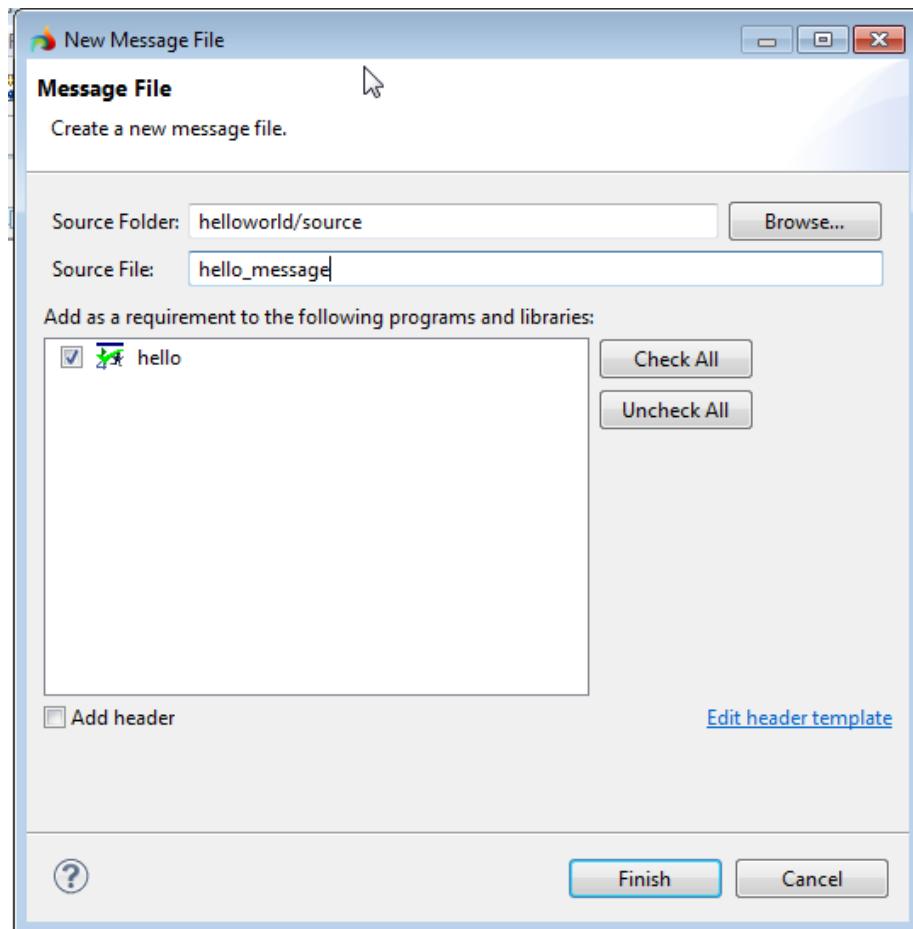


Right Click New File Menu

Creating source files

Most of the files that can be included into a project can be created by means of LyciaStudio. To create 4GL source files, form files (both in graphical and text formats), script files, message files, C files, and text files (.txt), you need to select the corresponding option from the New menu. All these files are created in the same way, so the process of file creation will be explained by providing an example of a message file:

1. Select the New 4GL Message File option from the New File Menu
2. In the New File dialog, enter the name of the file. Note that no file extension is required for any of the above listed file types:

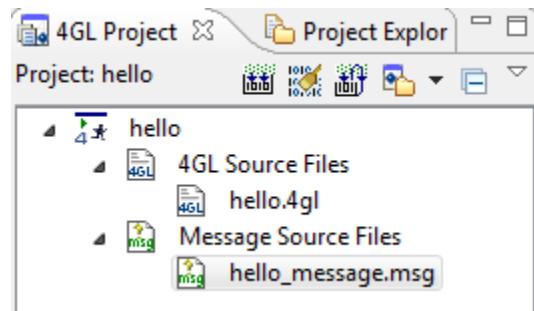


Also note that you can choose a project folder where you want the file to be created in. The location of the file implies some restrictions which must be taken into account:

- a. By default, the file is placed in the “source” folder. The default name for the future projects can be changed using the **Window -> Preferences -> 4GL -> Build Path** preferences page.
- b. You can use the **Browse...** button to select another folder. However, you can only select folders which are nested within the “source” folder.
- c. You must not rename the “source” folder of an existing project, otherwise you will be unable to create new files within the project. You can set another existing folder as the source folder using the **Project ->Properties -> 4GL Project** properties page.
- d. You must not delete the “source” folder, otherwise you will be unable to create new files within the project and all the existing source files will be removed from the project



3. If you want this file to be required by the program, select this program from the list
4. Press the Finish button.
5. A message file will be created and placed in the section for Message Source Files. It will also be opened in the editor area.

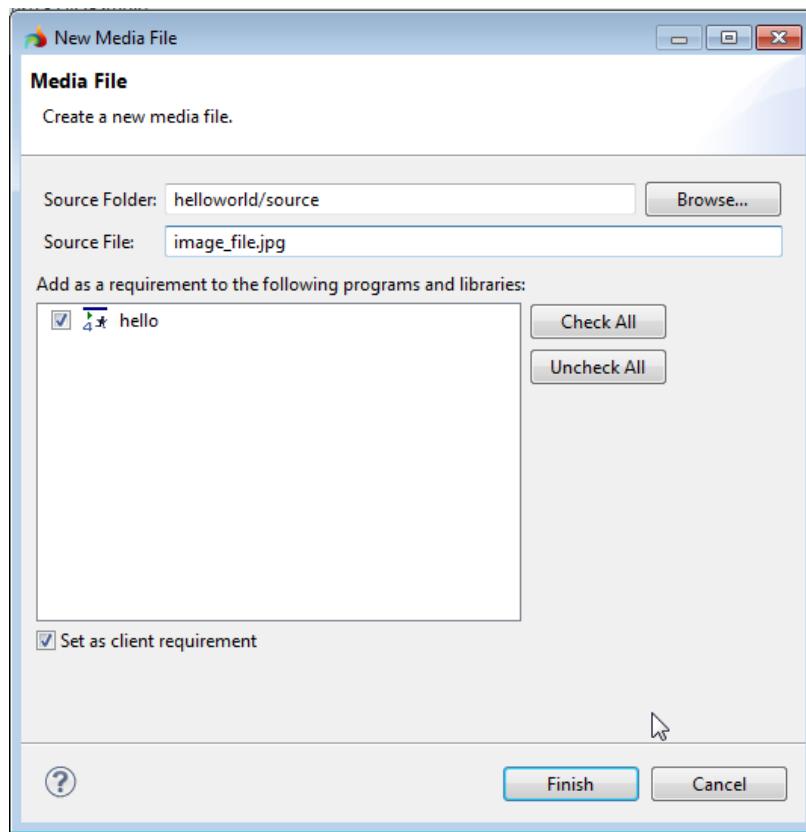


Creating media files

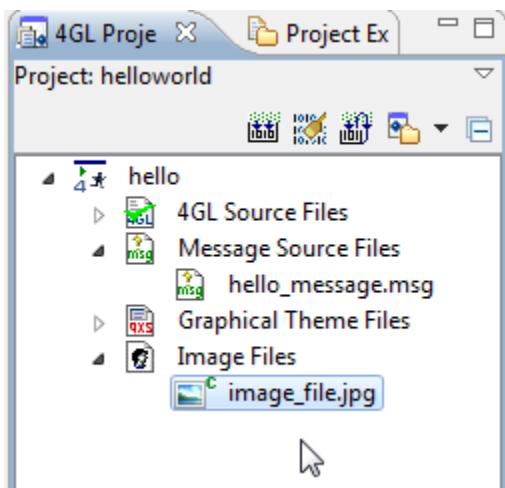
Lycia also supports a number of media files which can also be created by means of the Studio though the process of their creation differs a bit from the process of creation of the non-media files. To create an image file follow these steps:

Select the New Media File option from the New File Menu. This option is used for creating image files, text files and other types of media files.

In the New File dialog, type the name of the file. Note that you have to enter the extension of a file explicitly, otherwise the Finish button will remain disabled. Lycia does not know what kind of media file you want to create unless the extension is given. Enter the name of a file and the extension (e.g. image_file.jpg)



1. Press the **Finish** button.
2. The media file will appear in the 4GL Project view in the section for Image files:



3. The newly created file will be opened for edition in an external editor registered with the system for the JPEG file type.



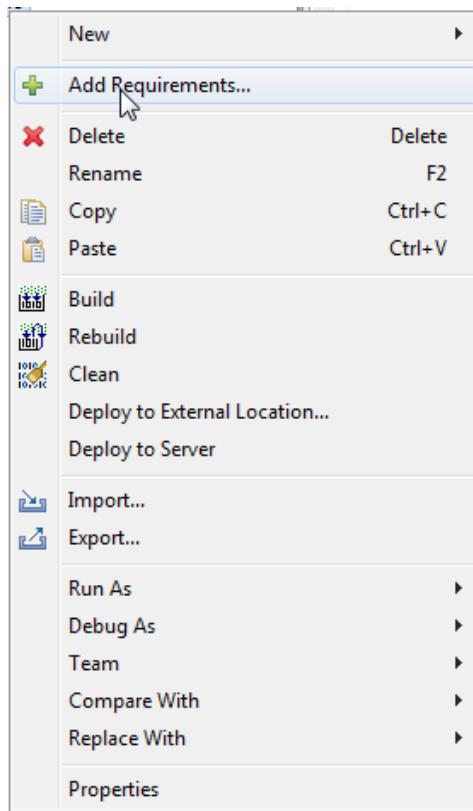
LyciaStudio does not have an internal image editor. All the media files are opened in the associated external editors. The JPEG file type can be associated with a viewer which has no editing options. In such case you need to change the default editor associated with this file type or open the file in another external editor. For the information about opening a file in another editor see the "Editors" part of the "Perspectives and Views" section of this guide.

Adding imported files to project requirements

Files are sometimes added to a project and not created in it. To add a file to a project, use the Import option (see the "Importing files, folders, programs and libraries" section above).

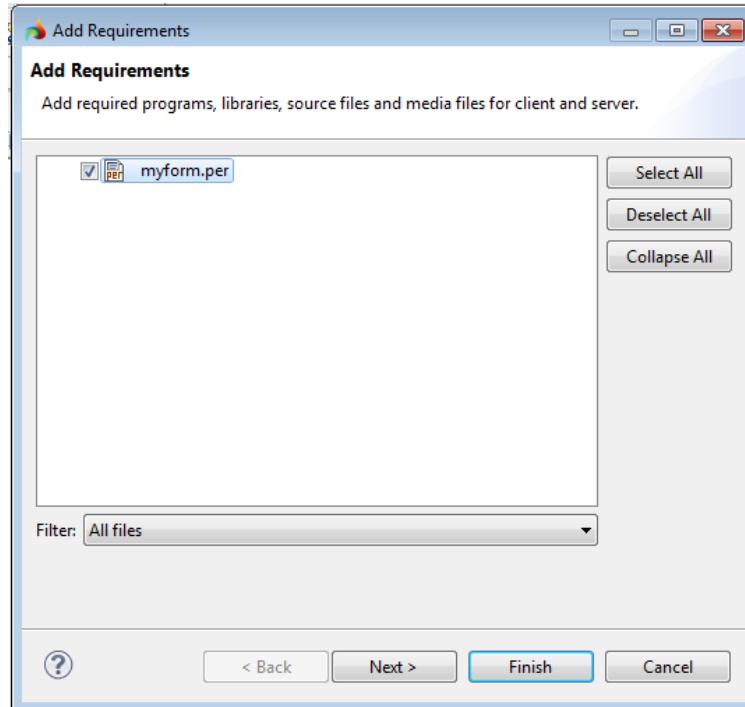
An imported file (both media and non-media) needs to be added as a project requirement. When it is imported, it appears in the Project Explorer view, but it does not appear in the 4GL Project view, does not influence the process of building and running the project and it cannot be compiled until it is added to the project requirements. To add a file to the project requirements follow these steps:

1. Right-click on a project in the 4GL Project view and select the Add Requirements option:

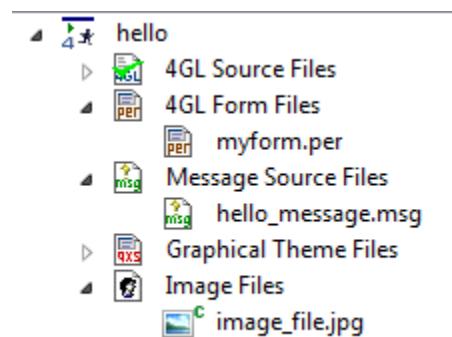




2. Select the file you want to add to the requirements form the list and press Finish. Note that only imported files will appear in the list.



3. The selected file will appear in the corresponding section of the 4GL Project view:



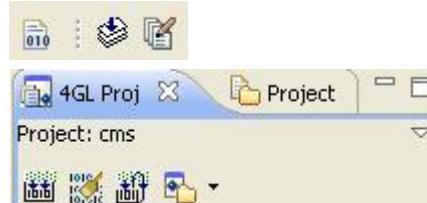


Building and compilation

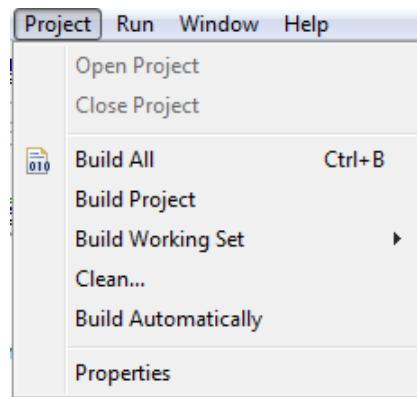
LyciaStudio offers several ways in which the resources in the Workbench can be built or compiled. Building and compilation operate on the resources that have changed since the last build or compilation. You can use the clean tool or the rebuild tool, if you want to rebuild the resources which have already been built before.

The tools used for building and compilation can be found:

- on the main LyciaStudio toolbar:



- on the 4GL Project View toolbar:
- in the **Project** option of the main menu:

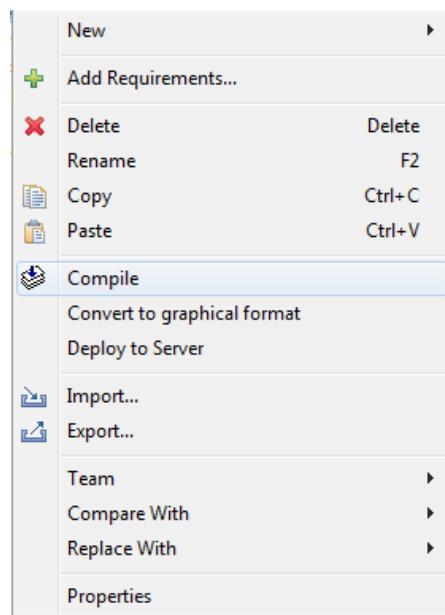


- in the context menus called by right-clicking on a source-file or a program.

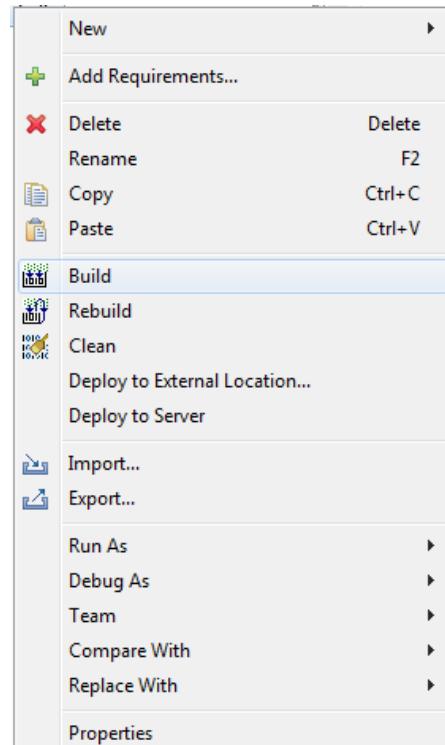
Building tools

There are several tools which are used to build or compile the resources:

- To compile a selected source file use the Compile () button on the LyciaStudio main toolbar. This option can also be found in the context menu of the source file: right-click the source file in the editor or in the 4GL Project View and select the **Compile** option:



- To build a single program or library, right-click it in the 4GL Project View and select the **Build** option:

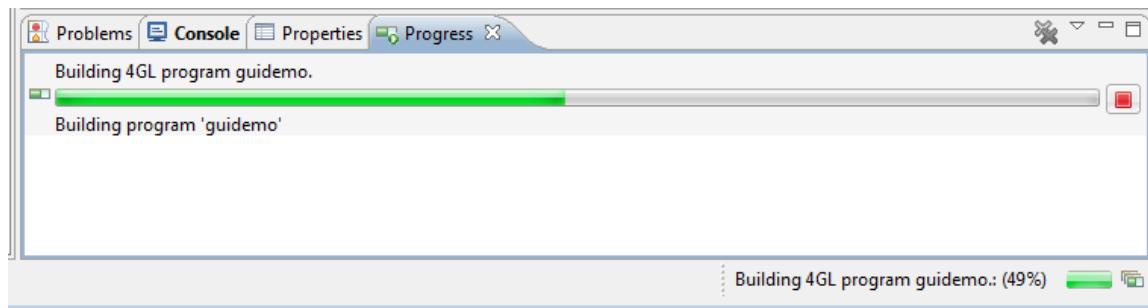


- To build a complete project, press the Build () button on the 4GL Project View toolbar, or select the **Project -> Build Project** option from the main menu.



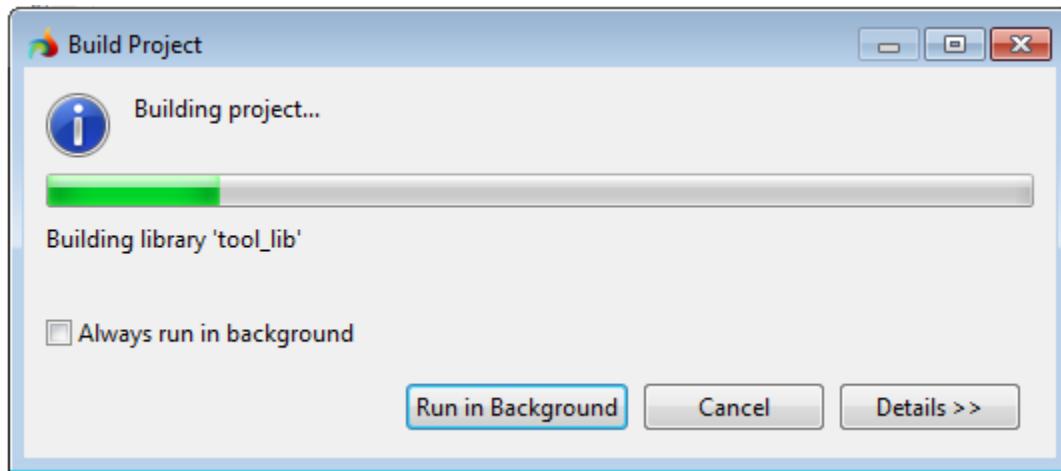
- To build all the projects present in the workbench, press the Build All () button on the main toolbar or select the **Project -> Build All** option. All the projects will be built, if this option is activated. The order in which the projects get built can be set in the **Preferences -> General -> Workspace -> Build Order** preferences page. For more details see the "Preferences Menu" chapter.

If a project contains many source files, it may take Lycia some time to build them all. You can observe the progress in the "Progress" View which can be opened by pressing its icon in the bottom right –hand corner of the Workbench:



Here you can interrupt the building by pressing the red "stop" button (). This view also reflects the progress of cleaning operations.

If you use the **Build All** option, **Build Working Set** option or any other option which builds several projects together, you will be presented with a Build dialog which shows the progress of the building and what project is now being built:



Press the **Run in Background** button to hide the dialog. When the dialog is hidden you can observe the build progress in the "Progress" View. You can select the "Always run in background" option to hide this dialog every time a build of multiple projects is performed. This dialog also reflects the progress of cleaning operations.

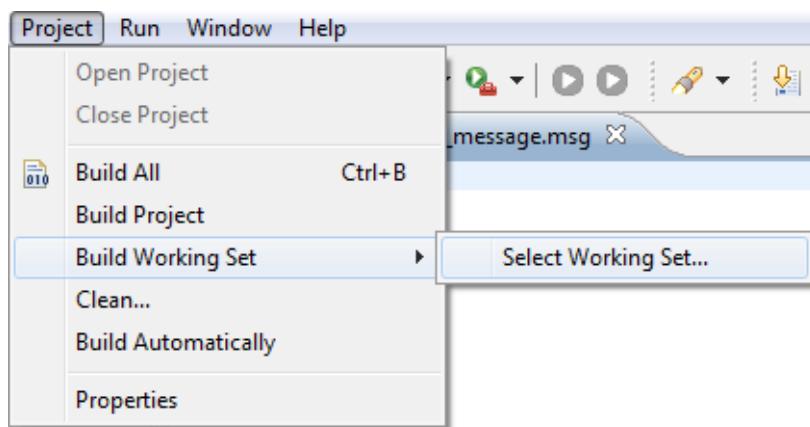


Building a working set

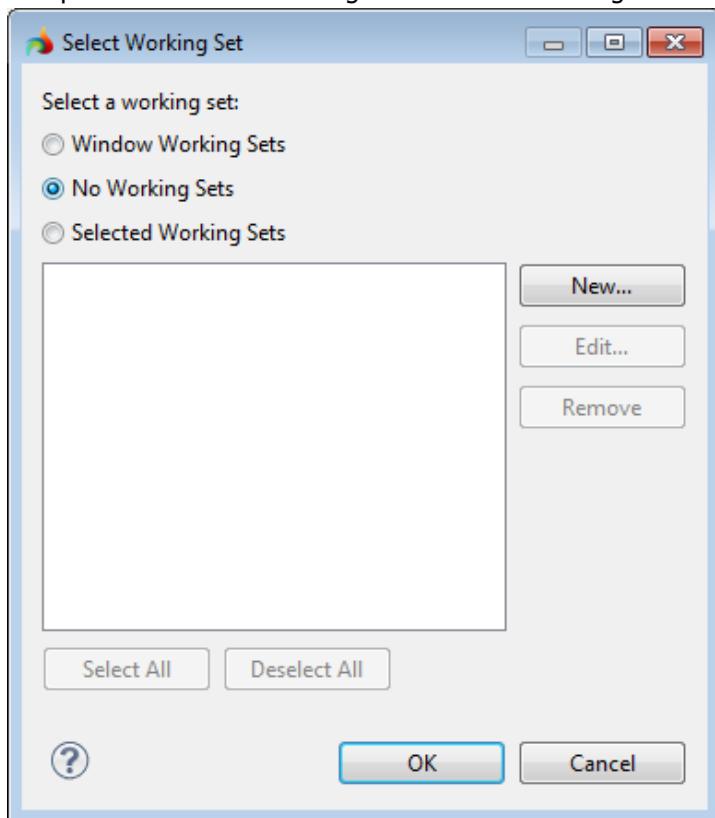
A number of projects united in one group that can be manipulated simultaneously are called a working set. To build a working set select the **Project -> Build Working Set** option and choose the working set you want to build.

If no working set is currently available, you can create one by the following method:

1. Select the **Project -> Build Working Set ->Select Working Set** menu option:

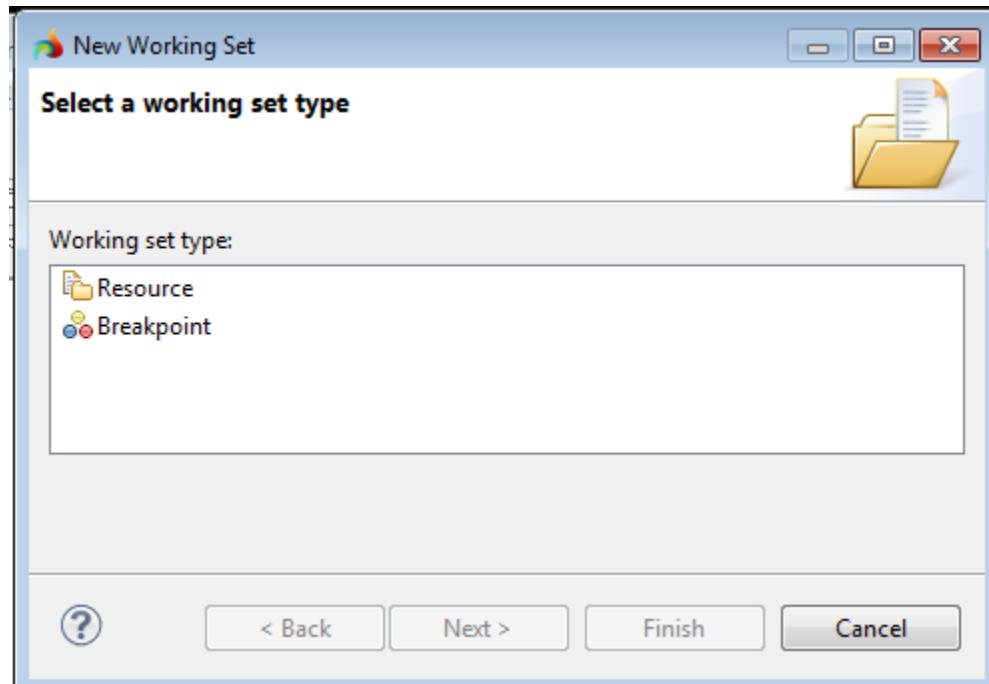


2. You will be presented with the dialog that is used to manage working sets:



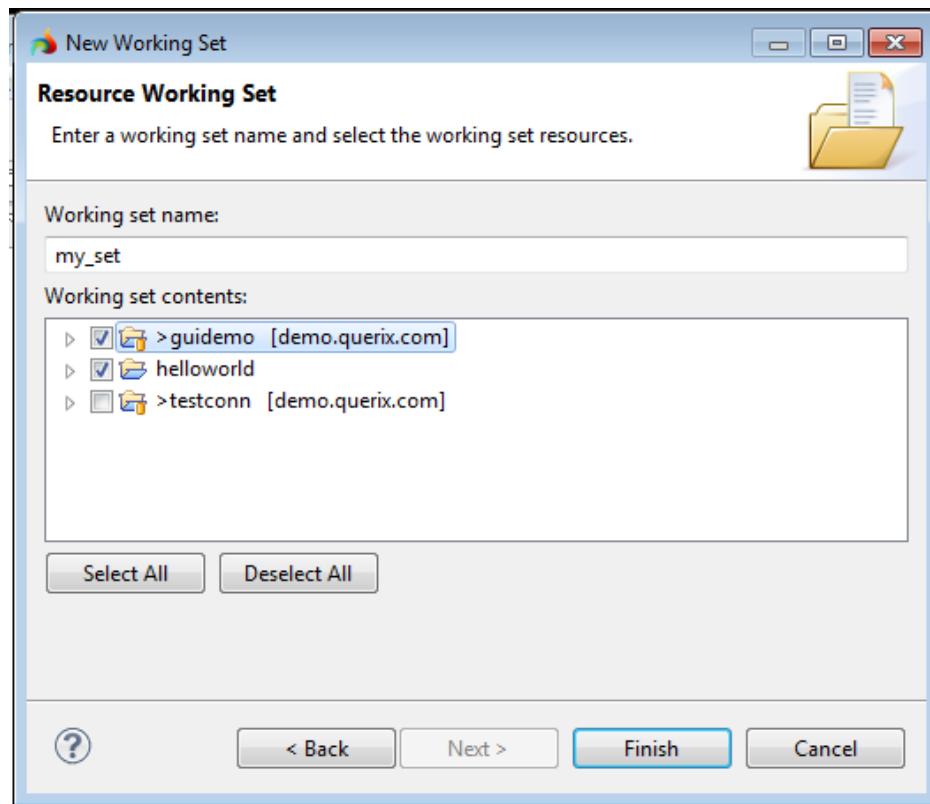


3. Press the **New** button to create a new working set, select the “Resource” option in the dialog that will appear and press the **Next** button:





4. In the next dialog, you can select the projects you wish to be included into the working set. Enter the name for the working set in the corresponding field:

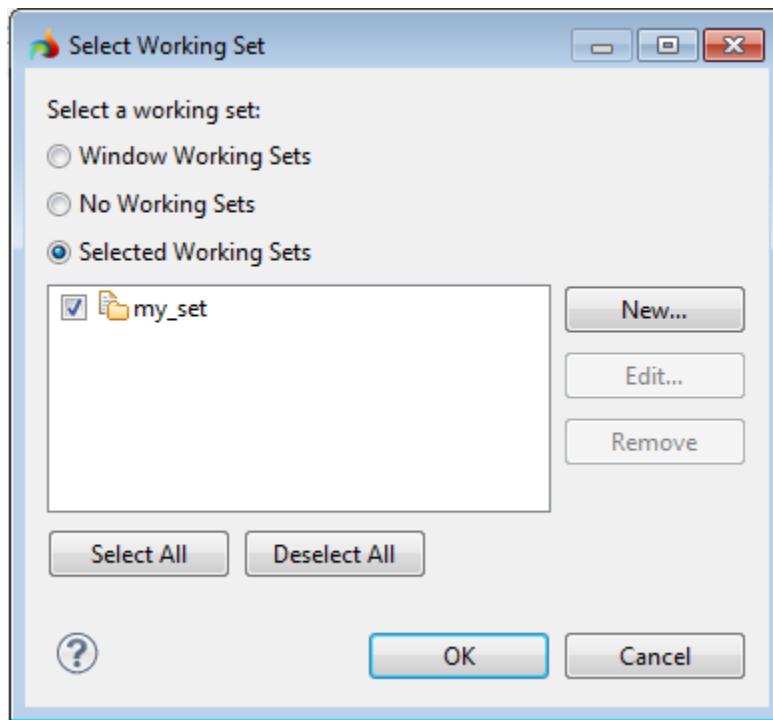


A working set does not necessarily include complete projects; it can include separate folders or source files which you want to be compiled together. To select the files and folders, unfold the project tree by pressing the plus button.

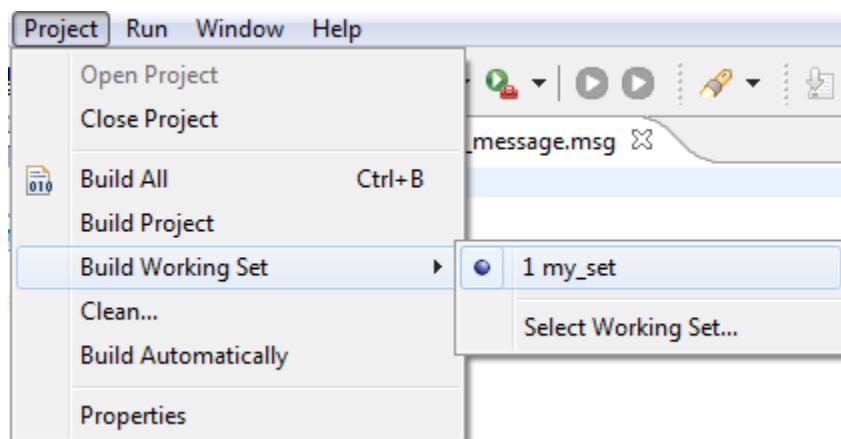
5. Press the **Finish** button after you have selected the resources to be included into the working set.



6. You will go back to the working set selection dialog. Select the newly created working set in the list and press the **Finish** button:



7. The building of the selected resources will start immediately.
8. Now you can compile all the resources included into the working set at any time by using the **Project -> Build Working Set** option and selecting the working set from the list of the available working sets:





Cleaning and rebuilding

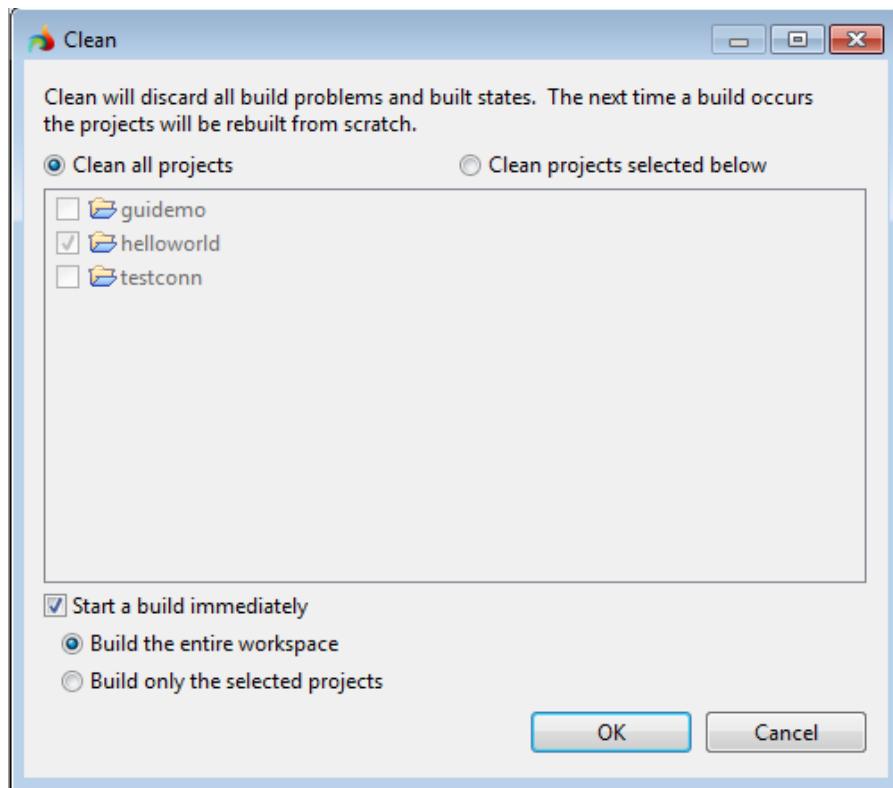
When a resource is cleaned, the compilation markers of both successful and unsuccessful compilations are removed from within the 4GL Project View, and the files resulting from the compilation are removed from the output folder of the corresponding project.

- To clean a program, right-click it in the 4GL Project View and select the **Clean** option:

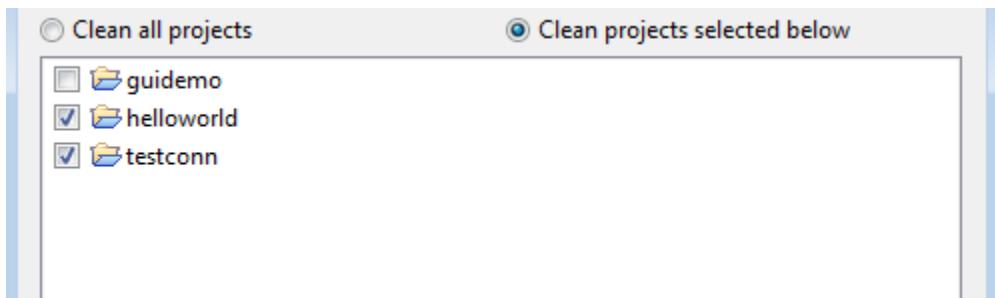


- To clean the whole project, press the Clean button () on the 4GL Project View toolbar
- To open a Clean dialog which allows you to adjust the cleaning procedure and specify which projects should be cleaned, select the **Project -> Clean...** option:

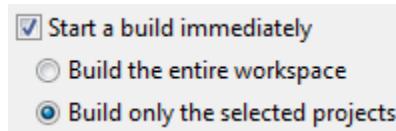
1. A Clean dialog will be opened as shown below:



2. By default, all the projects in the Workbench will be cleaned. Select the "Clean projects selected below" option and select the projects you want to clean, if you do not wish to clean all the projects:



3. By default, a build is started automatically after the projects are cleaned with the help of the Clean dialog. You can choose whether all the projects in the workspace should be built after the cleaning or just the selected ones:



If you do not want the projects to be built after cleaning, deselect the "Start a build immediately" option.

The cleaning may be required, if you want to build a project once more and do not want the compiler to skip the previously compiled source files. However, you can use the rebuilding instead of cleaning a project and then building it again.

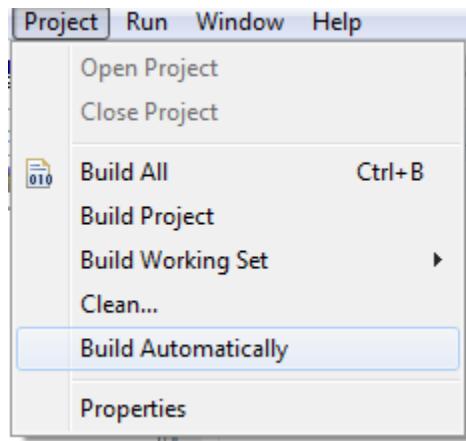
The rebuild option combines the build and clean options: first it cleans the resources and then builds them again.

- To rebuild a program right-click it in the 4GL Project View and select the **Rebuild** option from the context menu.
- To rebuild the whole project press the Rebuild button () on the 4GL Project View toolbar.

Building automatically

LyciaStudio offers a tool for performing build automatically each time a source file is saved, cleaned, or imported into the workspace. All the source files in the Workbench that require building are built immediately after the **Build Automatically** option is activated.

Building of a project can be set to be automatic by selecting **Project->Build Automatically**.



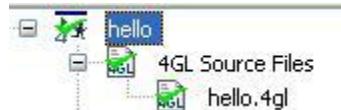
It can be also enabled by pressing the Build Automatically () button on the main toolbar.

The resources are built automatically when they are saved, if the “Build Automatically” option is activated. To save the source files automatically before each manual build use the **Preferences ->General -> Workspace** preferences page to select the “Save automatically before build” option. This option is selected by default.

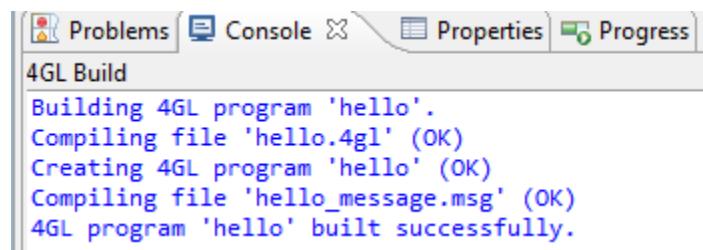
Building results and compile-time errors

There are two results of building process: it can be either successful, or unsuccessful due to compile-time errors.

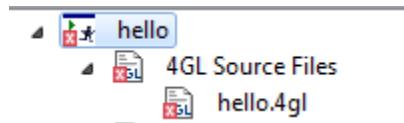
A successful build results in the green check mark on the resource in the 4GL Project View:



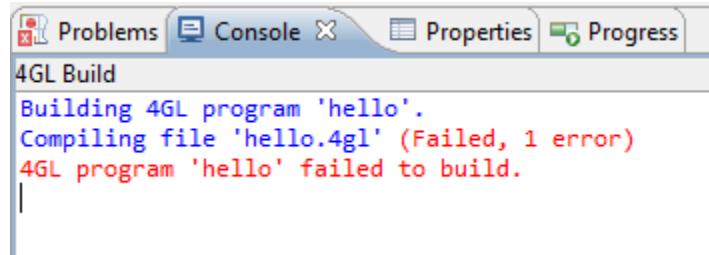
In the “Console” View, the messages indicating the successful build will appear:



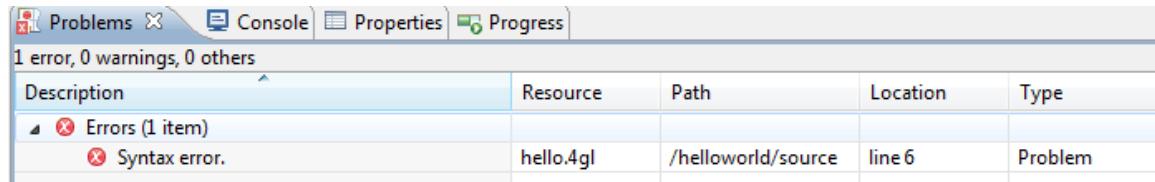
If a compile-time error occurs, the building is unsuccessful and a red cross appears on the resource in the 4GL Project View:



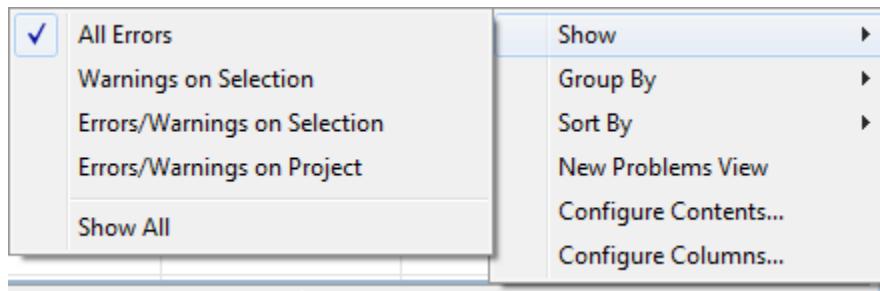
The corresponding messages will appear in the “Console” View:



To see what sort of error has prevented a file or a program from being compiled, open the “Problems” View:



Here you can see the source file which contains the error and the exact location of the error and the cause of the error. By default, all the errors and problems present in the Studio are displayed in this view. You can filter the errors using the drop-down menu in the top right corner of the view, where you can select the content to be displayed in the view:



- **All Errors** option displays all errors and warnings present in the workspace
- **Warnings on Selection** displays only warnings for the selected resource (file or program)
- **Errors/Warnings on Selection** displays both warnings and errors only for the selected resource (file or program)



- **Errors/Warnings on Project** displays all warnings and errors for the project selected in the Project Explorer view

You can also group the errors by severity or type or sort them by the description, type, location, etc. The **Configure Contents...** option allows you to adjust the view even more. For example you can adjust the view so that it would display only errors which have certain characters or words in their description. Here you can also change the scope of the displayed errors more freely.

To jump to the error, double-click the error in the errors list. The file with the error will be opened and brought to the top. The line with the error will be highlighted as shown below:

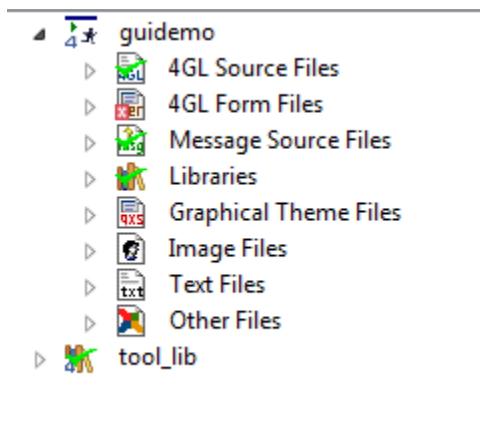
The screenshot shows a code editor window with two tabs: 'hello.4gl' and 'hello_message.msg'. The 'hello.4gl' tab is active and contains the following code:

```
MAIN
DISPLAY "Hello World" AT 2,
CALL fal_getkey()
END MAIN
```

A red error marker (a small red circle with a white cross) is positioned next to the word 'CALL' on the second line. The entire line 'CALL fal_getkey()' is highlighted in blue, indicating it is the current error being viewed.

If you hover the cursor over the error marker on the vertical ruler, you will also see a pop-up message that contains the description of the error.

The build typically fails and a program has a red cross on it if at least one .4gl source file fails to compile. However if a form file fails to compile, a program may still have a green check mark on it and it will be possible to launch said program. However, the runtime errors will occur when 4GL tries to open the form files which are not compiled and the program can crash.



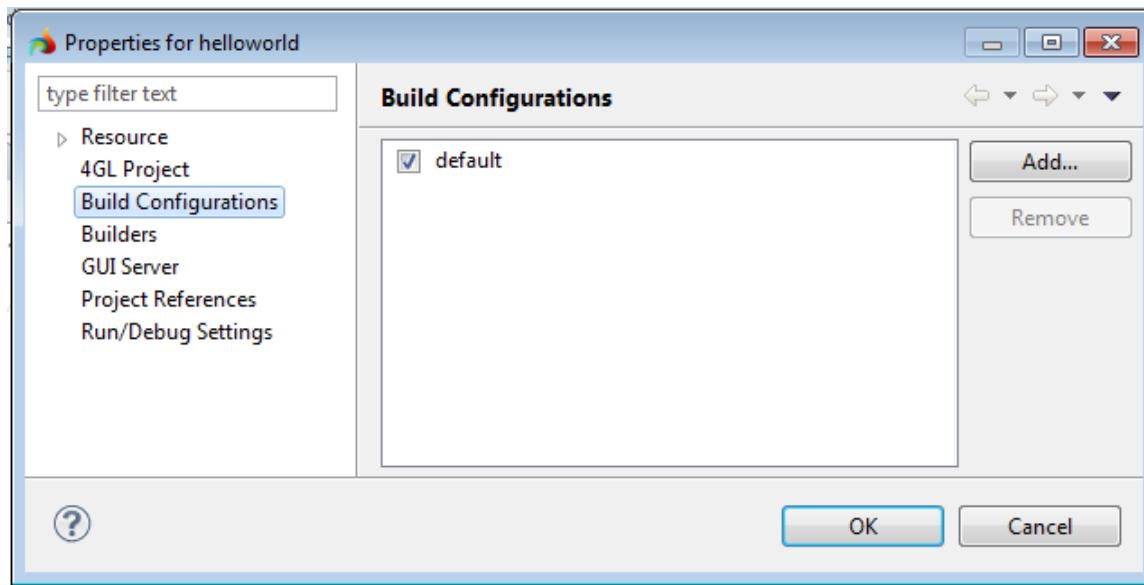
If a form file contained in a library fails to compile, the library will have an exclamation mark in the 4GL Project View as in the picture above. It will be possible to run the program to which such library is linked, but runtime errors will occur if the program tries to address the form files of the library which are not compiled. A compile-time error in any .4gl source file either in a linked library or in the program will make it impossible to launch a program.



Build Configurations

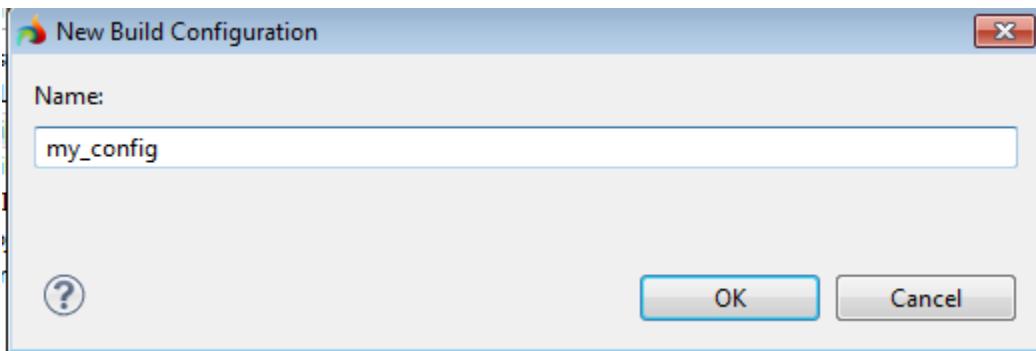
One project can have more than one build configuration. All projects have at least one build configuration which is the default build configuration, but you can add more build configurations in the following way:

1. Select the "helloworld" project you have created
2. Go to **Project -> Properties** main menu option
3. Select the **Build Configurations** option from the list on the left
4. You will see the list of the available build configurations which will include only the default configuration:



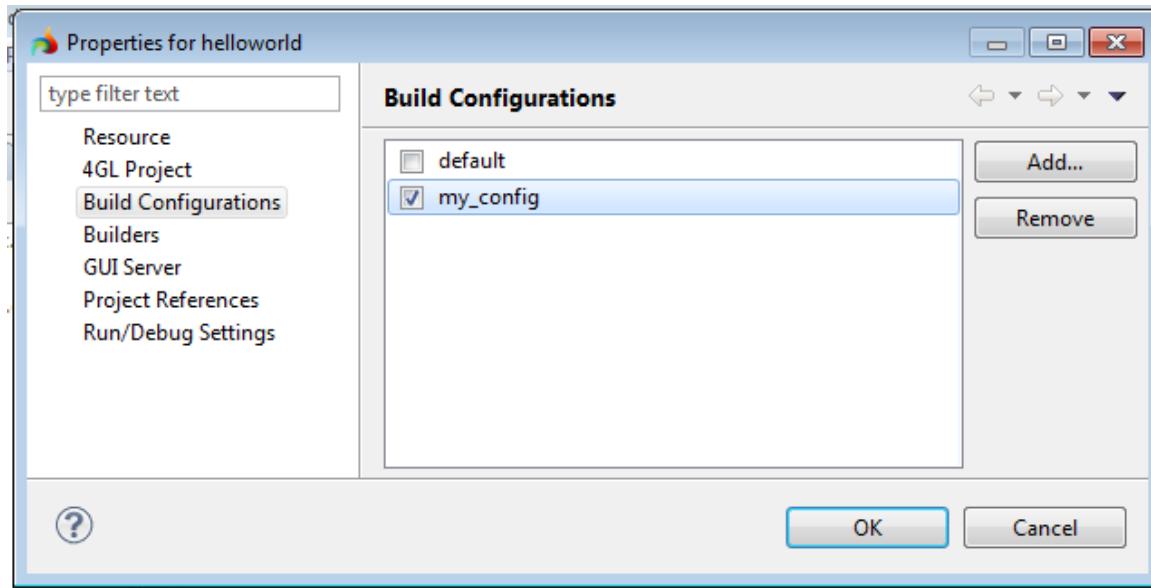
The default configuration cannot be deleted.

5. Press the **Add** button to add one more build configuration and type the name for the new configuration:



6. Press **OK**. The new configuration will appear in the list.

7. Select the new configuration:



Now you can add or remove a source file to or from the requirements of the current build configuration without affecting the default configuration.

Adding files to the build configuration

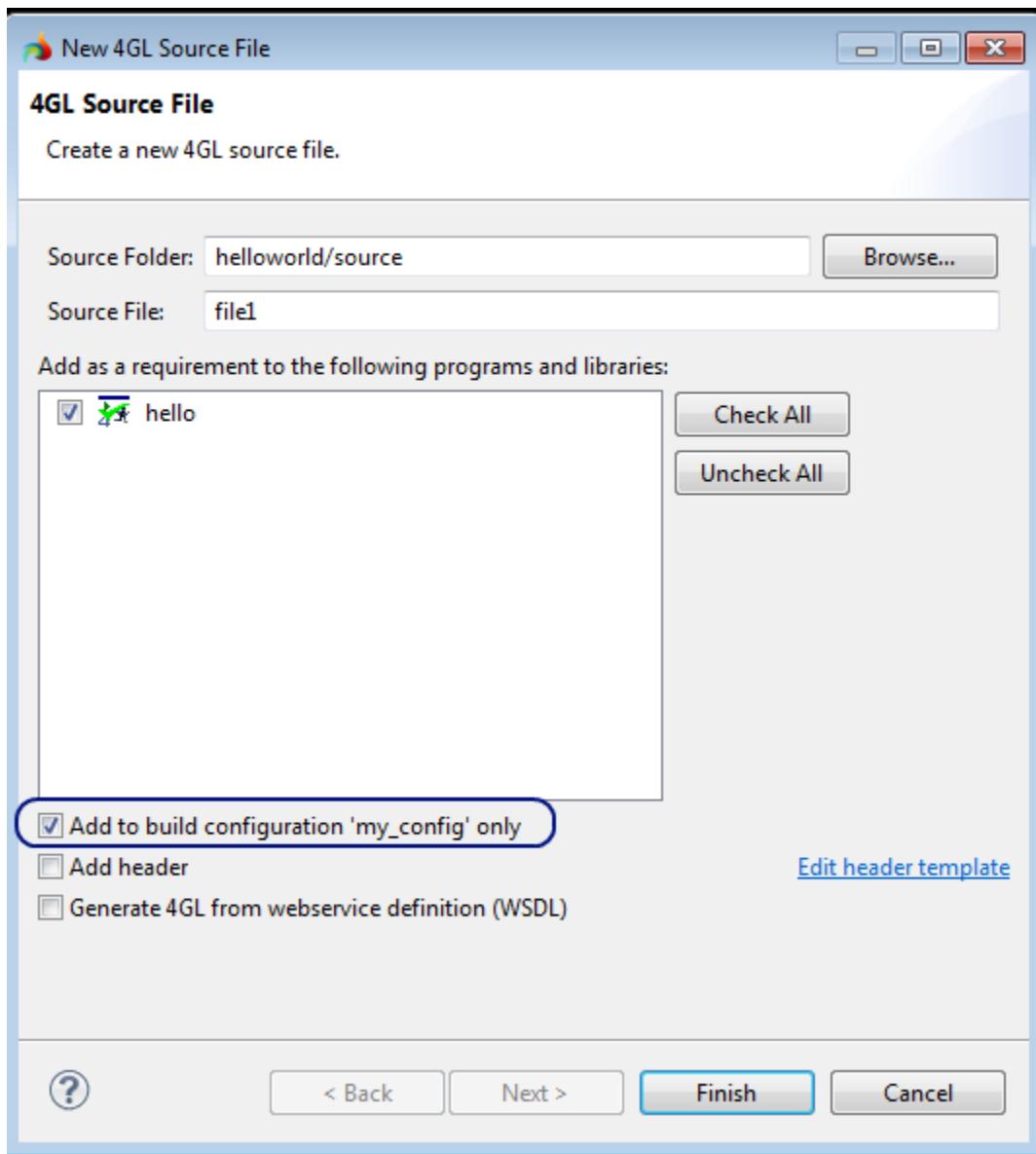
You can add already existing source files to your build configuration or you can create a file which will be required only by the current build configuration. It can be done only if a non-default configuration is the current configuration. All the special options are not visible if the default configuration is enabled.

To add a new file to the requirements of a non-default build configuration follow these steps:

1. Open the "helloworld" project you have created
2. Use the **File -> New -> 4GL Source File** option to create a new source file

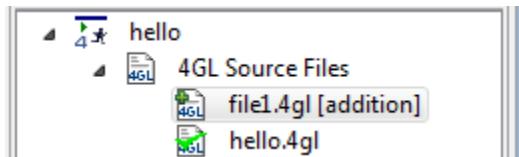


3. Type the "file1" as the name of the new file
4. Make sure that the "hello" program is selected in the Requirements field
5. Select the "Add to build configuration <configuration name> only" as shown in the picture below and press **Finish**:

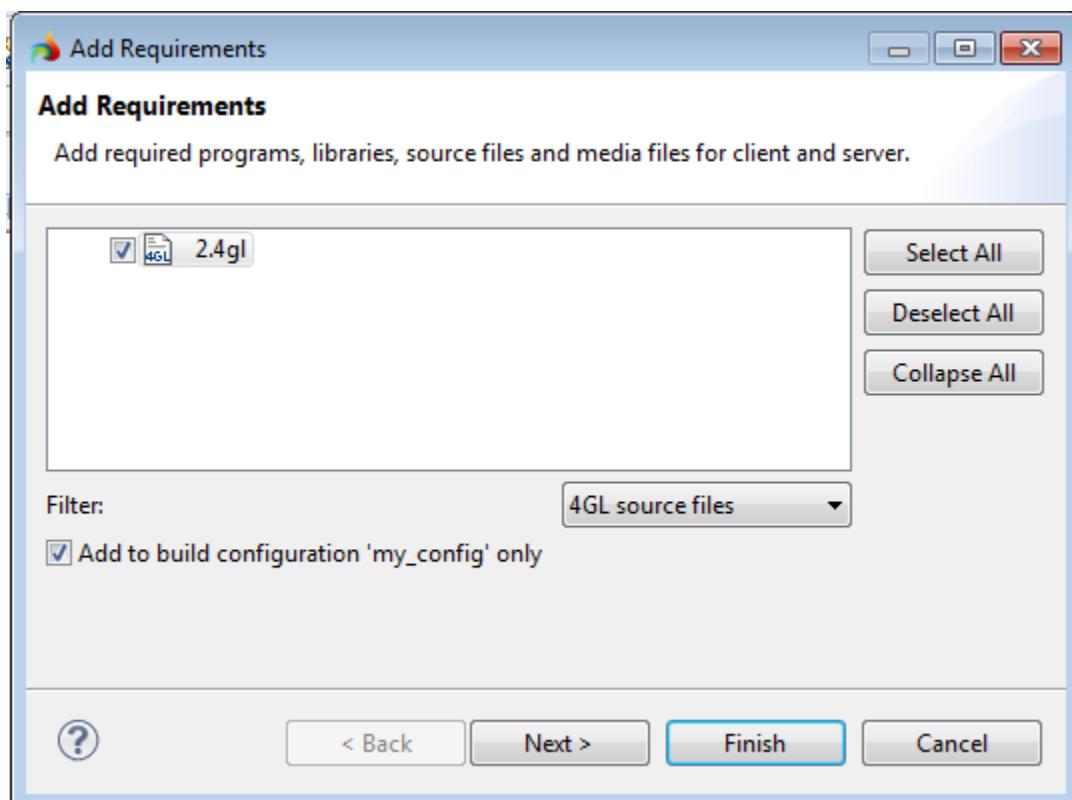




6. The new file will appear in the 4GL Project View with the special note “[addition]” and with a special file icon:



To add a file that already exists to the requirements of a non-default build configuration, right-click the project and select the **Add Requirement** form the project context menu. In the Add Requirements dialog select the file and the “Add to build configuration <configuration name> only” option and press **Finish**:



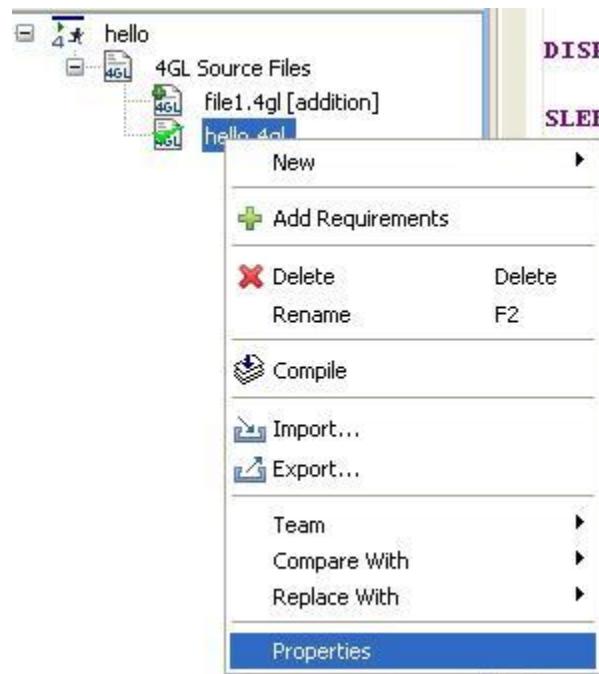
This option is visible if a non-default build configuration is enabled.

Removing files from the build configuration

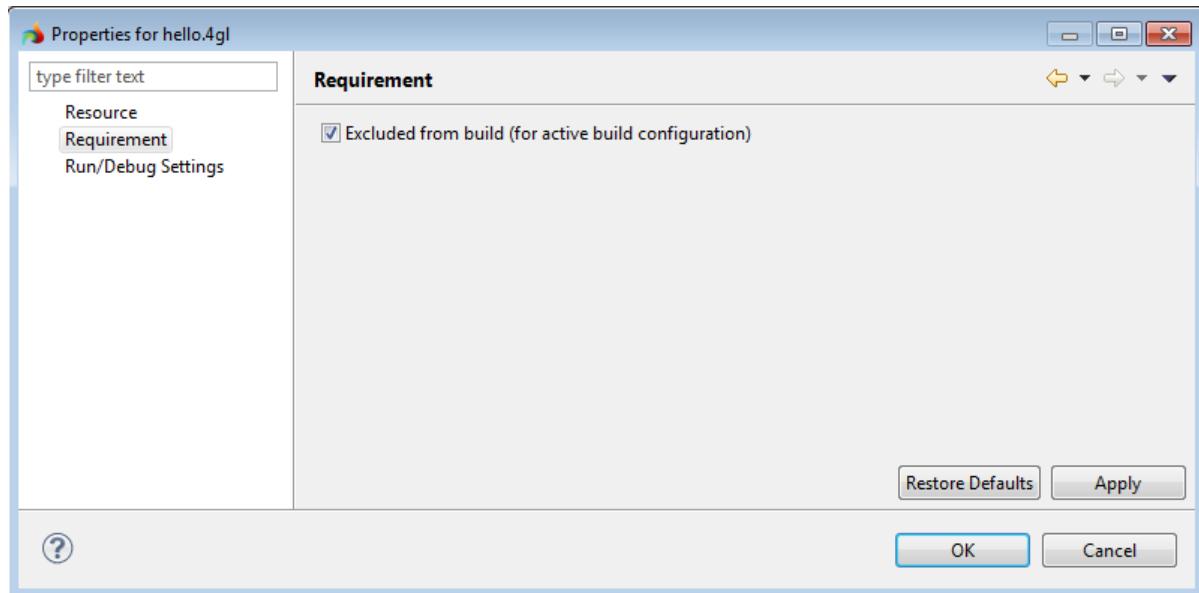
The way in which you can exclude a file from a non-default build configuration depends on whether this file is also present in the default build configuration.

To exclude a file which is also present in the default configuration do the following:

1. Right-click the “hello.4gl” file of the “helloworld” project and select the **Properties** option from the context menu:

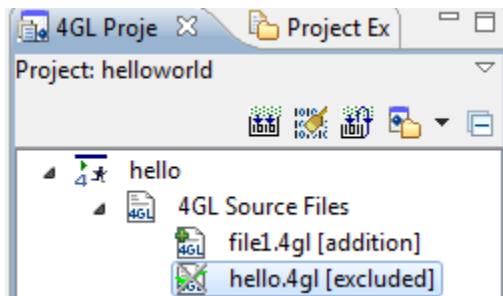


2. The properties of this source file will be displayed. Select the Requirement tab of the Properties dialog and select the corresponding option:

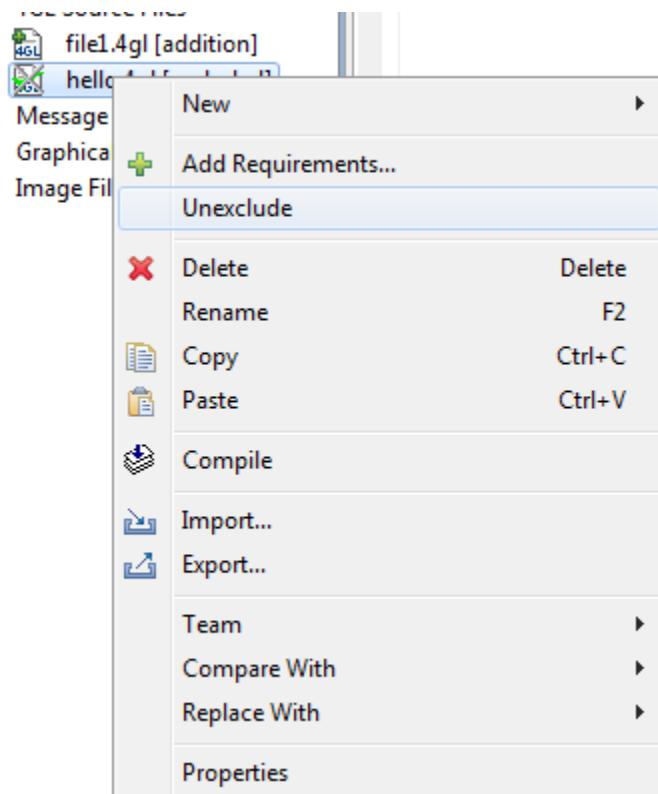




3. Press **OK**. The "hello.4gl" file will be excluded from the current build configuration, the note "[excluded]" will be added to its name and the file icon will become crossed:

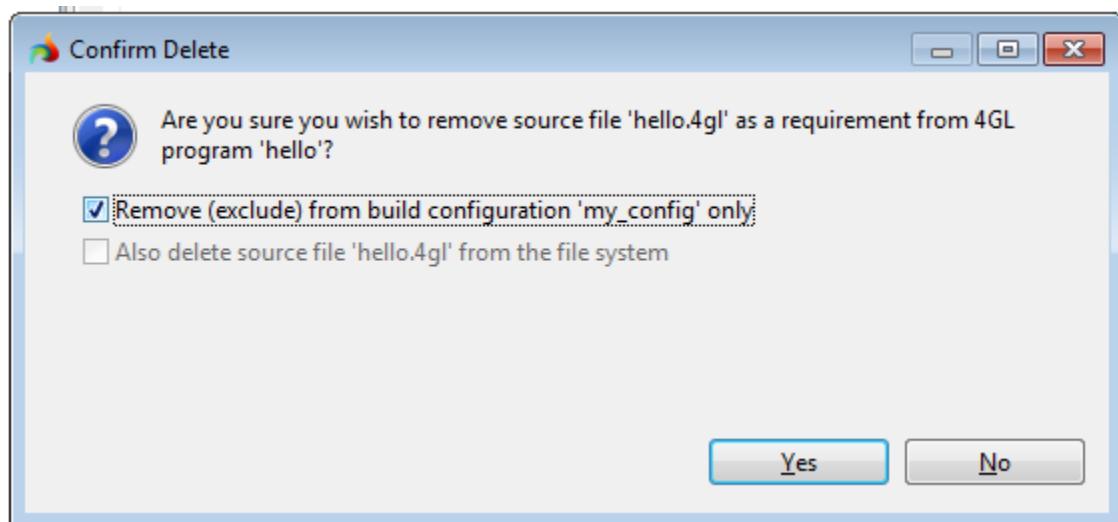


4. If you decide to include this file into the build configuration again, right-click the file and select the **Unexclude** option from the context menu:

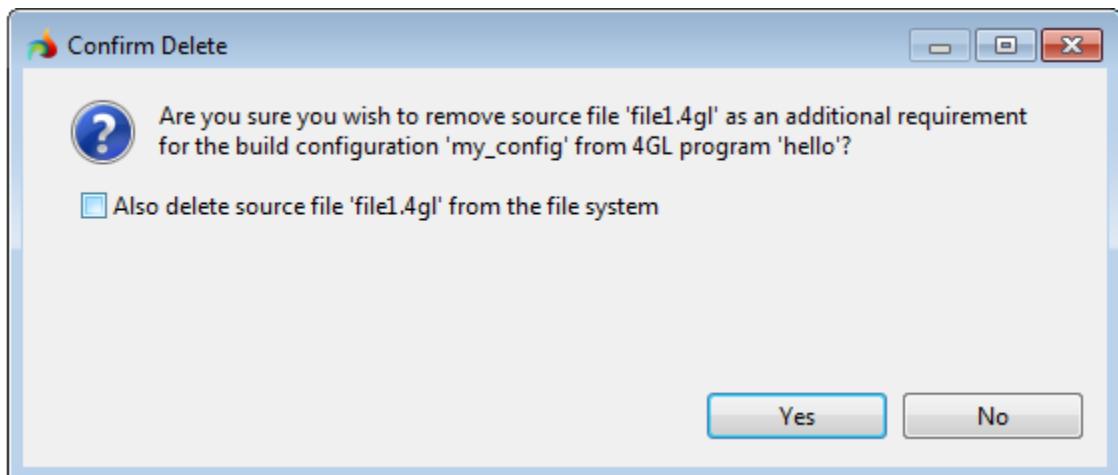


You can also remove the check mark in the Requirement properties page.

You can also exclude a file from the current build configuration by selecting the **Delete** option from the context menu of the file. You will be asked whether you want to delete the file from the file system or only from the current build configuration:



To exclude a file which is present only in the current build configuration, you can delete it either by pressing the Delete key or by selecting the **Delete** option from the context menu of the file. You will not see the option deleting it only from the current configuration, because it is the only configuration it is present in:



	Note: Do not select the option offering you to delete the file from the file system, otherwise the file will be lost and you will not be able to add it again as the requirement.
--	--

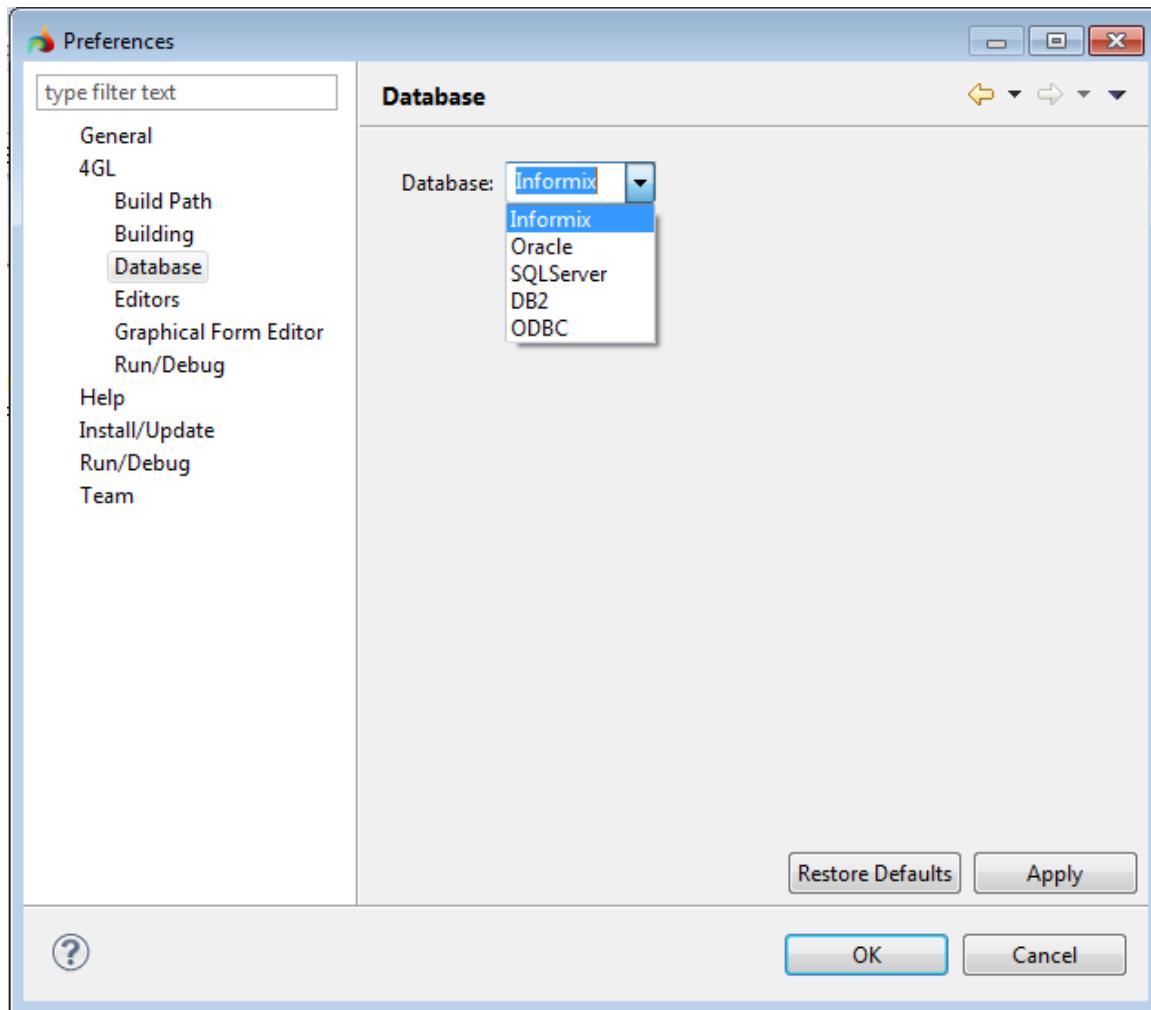
To add this file to the configuration again you should add it as an already existing file as described in the "Adding files to the build configuration" section above.

You can return to the default build configuration unaffected by all these changes by selecting it again in the project properties dialog.



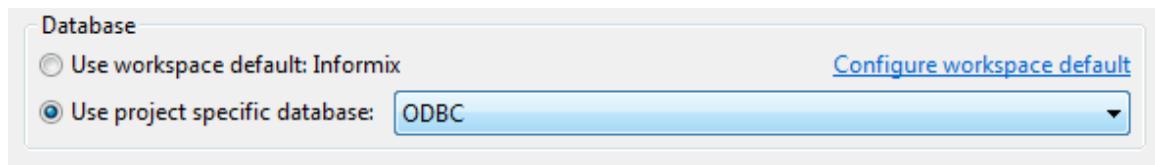
Changing the database

The database the program is being compiled and run against in character mode can be easily changed in Lycia. This option can be found under the **Window->Preferences -> 4GL ->Database** submenu option once the window below appears.



This preferences page is used to set the default database which is stored in the env.properties file. The database driver is also set in the inet.env file used for programs launched in GUI mode and in the environ.bat file used for the command line environment. For more information about setting the database driver you can read the Lycia Getting Started guide, Chapter 3: "Database connectivity", the additional information can be found in the "[Database connectivity](#)" and "[Configuration settings](#)" chapters of this guide.

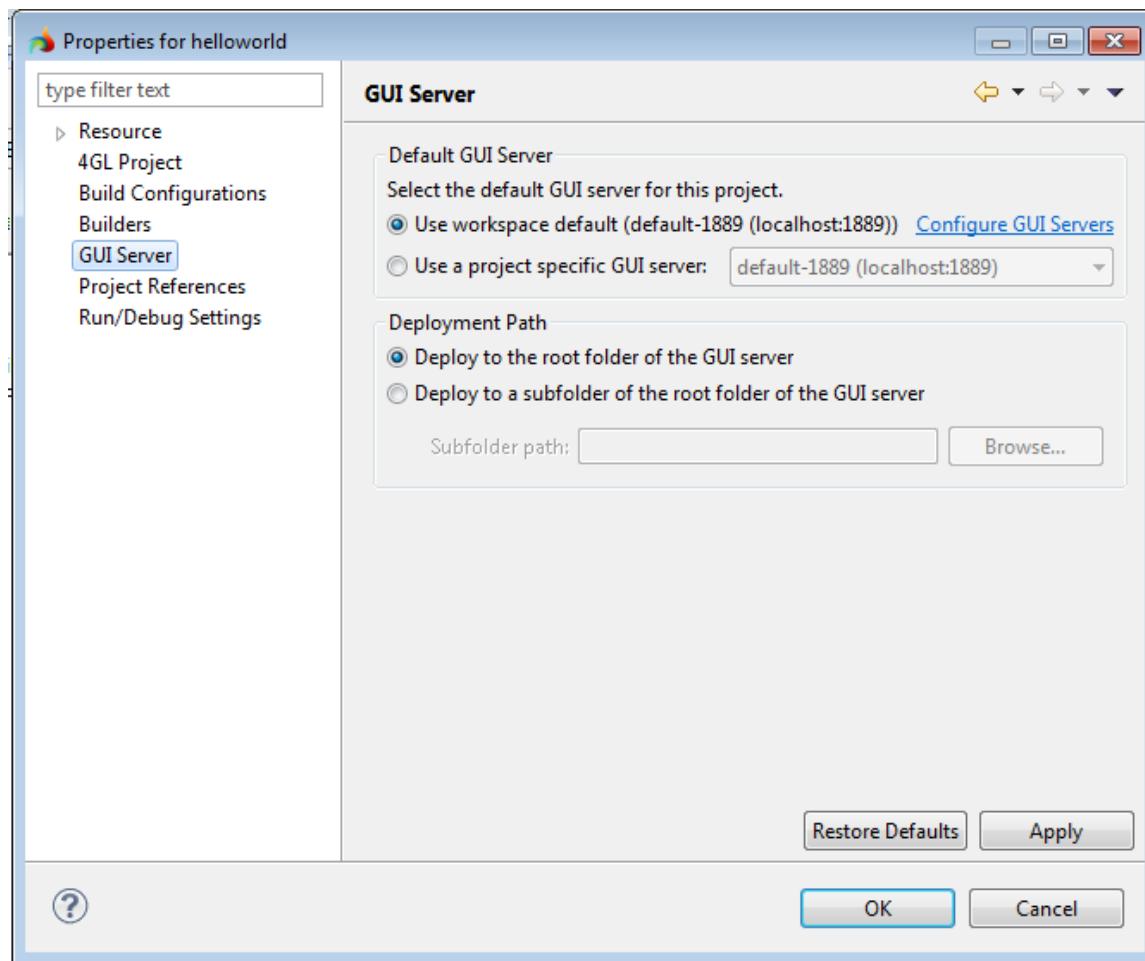
Each project within LyciaStudio can have its own database settings which differ from the default ones. To set a project specific database select the project and go **Project -> Properties ->4GL Project**. In the properties page you can set the type of database for the selected project to connect to:



Changing the GUI Server

The default GUI server can be set by using the option **Window ->Preferences -> 4GL -> Run/Debug -> GUI Servers**. Here you can choose the default server, edit the environment for the server and see the details of the selected GUI server. This preferences page is described in full detail in the “Preferences Menu” section of this guide; here we will explain how to use it.

The GUI Server can be easily changed within Lycia by using the option **Project->Properties ->GUI Server**. Here a project-specific GUI server can be set, which means that one workspace can have many projects and be deployed to different GUI servers. A project specific GUI server can be selected from the list of available GUI servers.

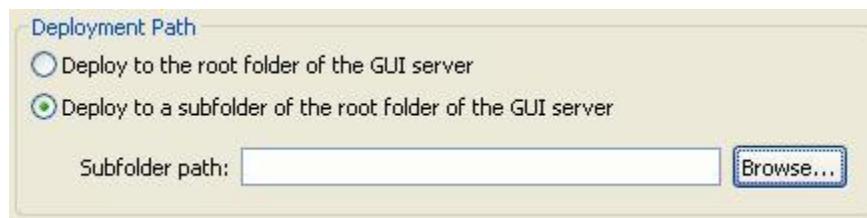




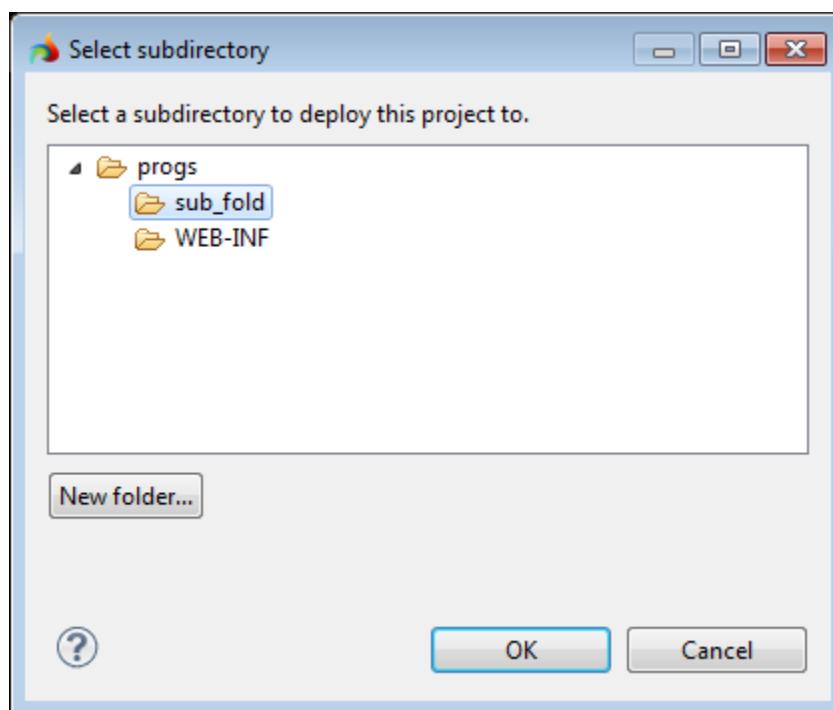
This properties page also contains settings which affect the deployment path. A project can be deployed either in the root folder of the GUI server or to a subfolder. The subfolder must be located within the \$LYCIA_DIR\progs directory.

To use a subfolder for deployment, follow these steps:

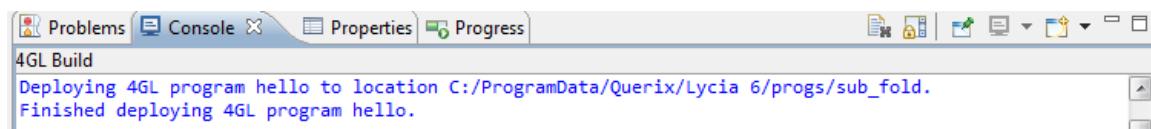
1. Select the "Deploy to a subfolder" option:



2. Press the **Browse...** button. If the "progs" folder does not contain the required subfolder then use the **New folder...** button to create it as shown below:



3. Select a subfolder and press **OK**
4. Now the selected project will be deployed to the specified subfolder when it is run in the GUI mode:





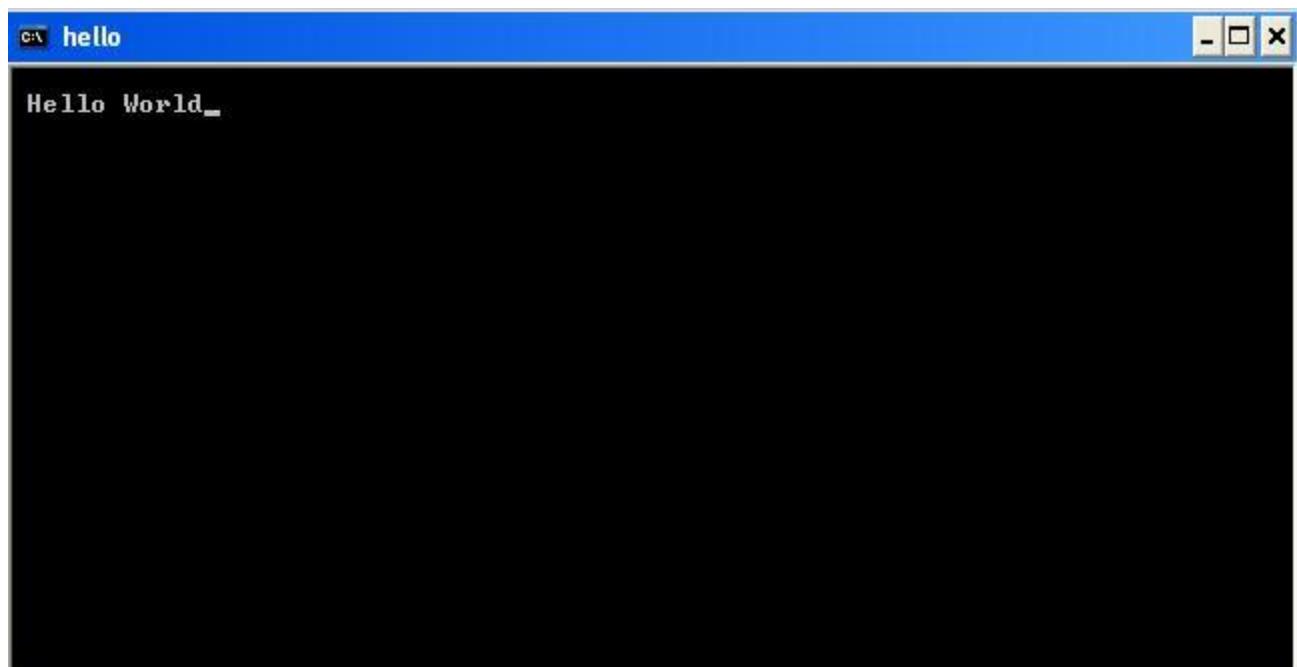
5. A project that is run using an existing run configuration will be deployed to the folder specified in this run configuration. The project properties including the deployment path will be applied only to a new run configuration. For more details on this, see the "Running a program" section below.

Running a program

Running a program from LyciaStudio can be as simple as clicking one button. To be able to run a program you must build it first. You can use the **Preferences ->Run/Debug -> Launching** to make LyciaStudio build a program before running it, if the build is required. For more information see the "Preferences Menu" chapter.

There are several ways to run a program in LyciaStudio:

- Double-click a program in the 4GL Project View. It will be run according to its run configuration. If the program is run for the first time and its run configuration is not set, it will be run in character mode by default:

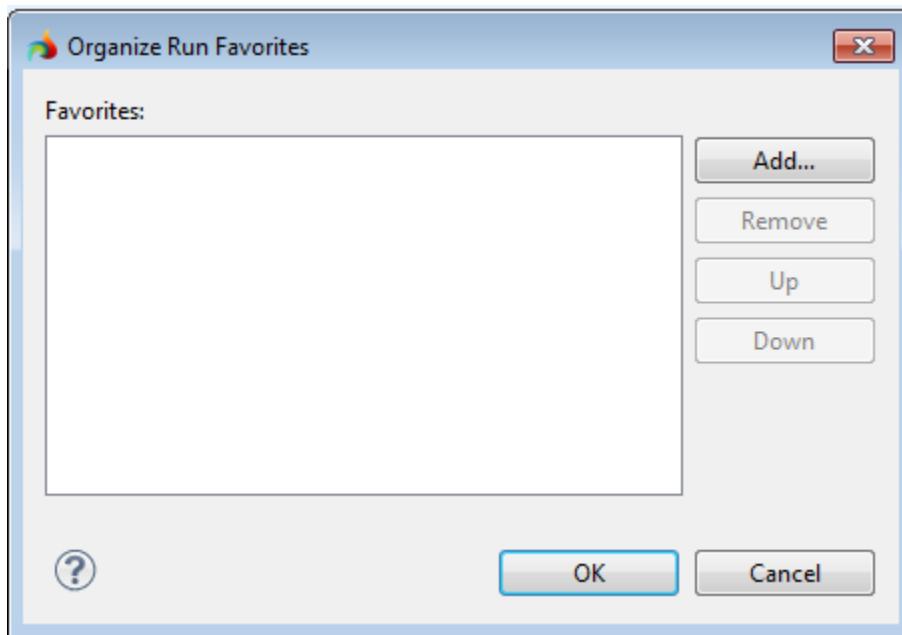


If there are several run configurations for the launched program, you will be presented with the list of available configurations from which you can choose the run configuration you want.

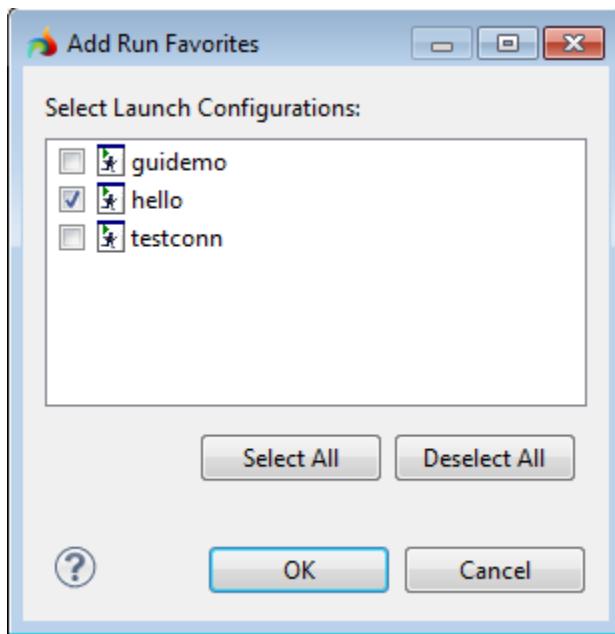
- Select the program you want to run and press one of the main toolbar buttons. If the graphical client you try to launch a program with is not installed on your system, you will receive a warning message and the program will not be launched.



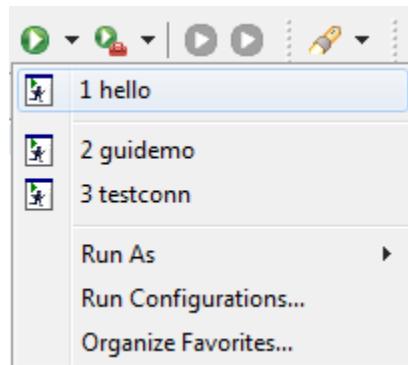
- Right-click the program in the 4GL Project View. Select the **Run As...** option from the context menu and choose the mode in which you want to run the program. If a graphical client is not installed on your system, it will be absent from the list of the **Run As...** options. You can also use the **Run -> Run As...** option of the main menu to run a program.
- Use the **Run As** button on the main toolbar
 - Press the Run As button and choose the launch mode from the pop-up window from which you can select the client to run the program with.
 - Press the arrow next to the **Run As** button to open a dropdown menu and select the launch mode.
 - The **Run As** dropdown menu also stores the most recently used run configurations. If a program has several run configurations, there will be several items for the same program in the list.
 - Use the **Organize Favorites...** option of the Run As dropdown menu to select the launch configurations which will be placed at the top of the launch list:



Press the **Add...** button and select the launch configurations you want to include into your run favourites list and press **OK**:



Now your favourite launch configurations will be at the top of the list:



Run Configurations

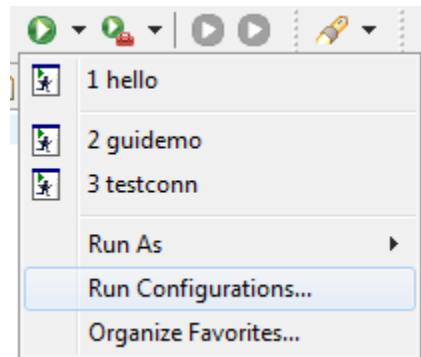
Run configurations are the configurations used to launch a program. Initially, when a program is created or imported into LyciaStudio, it has no run configuration. A run configuration for such program is typically created when it is run for the first time. The mode in which the program is run, the GUI server and other launch conditions are stored in the run configuration. This configuration is used every time a program is launched by double-clicking or by means of the **Run As** button.

By default, a run configuration of a program is updated each time the program is run with the help of the toolbar run buttons or by using the **Run As...** option, when the launch mode is selected explicitly. The default behaviour can be changed using the **Preferences -> 4GL -> Run/Debug** preferences page. For more details see the "Preferences Menu" chapter.

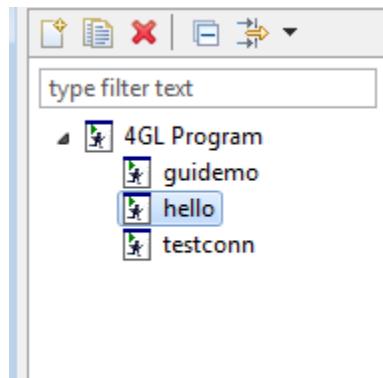


To modify the launch configuration of a program, do the following:

1. Select the **Run Configurations...** option from the **Run As...** dropdown menu or select the **Run -> Run Configurations...** option from the main menu:



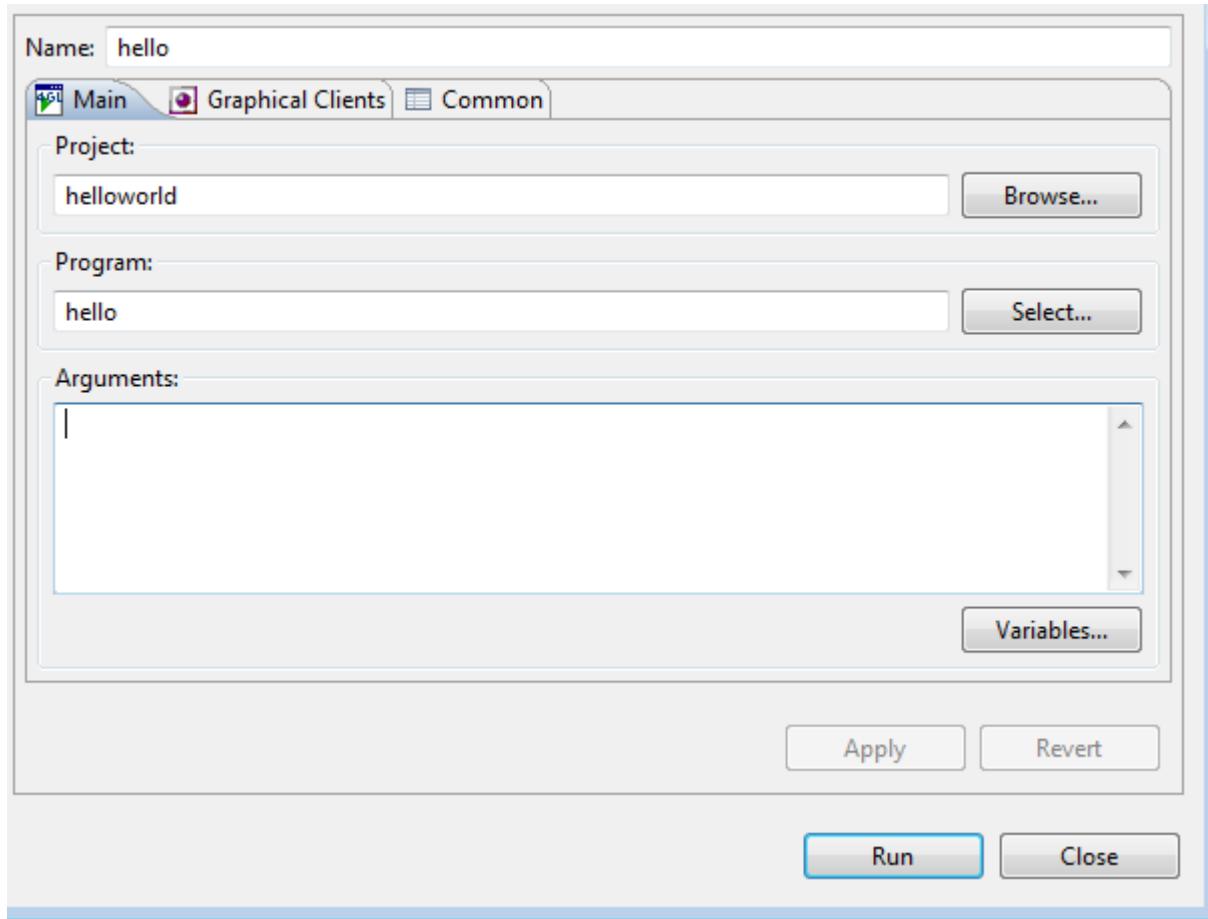
2. The Configurations dialog will be opened. Select the launch configuration which you want to modify from the list in the left pane:



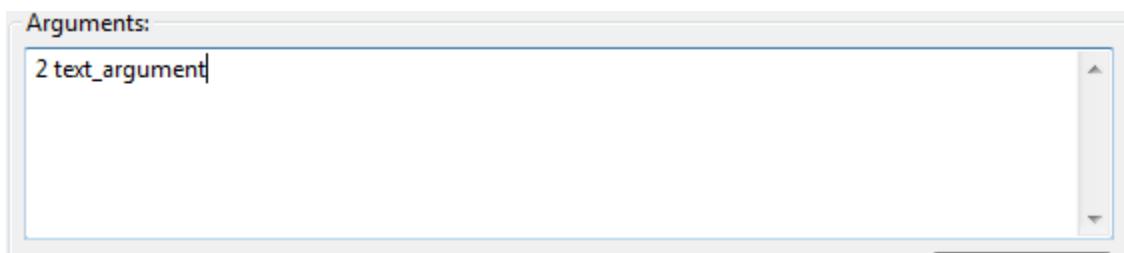
The name of the launch configuration which is created automatically on the first launch of a program is typically the same as the name of the corresponding program. You can change the name of any launch configuration using the right pane of the dialog.



3. The Configurations dialog consists of three tabs. The "Main" tab contains the general information about the run configuration:



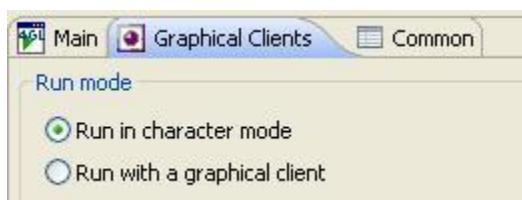
- a. You can press the **Select...** button to select a program if a project contains more than one program. You cannot rename a project or a program using this interface.
- b. Here you can also specify one or more arguments for the program, which must be separated by whitespaces:



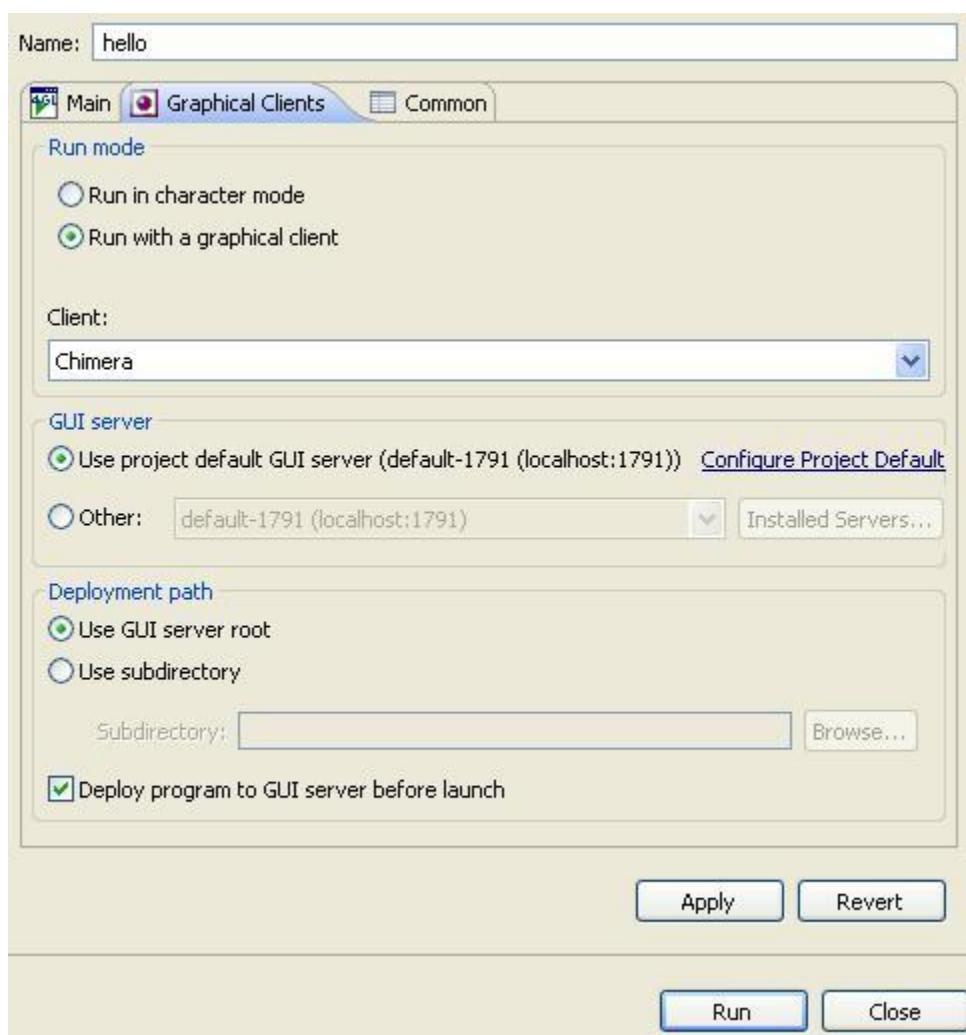


4. The “Graphical Clients” tab contains the settings connected with the thin clients (Phoenix and Chimera).

- a. When the “Run in character mode” option is selected, all other options are disabled:



- b. When the “Run with a graphical client” option is selected, the other options become enabled:





c. You can choose the graphical client from the combo box:

A screenshot of a dropdown menu titled "Client". The menu contains three items: "Chimera", "Phoenix", and "Chimera". The last item, "Chimera", is highlighted with a blue selection bar at the bottom of the list.

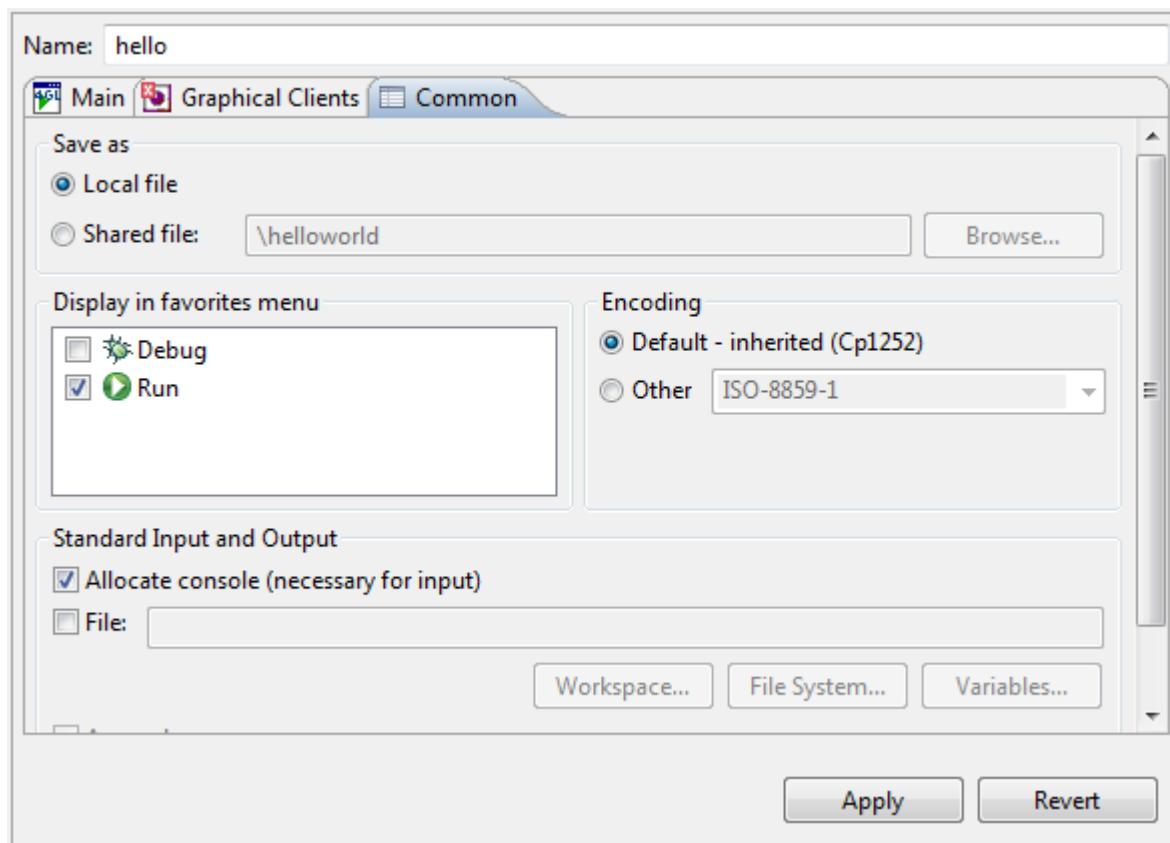
d. A GUI server used for deploying can be selected for the particular program. It can be either the default server or any other server from the installed ones:

The screenshot shows the "GUI server" configuration panel. It has two main sections: "Deployment path" and "Deployment". Under "Deployment path", the "Other" radio button is selected, and "default-1889 (localhost:1889)" is chosen from a dropdown menu. There is also a "Configure Project Defaults" link. Under "Deployment", "default-1891 (localhost:1891)" is selected. A "Browse..." button is located at the bottom right of the deployment section.

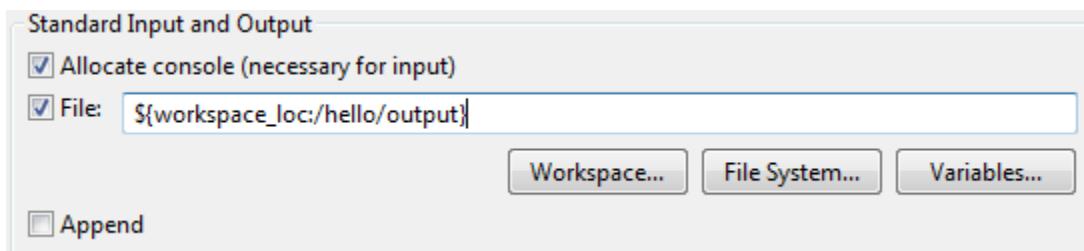
e. You can select a folder into which you want to deploy a program by pressing the **Browse...** button. This folder must be within the \$LYCIA_DIR\progs directory.

The screenshot shows the "Deployment path" configuration panel. It has two main sections: "Deployment path" and "Deployment". Under "Deployment path", the "Use subdirectory" radio button is selected. A "Subdirectory" input field and a "Browse..." button are located below it. Under "Deployment", the "Deploy program to GUI server before launch" checkbox is checked.

5. The "Common" tab deals with other launch options.



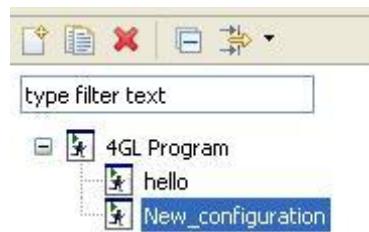
- The "Save as" option allows you to save the .launch file which contains the launch configuration settings as a local file or as a shared file.
- You can specify a file for processing output. To select a file, use the buttons below the "File" field. To append new data to the selected file during the next launch, select the "Append" option:



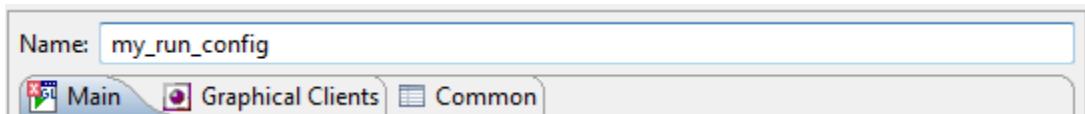
Adding a configuration

A program can have more than one run configuration. To create a new configuration for a program you can do the following:

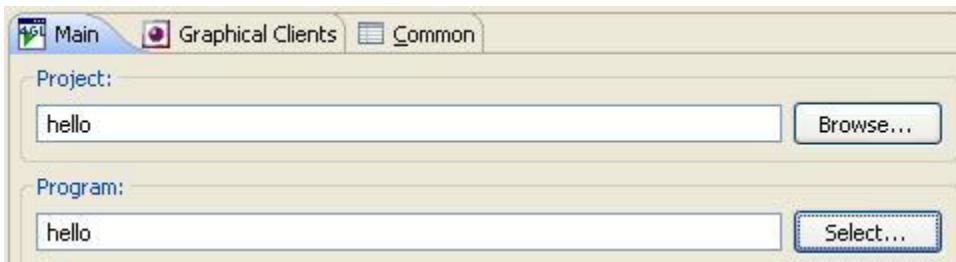
- Press the New launch configuration button () at the top of the left pane.
- A new configuration will be created:



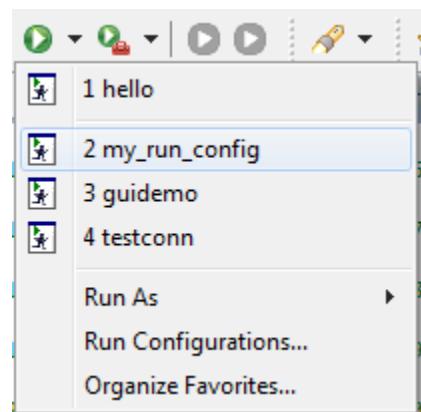
3. Type in the new name for the run configuration:



4. Select the project and the program to which the configuration will apply by pressing the **Browse...** and **Select...** buttons respectively, or you can type them in the corresponding fields:

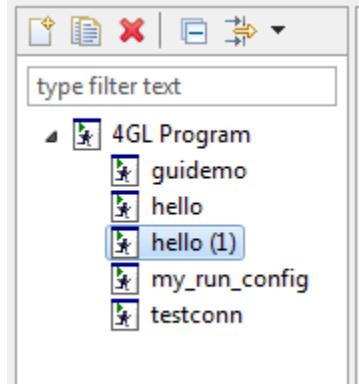


5. Adjust the setting of the "Graphical Clients" and "Common" tabs as you like. To save the changes, press the **Apply** button on each tab. The changes will be saved automatically, if you press the **Run** button.
6. In the dropdown menu of the **Run As** button you will see your new run configuration:





To create one more run configuration for a program, select the existing configuration and press the Copy button. A copy of the existing configuration will be created. You can adjust the settings of this configuration to meet your requirements:



Searching

LyciaStudio comprises a quick and convenient search engine which allows you to look for files and for the text within the files. The search is performed among the files currently open in LyciaStudio by means of the Search dialog. The results of the search are displayed in the "Search" view which is opened automatically, when the search is complete.

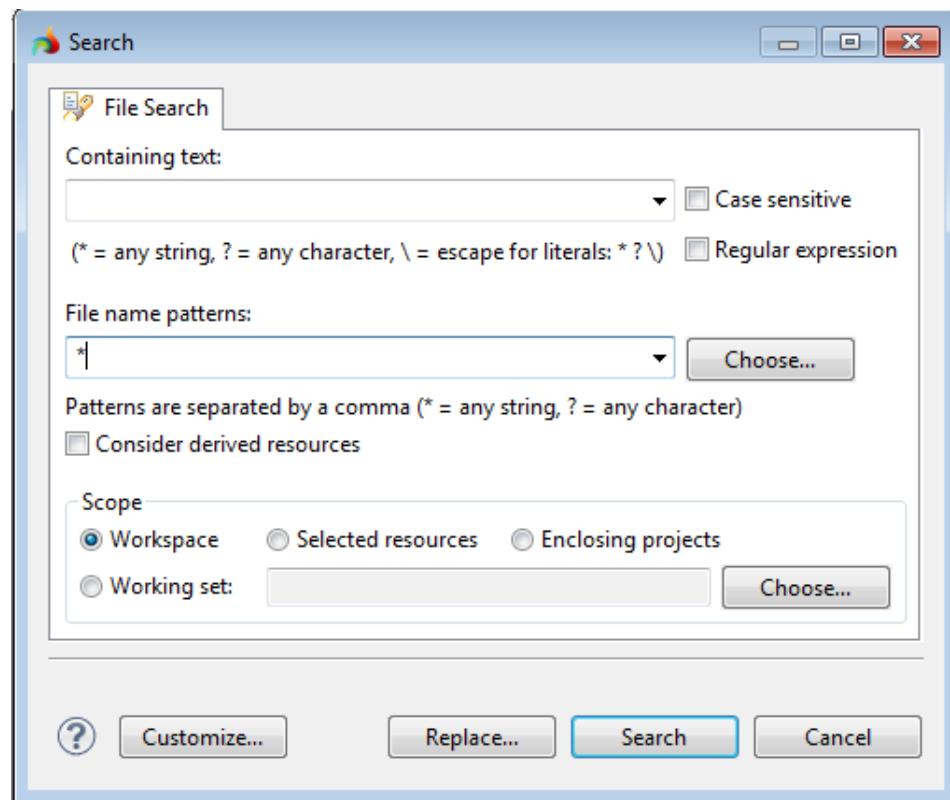
Searching for source files

To search for source files, do the following:

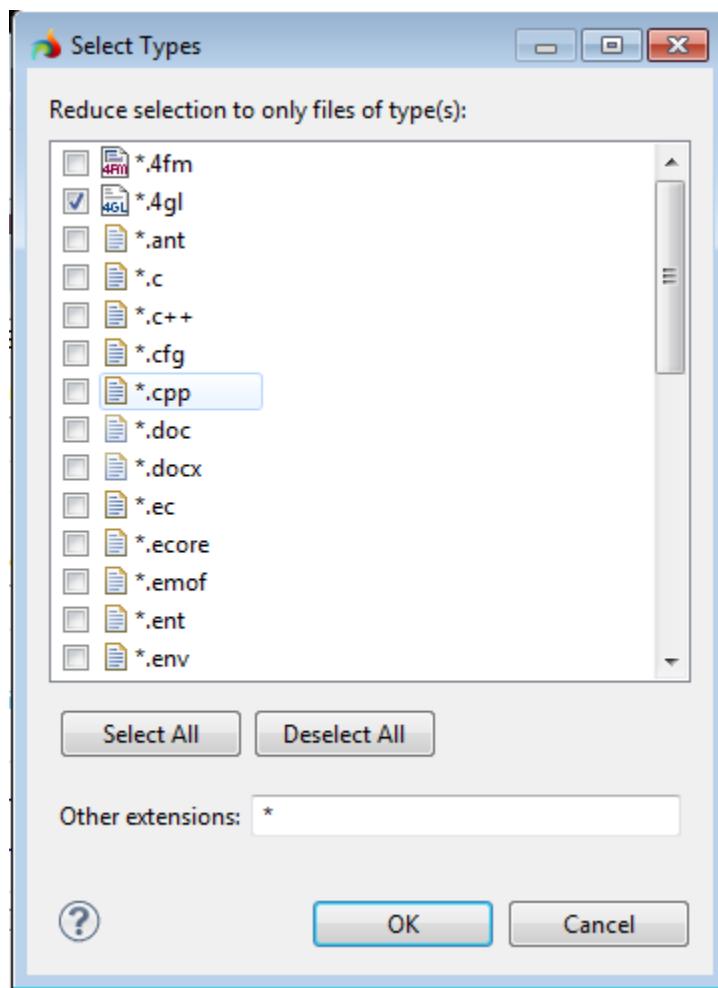
1. Make sure that "helloworld" project created according to the instructions of this guide is present in LyciaStudio.
2. Press the Search button on the main toolbar or select the **Search -> File** main menu option.



3. You will be presented with the search dialog:



4. If you know the name of a file you are looking for, type it in the "File name patterns" combo box, e.g "hello.4gl":
- If you do not know the exact name of a file, you can enter a part of its name with a wildcard symbol (e.g. hello.* , *.4fm).
 - If you want to search for a definite type of source files, you can press the **Choose...** button and select the types of files you want to include into the search criteria:



- c. Select the *.4gl file type from the list and press **OK**, you will see it in the "File name patterns" field:

File name patterns:

Patterns are separated by a comma (* = any string, ? = any character)

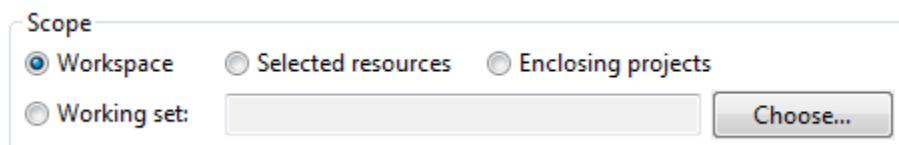
5. You can search for a file that contains certain text. To do this type the text in the "Containing text" field. You can make the text case sensitive and if you tick off the "Regular expression" option, Lycia will search only for regular expressions which match the entered text. You can also use the wildcard symbols to substitute a random symbol or a character string. Type the "Hello" string in this field:

Containing text:

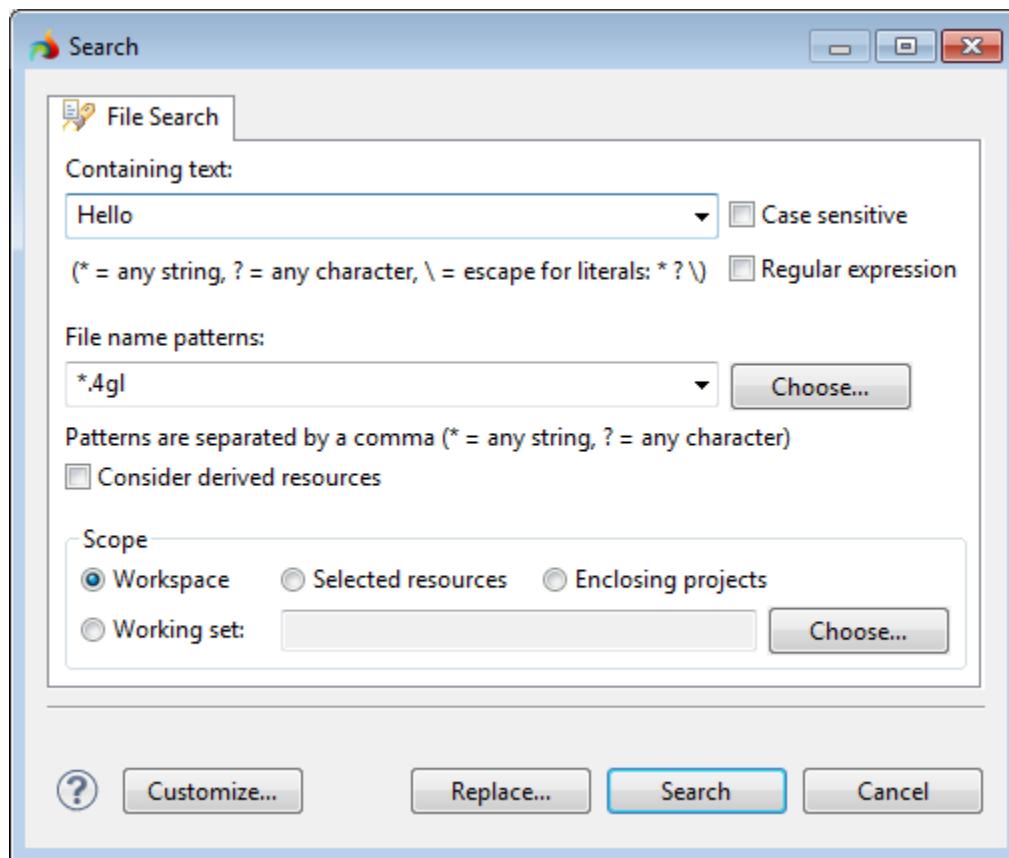
 Case sensitive
(* = any string, ? = any character, \ = escape for literals: * ? \) Regular expression



6. You can specify the files and folders which should be included into the search in the "Scope" section of the Search dialog. You can search in all files located in the workspace, when the "Workspace" option is selected. Select the "Enclosing projects" option to search only in the selected project. You can select a number of resources (projects, files, etc) in the Project Explorer view or in the 4GL Project view and choose the "Selected resources" option to search only in the selected resources. You can also select a working set as the search target by pressing the **Choose...** button (for more information about the working sets, see the "Building and compilation" section of this chapter). Leave the Workspace as the scope of search:



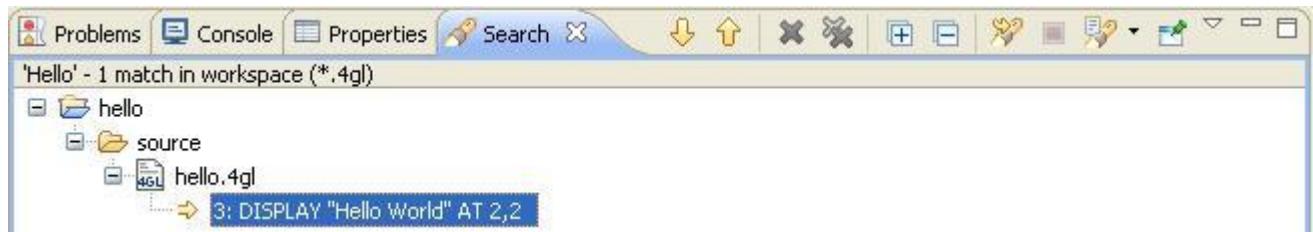
7. Now the Search dialog should look as follows:



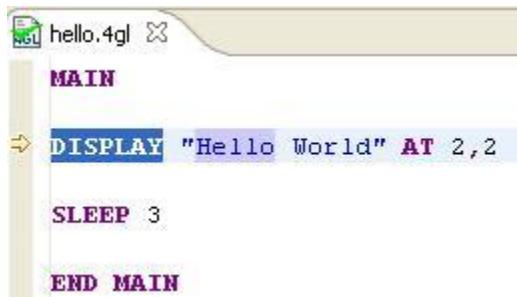
Lycia will look for all the .4gl files in the workspace that contain "Hello" character string.



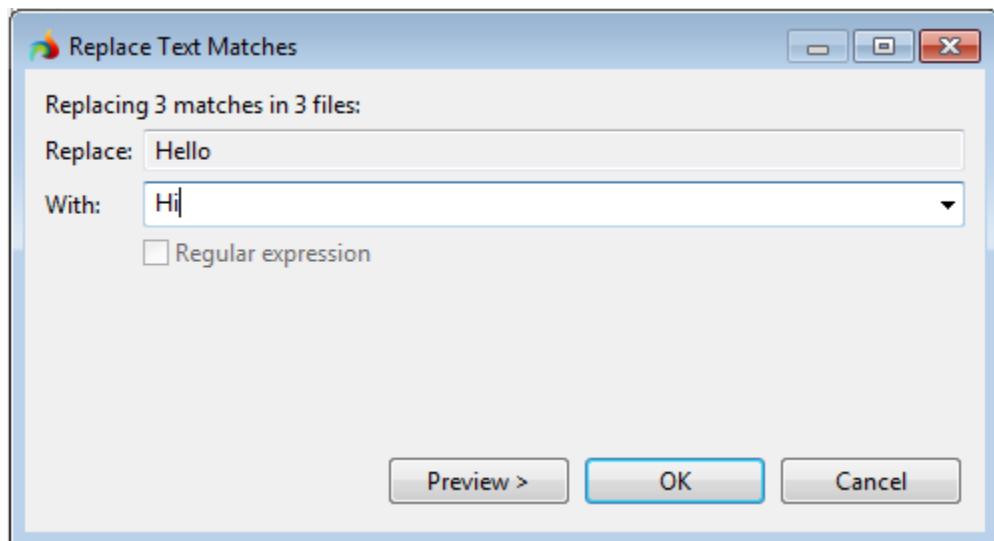
8. Press the **Search** button. The “Search” view will be opened and the search results will be displayed:



9. Double-click on the search result, the file which contains this string will be opened and brought to the top:

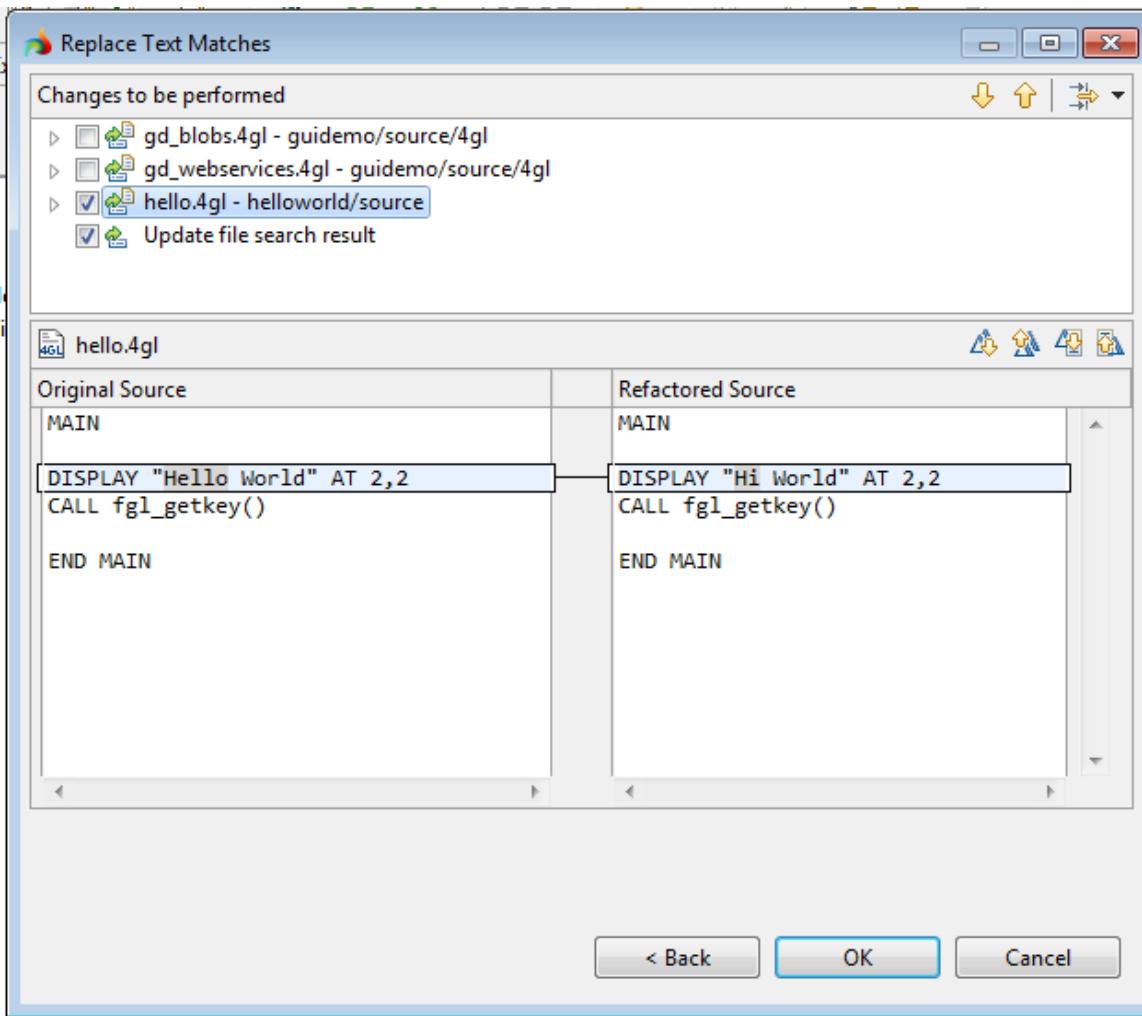


10. Press the Search button on the main toolbar again. The Search dialog will be opened with the search criteria used for the previous search.
11. Press the **Replace...** button, you will be presented with a Replace dialog which searches for the specified character string and replaces it with the new character string.
12. Type the character string "Hi" to replace the "Hello" character string:





13. Press the **Preview>** button to preview the result of replacement:



If the text is found in several source files, select those in which you want to perform changes.

14. Press **OK** to accept the changes or **Cancel** to cancel replacement.

Searching 4GL code

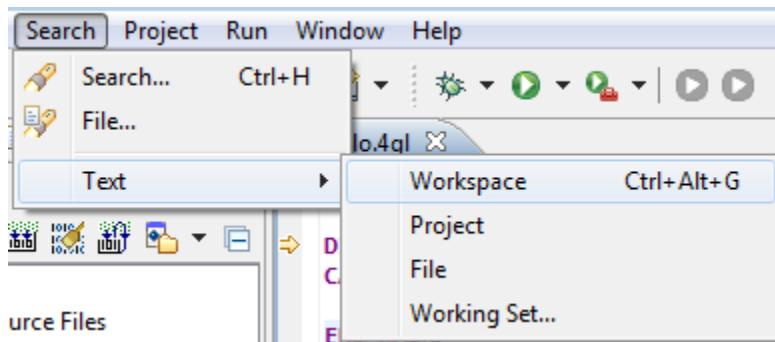
You can search for the existing source code in other locations. It is a convenient way to find similar source code in different files.

1. Import any demo project, thus you will have at least two projects in the workspace: your "helloworld" project and the demo project.
2. Open the hello.4gl file and select the DISPLAY statement with the mouse cursor:

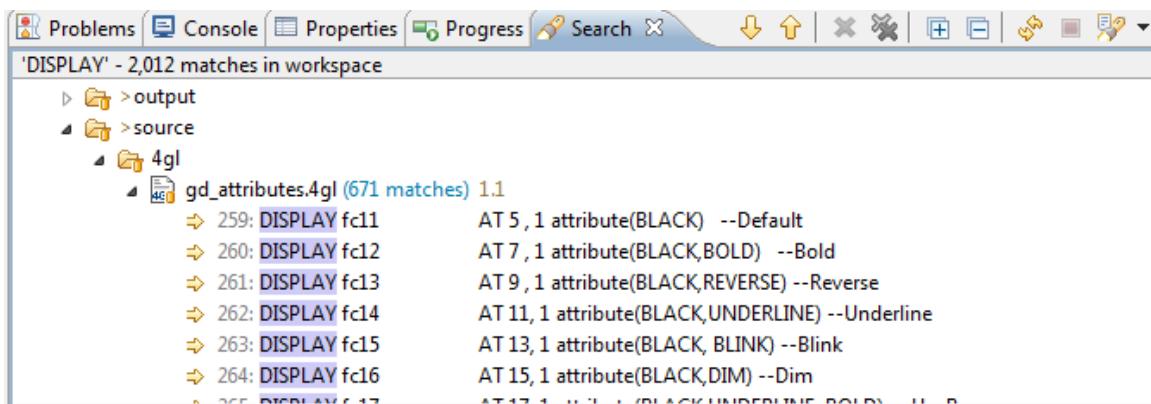
A screenshot of the hello.4gl file in a code editor. The DISPLAY statement 'DISPLAY "Hi World" AT 2,2' is highlighted with a blue selection bar around its text.



3. Select the **Search -> Text -> Workspace** main menu option:

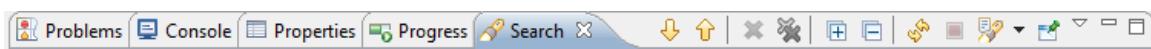


4. Lycia will search for the DISPLAY statement in all the source files within the workspace and will display the search result in the "Search" view:



Search View

The "Search" view is opened each time a search is complete. It contains the search results. The toolbar of the "Search" view looks as follows:



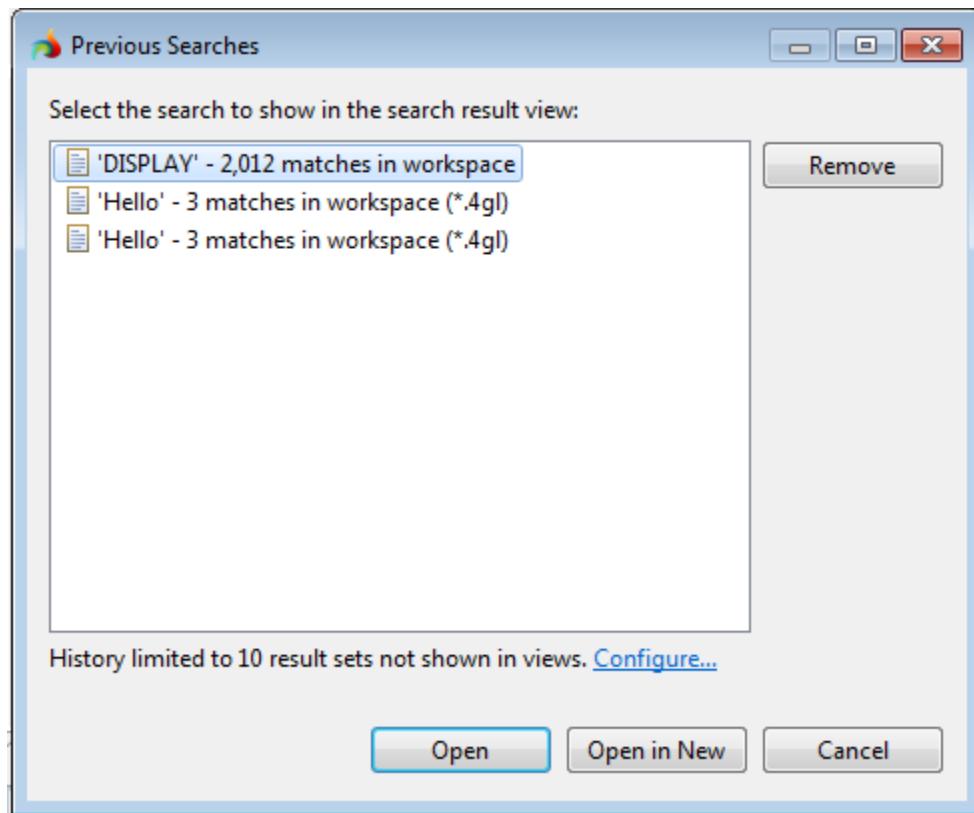
The following tools are used to manage search results:

- Up and down () buttons navigate through search results
- Delete () buttons allow you to delete one or more search results
- Pin the "Search" with the help of the button to keep it on top even if something is output to other views of this view stack

To run the current search one more time, press the F5 button or the button on the search toolbar.



Press the button to view the search history. You will be presented with the list of previously performed searches:



Here you can view all the searches that have been run and open any of them, either in the same "Search" view or in a new "Search" view which will appear next to the current "Search" view in the same view stack. Opening a search does not run this search again automatically. You can view only the results which were found the previous time this search was run. To update the results of a search opened from the search history, press the button on the view toolbar to run the opened search again. The search history cannot contain more than 100 search results; the default limit is 10 results.

Comparing and replacing source files

LyciaStudio comprises a tool for comparison of multiple resources and for the presentation of the results in a special compare editor. A source file can be compared with another source file or with the local history.

The local history preserves all the changes made in a source file beginning from the moment when it is imported or created. The local history of a file is maintained when you create or modify a file. Each time you edit and save a file, a copy of it is saved. This allows you to compare your current file state to a previous state, or replace the file with a previous revision. Each state in the local history is identified by the date and time the file was saved. Neither projects nor folders have local history.

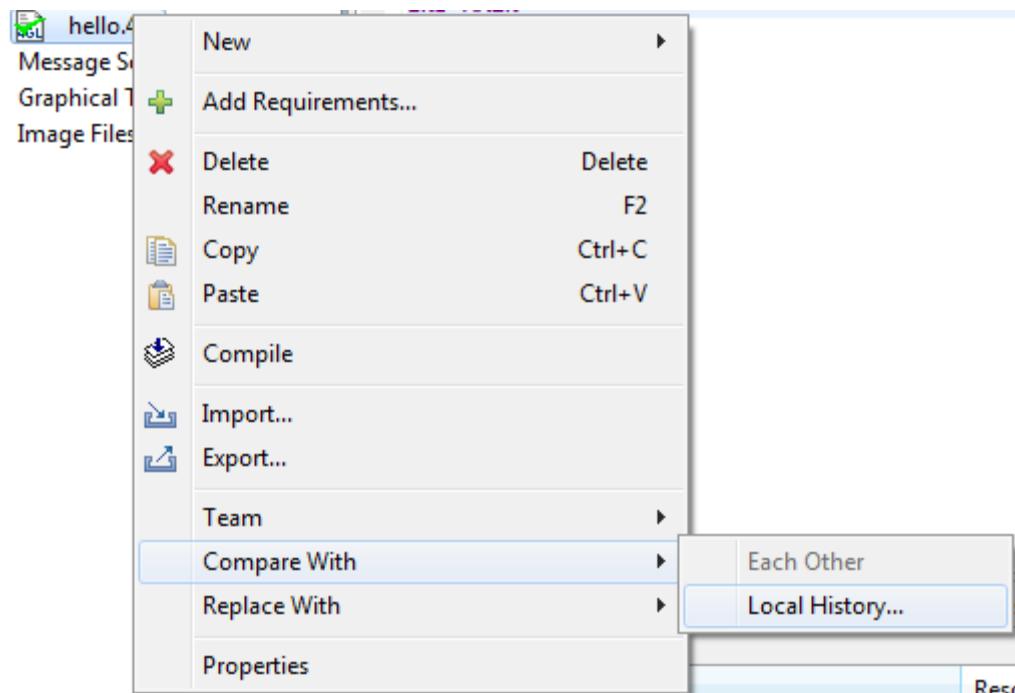


The files which are compared are opened in the Compare view side by side. The options and appearance of the Compare view can be adjusted using the **Window -> Preferences -> General -> Compare/Patch** preferences page.

Comparing source files

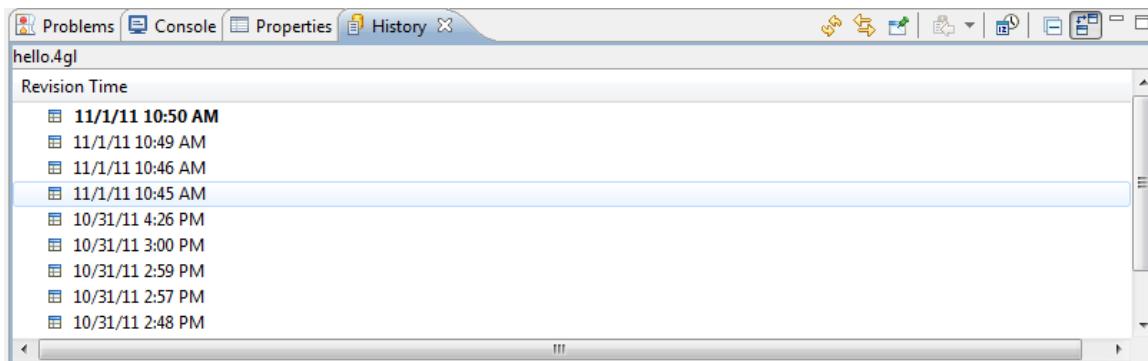
To compare a file with its previous version stored in the local history, follow these steps:

1. Make sure that "helloworld" project created according to the instructions of this guide is present in LyciaStudio.
2. Right-click the "hello" source file in the 4GL Project view and select the **Compare With - > Local History** option:

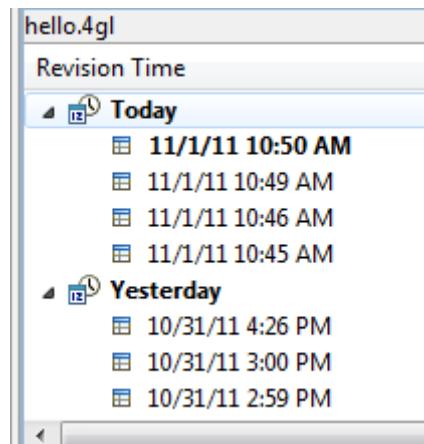


The **Compare With** option is also available from the context menu of the editor in which this file is opened.

3. The "History" view will be opened. The toolbar of the "History" view contains several tools that allow you to manage the local history:



- a. When the Link button is pressed the "History" view displays the history of the source file currently opened in the editor area.
- b. When the Group button is pressed, the revisions in the "History" view are grouped by date:



- c. If the Compare Mode button is enabled, double-clicking a revision will open a Compare editor. When it is disabled, the revision will be opened in a separate source text editor. Do not disable this button for this demonstration purpose.
4. Double-click the previous revision of the source file in the "History" view.



5. The Compare editor will be opened in the editor area:

```
Local: hello.4gl
Local history: hello.4gl Nov 1, 2011 2:39:41 PM

DISPLAY "Hi World" AT 2,2
CALL fgl_getkey()

END MAIN
```

```
MAIN

DISPLAY "Hello World" AT 2,2
CALL fgl_getkey()

END MAIN
```

6. You can see from above that the "Hi" character string differs from the "Hello" character string, as we have replaced "Hello" with "Hi" in the "Search" section of this chapter. The differences are marked with grey colour and the lines in which the differences are spotted are linked to each other.
7. The Compare editor toolbar contains tools which are used to manage changes and differences in the compared files. The tools differ depending on whether a file is compared with the local history or with another source file.
 - a. button copies all the non-conflicting changes from the right pane into the left pane.
 - b. button merges changes in two files by copying the highlighted change in the right pane into the highlighted fragment on the left. This will overwrite the highlighted fragment in the left pane.
 - c. and buttons navigate among the lines that contain differences found between the compared resources.
 - d. and buttons navigate among the individual changes found between the compared resources.

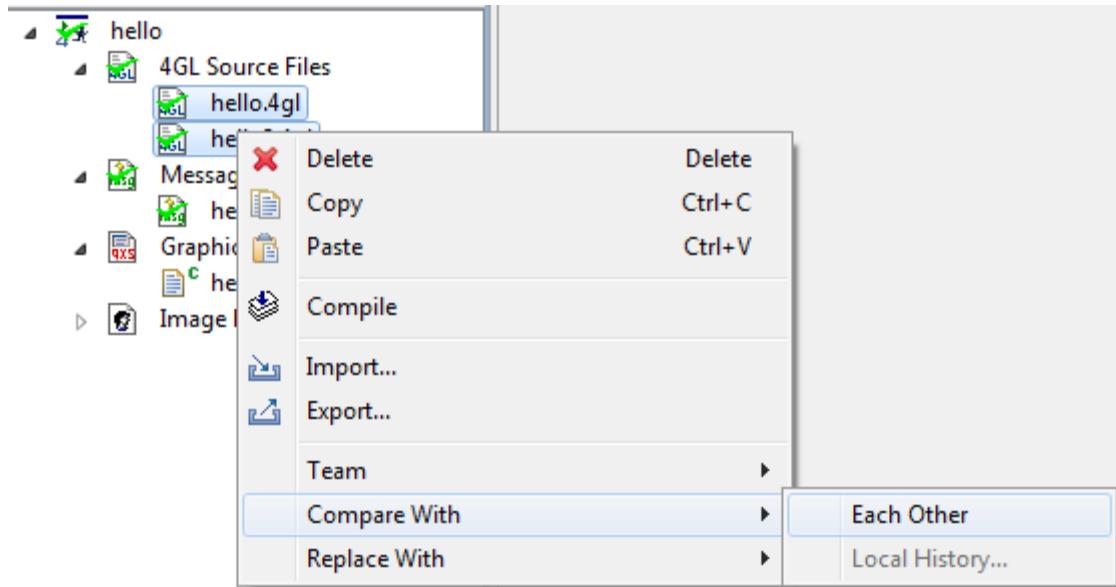
To compare two source files, follow these steps:

1. Make sure that "helloworld" project you created according to the instructions of this guide is present in LyciaStudio.
2. Create a second 4GL source file "hello2" within the "helloworld" project and copy the following code into this file:

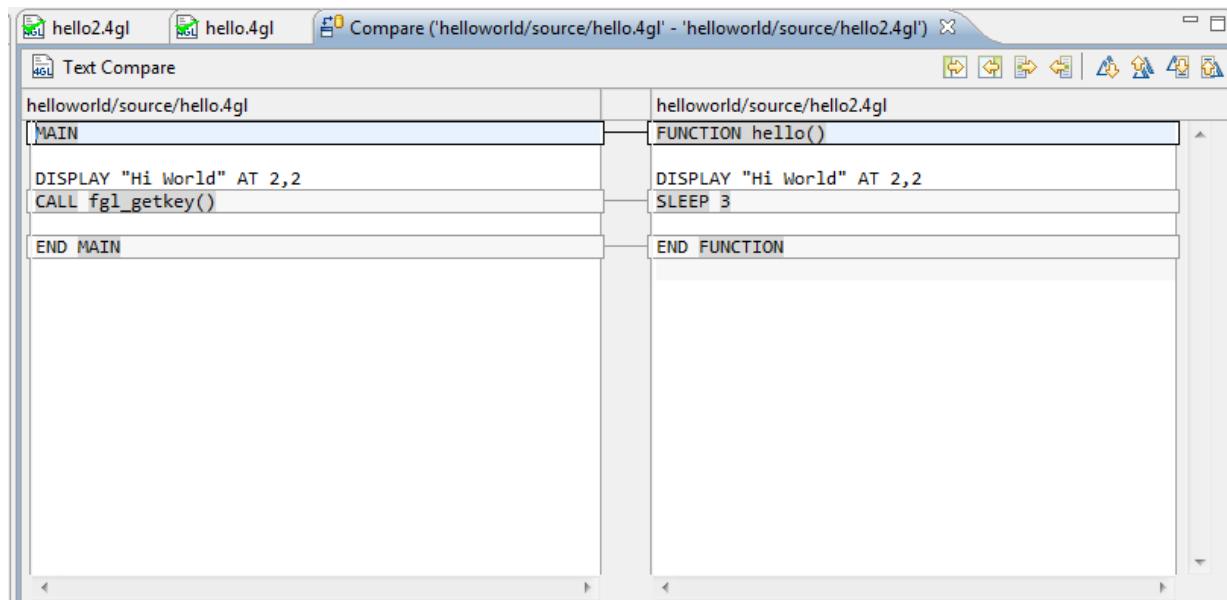


```
FUNCTION hello()
DISPLAY "Hi World" AT 2,2
SLEEP 3
END FUNCTION
```

3. Save the file.
4. Select both "hello" and "hello2" files in the 4GL Project view, right-click one of the selected files and choose the **Compare With -> Each Other** option:



5. The Compare editor will be opened displaying the both files:





6. The Compare editor toolbar will contain two new options which are not available when a file is compared to its history:
 - a. The button copies all the non-conflicting changes from the left pane into the right pane.
 - b. The button merges changes in two files by copying the highlighted change in the left pane into the highlighted fragment on the right. This will overwrite the highlighted fragment in the right pane.

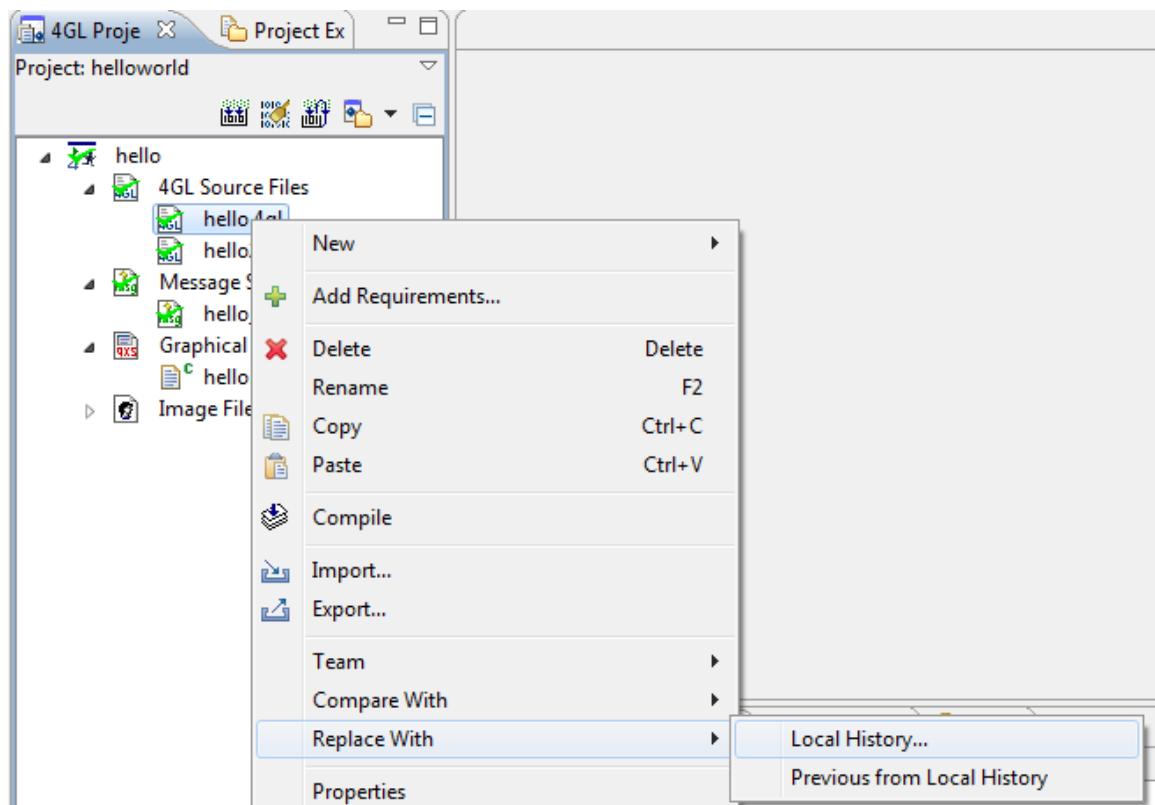
When two graphical form files are compared, they are transformed into special text code, because only text can be compared by means of the Compare editor. However, this code is not the 4GL code, thus it is advisable to compare form files in .per formats.

Replacing a source file

If an earlier revision of a source files needs to be restored, it can be done using the **Replace With** option.

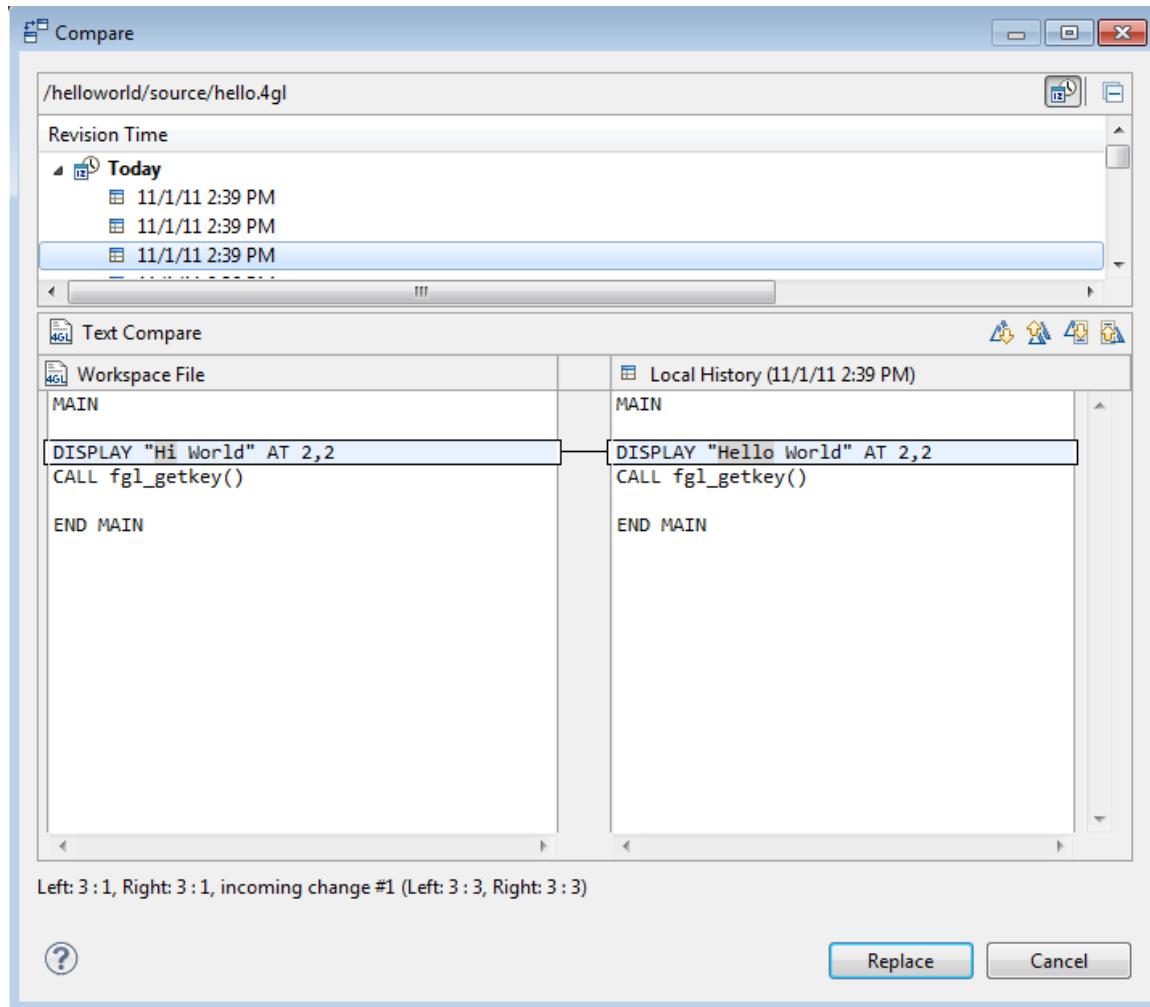
To replace a source file with its earlier revision from the local history, do the following:

1. Right-click the "hello" source file (or any other source file that needs to be restored) in the 4GL Project view and select the **Replace With -> Local History...** option:





2. The replace dialog will be opened where you will find the list of revisions of the source file.
3. Double-click on the previous revision of the source file. The compare editor will be opened in the pane below:



4. Press the **Replace** button to replace the source file with the selected revision.

Using libraries

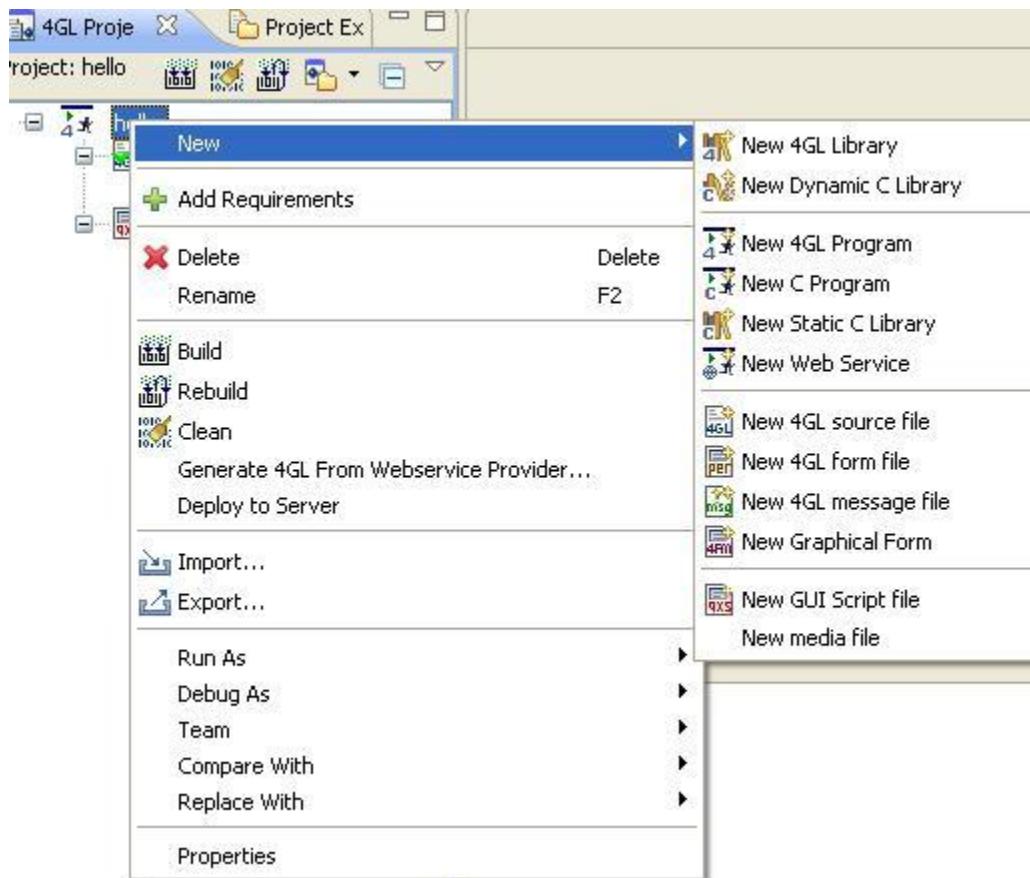
A 4GL program can have dependencies on different kinds of libraries, these can be 4GL libraries, static C libraries, or dynamic C libraries. A 4GL program can have a dependency on a library that is defined in the same project or on a pre-built library that exists outside the current workspace.

To create a library:

1. Make sure that your "helloworld" project is present in LyciaStudio
2. Right-click on the "hello" program to display the context menu.



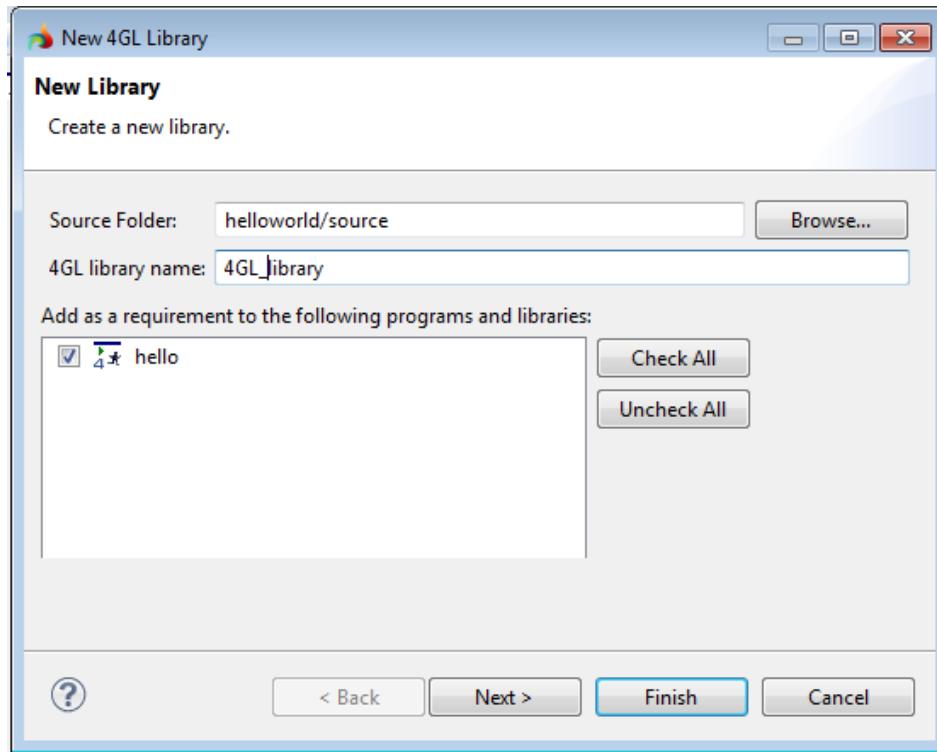
3. Here you can find options to create a 4GL library, Static C library or Dynamic C library:



4. Select the **New 4GL Library** option.

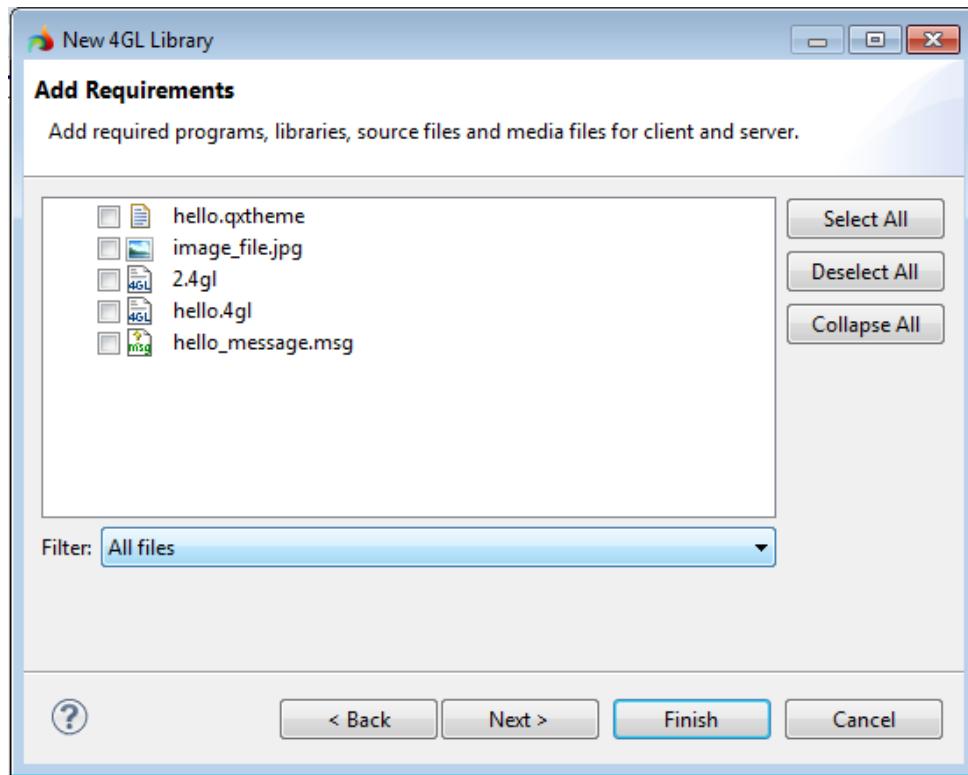


5. Enter the name for the new library and select the "hello" program. Press the **Next** button:





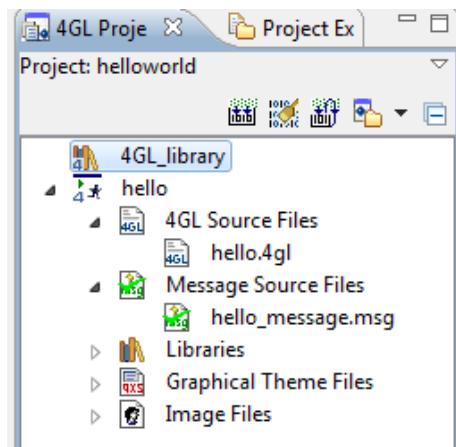
6. Here you can add any files existing in your workspace to the library:



7. Do not select any files and press **Finish**.



8. The library will be created and displayed in the 4GL Project view:

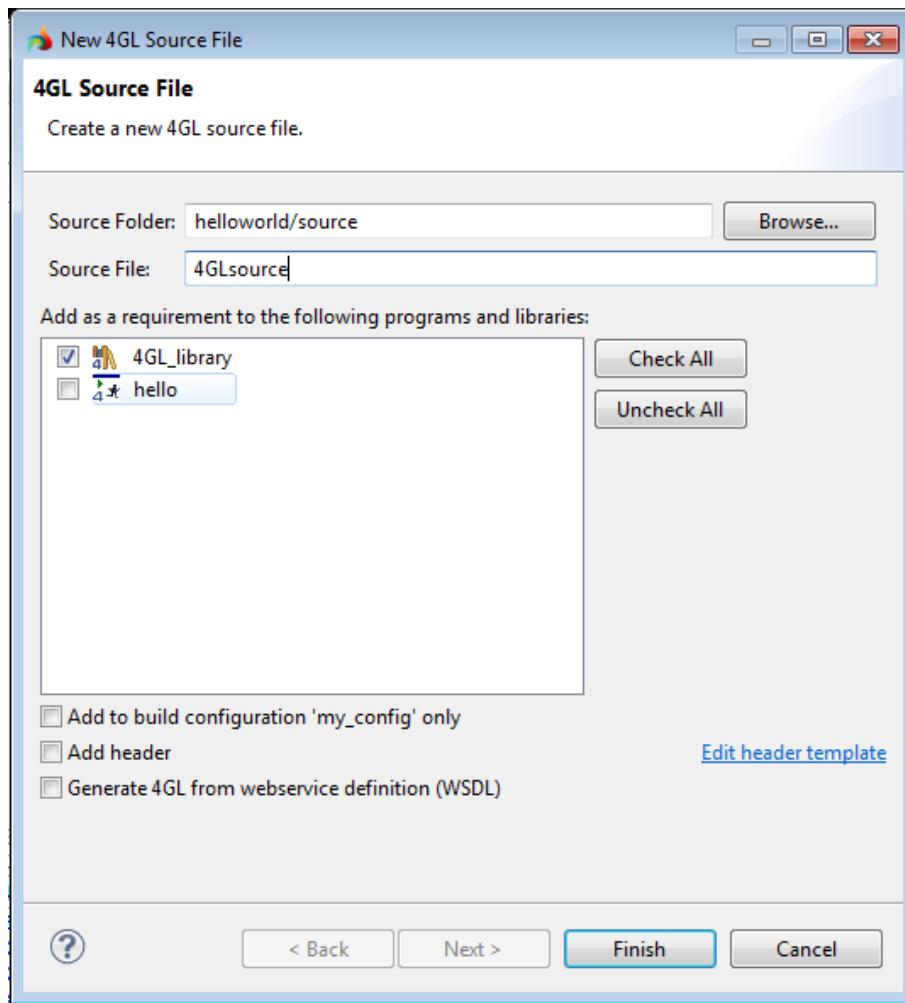


As you have selected the hello program, the library is automatically added to the requirements of the program which can be seen in the Libraries section.

9. The library is now empty. To create a file within this library use the **File -> New** menu option and select the file type you want to create (e.g. 4GL source file)



10. In the New File dialog tick off the 4GL library to include the new file into the library and press **Finish**:

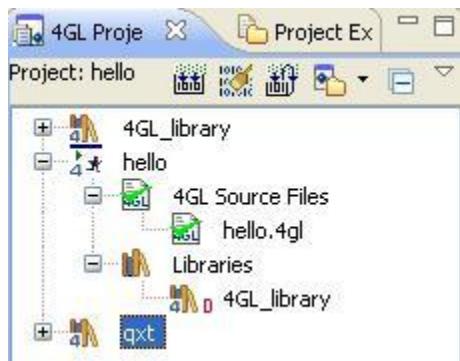


Dynamic and Static C libraries are created likewise.

Linking an internal library

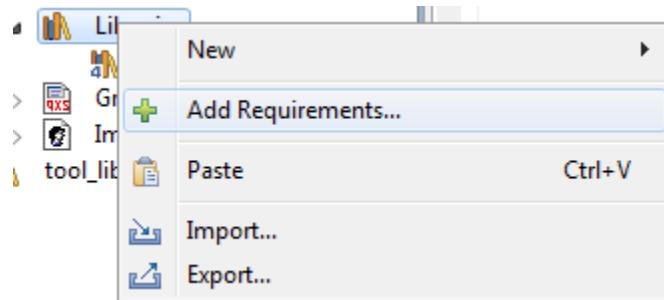
If a library is not created within a project, but you still need to use it, you may import a library into a project.

1. Import a library into the "helloworld" project as described in the "Importing Files, Folders, Programs, and Libraries into an Existing Project" section of this document:

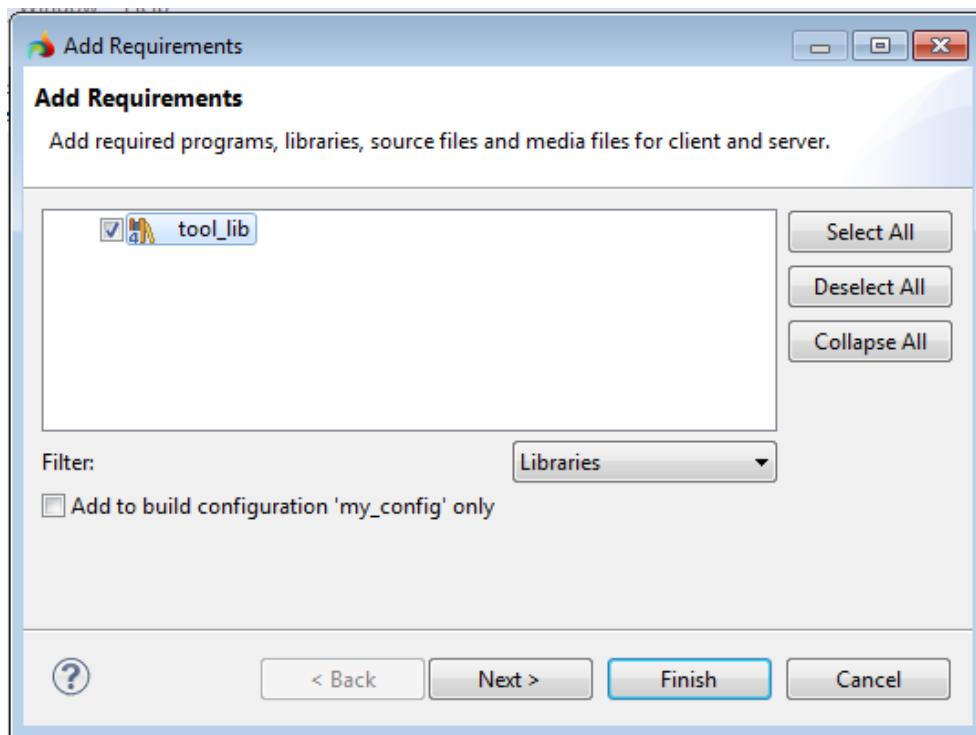


As you can see, the imported library is not added to the requirements of the program automatically. To use this library in the program you need to link it

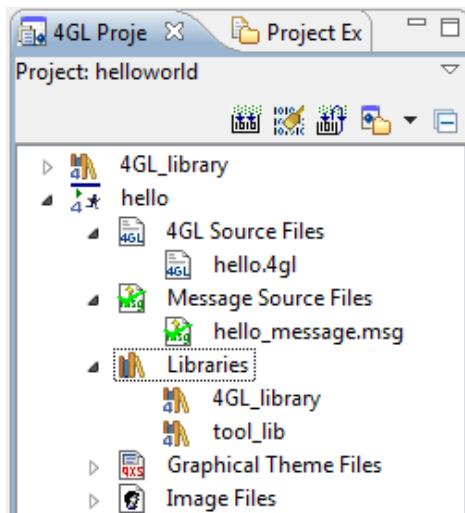
2. Right-click on the "Libraries" section of the "hello" program in the 4GL Project View. Select the **Add Requirements** option from the context menu:



3. The libraries and other files which are located in the workspace but are not linked to the program will be displayed in the dialog box. Select the library and press the **Finish** button:



4. The link to the library will appear in the "Libraries" section of the program:



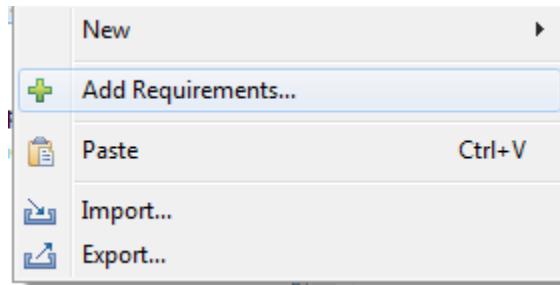
Linking an external 4GL library

If you have a compiled 4GL library outside the workspace and do not want to import it, you can link such library as an external 4GL library.

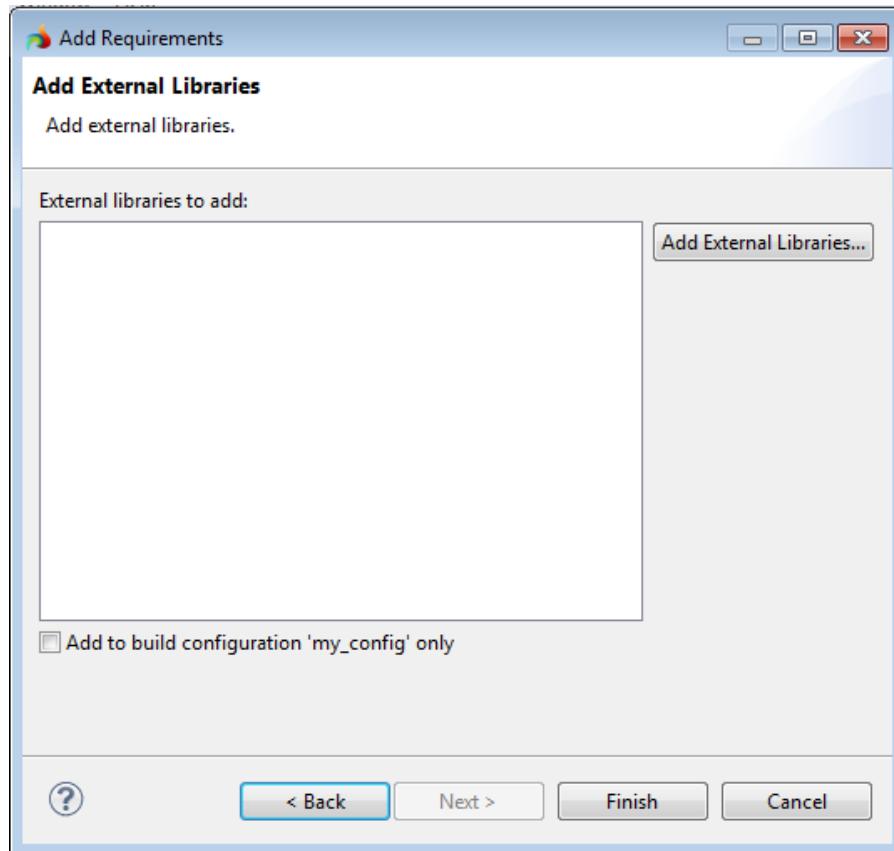
To link an external 4GL library:



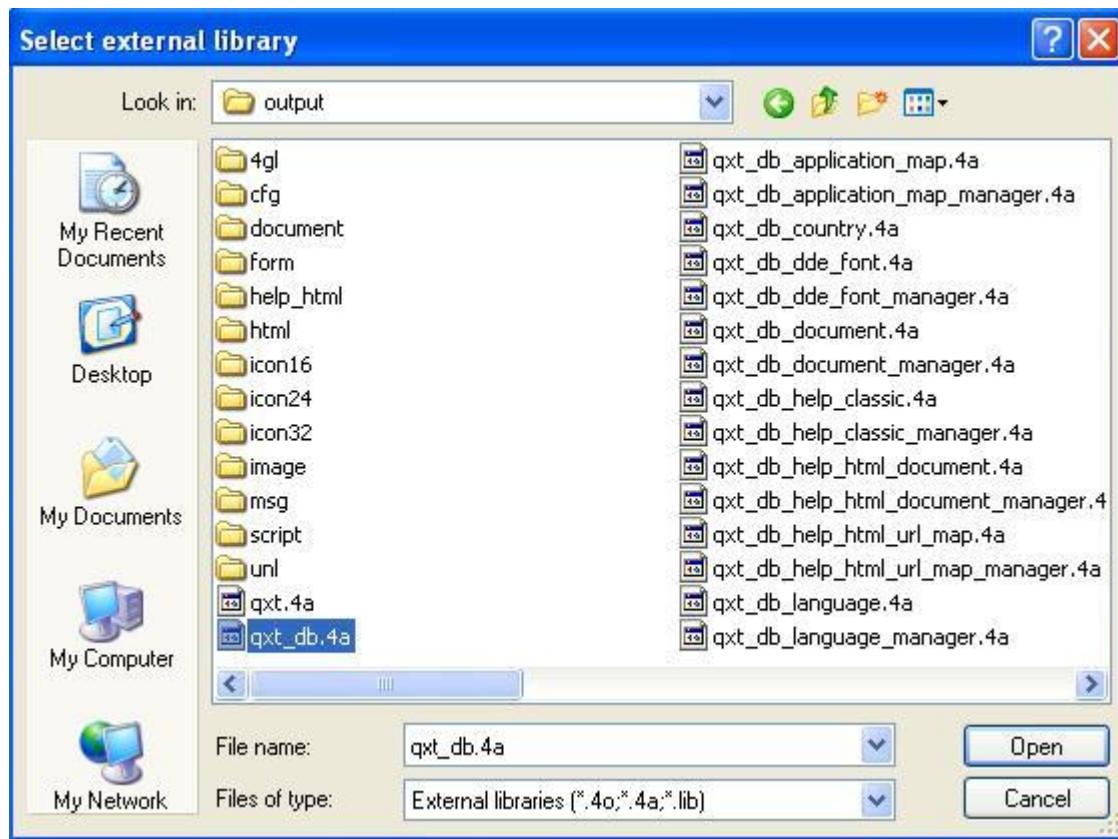
1. Right-click on the "Libraries" section of the "hello" program in the 4GL Project View. Select the **Add Requirements** option from the context menu:



2. In the Add Requirements dialog do not select any resources and press the **Next** button.
3. In the next page of the dialog, press the **Add External Libraries...** button:



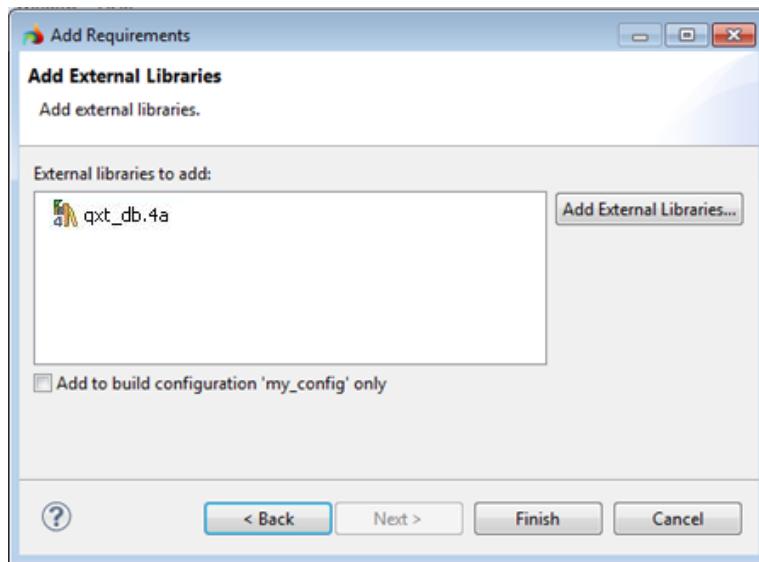
4. In the Select external library dialog, select a .4o or .4a file and press **Open**.



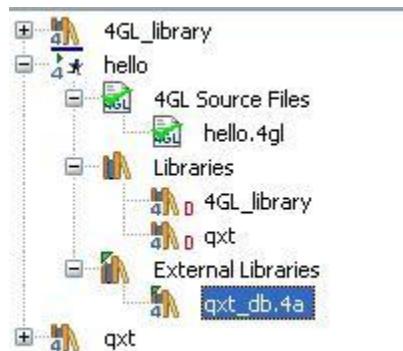
A .4o file is a 4GL object file created after compilation of a single 4GL source file. A .4a file is created by linking several 4GL source files into a module that can be executed directly from the operating system. These files can usually be found in the "output" folder of a 4GL project. They can be also created by the qfql and qlink command line tools of Lycia.



5. The selected file will be displayed in the list of external libraries:



6. Press the **Finish** button. The external 4GL library will be added to the External Libraries section of the "hello" program:

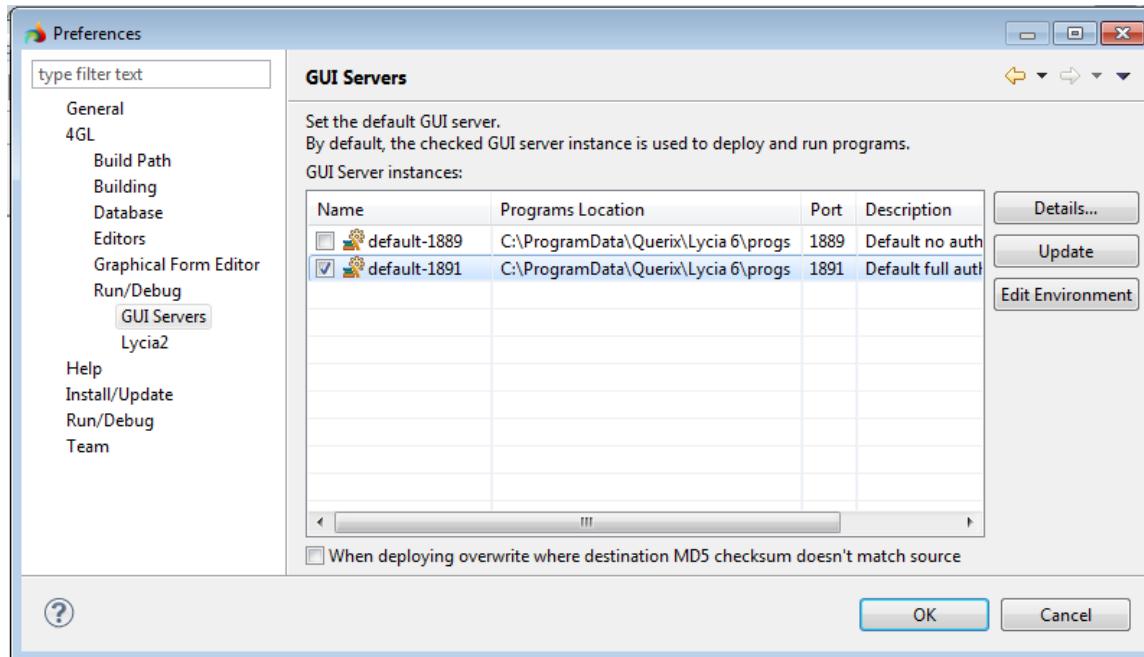


An external library is not added to the workspace and you cannot view or change the resources of this library.

7. In order to deploy an external 4GL library to the GUI server when the program is run, the path to that library must be added to the LYCIA_PATH environment variable. To open and edit the environment of a GUI server, go to **Window -> Preferences -> 4GL -> Run/Debug -> GUI Servers**.



8. Select a GUI server and press the **Edit Environment** button:



9. Press **OK**. The inet.env file will be opened in the editor area.

10. Add the LYCIA_PATH variable with the path to the external library:

```
LYCIA_DIR=C:\Program Files\Querix\Lycia Development Suite 6.0\Lycia
MSGPATH=C:\Program Files\Querix\Lycia Development Suite 6.0\Lycia\msg
LINES=24
COLUMNS=80
LYCIA_DRIVER_PATH=C:\Program Files\Querix\Lycia Development Suite 6.0\Lycia\lib
FGLPROFILE=C:\Program Files\Querix\Lycia Development Suite 6.0\Lycia\etc\fglprofile.std
QXDEBUG!=
QXBREAKCH_START="+-*"
QXBREAKCH_END=":"
PATH=C:\Program Files\Querix\Lycia Development Suite 6.0\Lycia\bin;C:\Program Files\Querix\Lycia\lib;BIInterface\activation-
PATH=$PATH;C:\Program Files\Java\jre6\bin;C:\Program Files\Java\jre6\bin\client
LYCIA_PATH:=C:\work\workspace\cms\output
```

11. Save the file and close it.

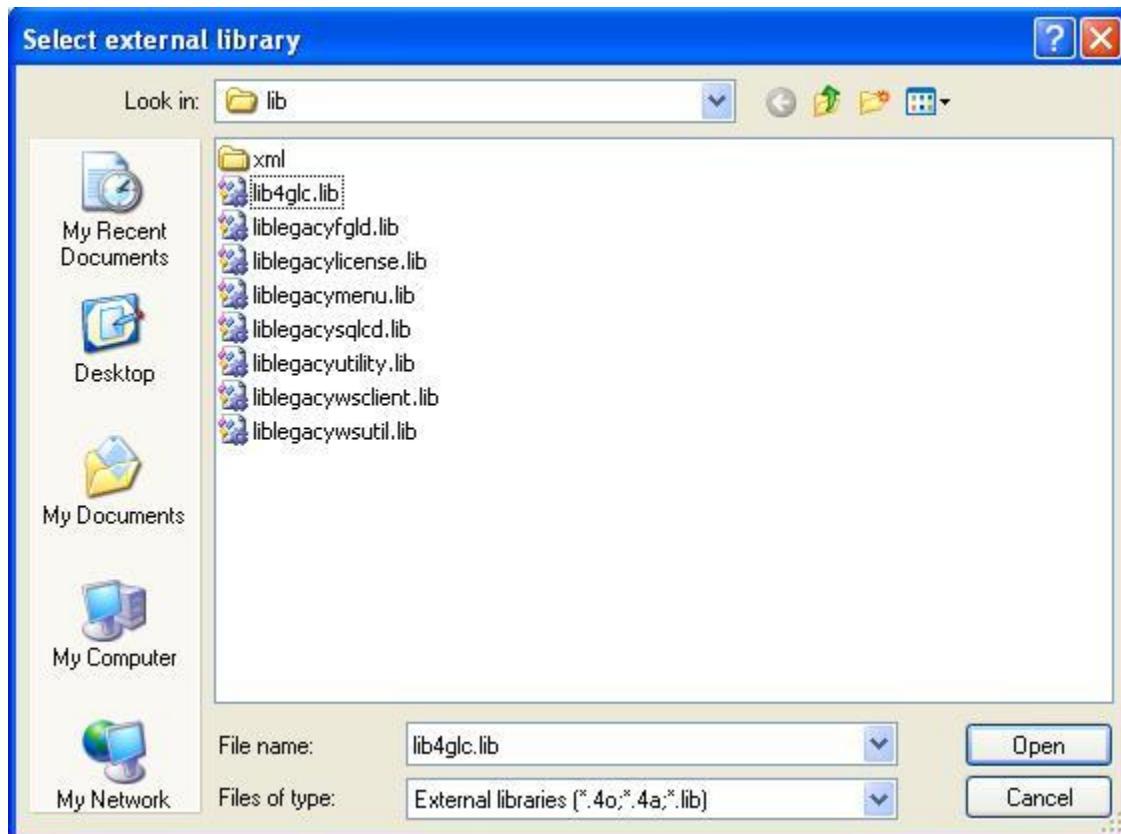


	<p>Note: The LYCIA_PATH variable can be used not only for external libraries, but also for all the dependencies between the build targets located in different directories.</p>
--	---

Linking an external static C library

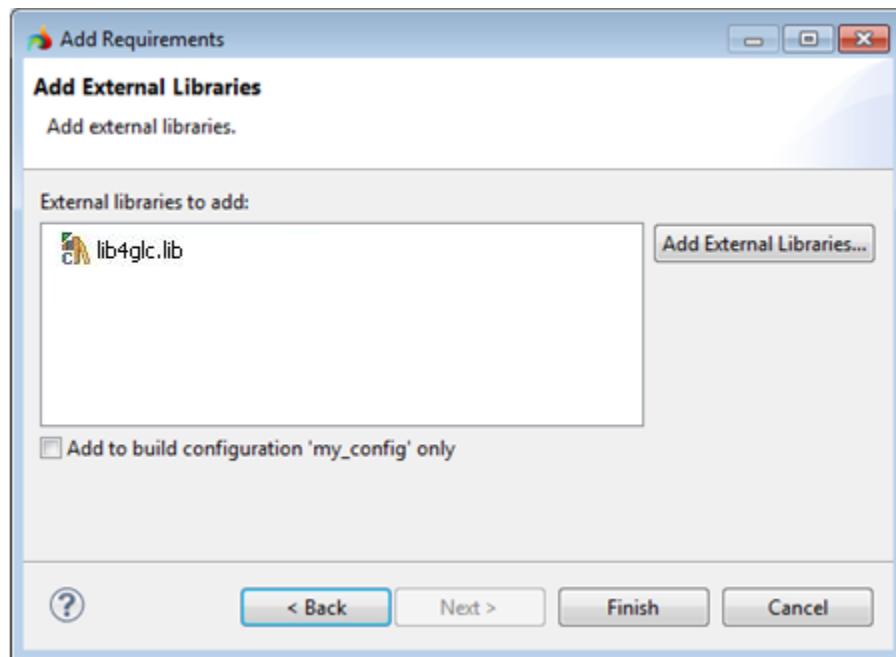
A static C library can be externally linked to a 4GL program in the same way as an external 4GL library.

1. Repeat steps 1-3 of the external 4GL library linking method
2. Select a .lib file located anywhere on your system and press **Open**:

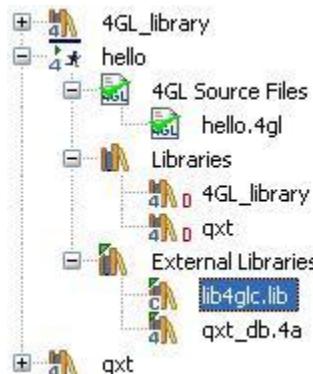


The .lib format for the static libraries is used when LyciaStudio is run on Windows system. The system dependant extensions are used in the filter for other operational systems.

3. The selected file will be displayed in the list of available external libraries:



4. Press the **Finish** button. The static C library will be added to the External Libraries section of the program:

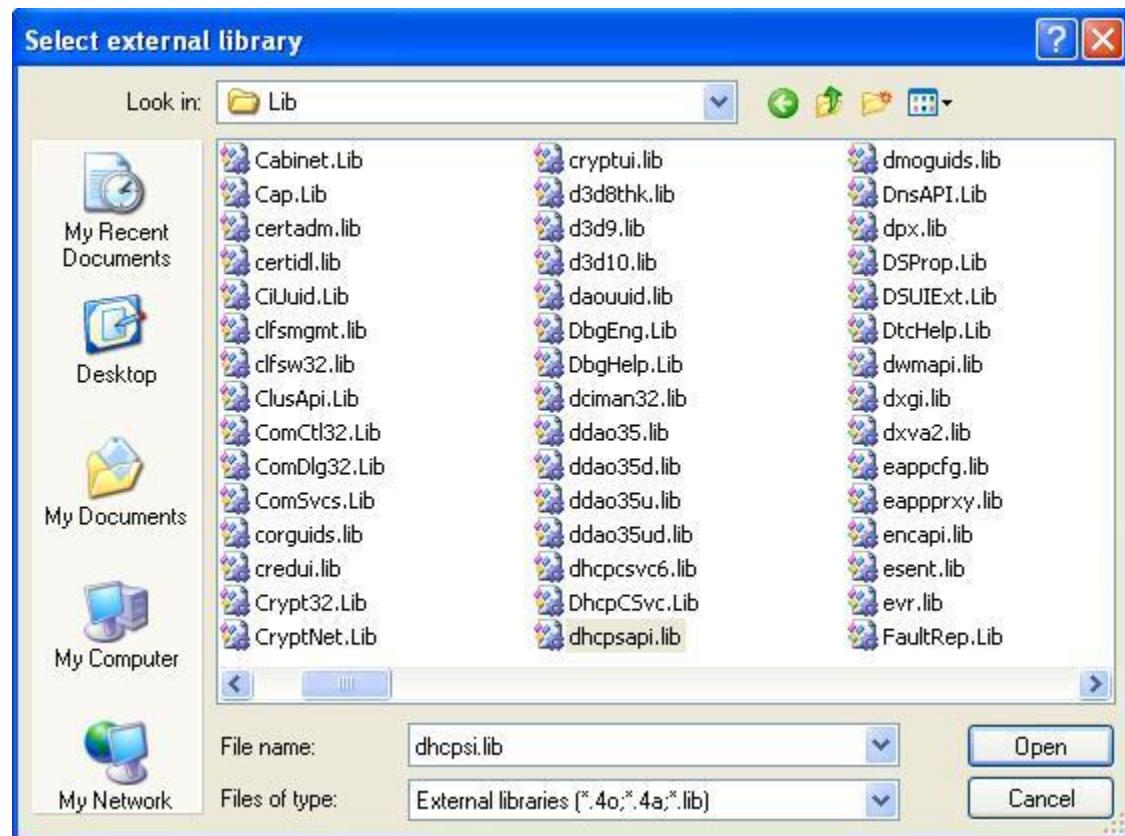


You do not need to add the path to the static library to any environment variable, as the library is added to the program at compile-time.

Linking an external dynamic C library

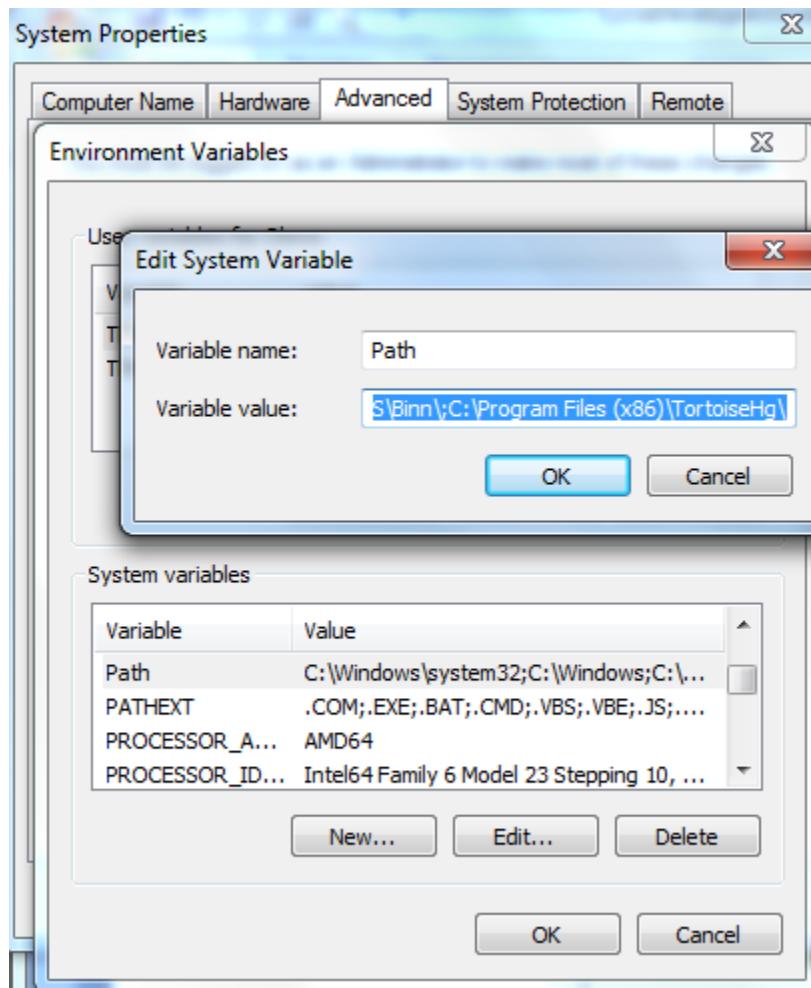
You can link dynamic C libraries to a 4GL program in the following way:

1. Repeat steps 1-3 of the 4GL library linking method
2. In the Select external library dialogue select an import library file for your dynamic library (.lib) and press **Open**.



You cannot link a .dll file in Windows directly, you need to generate an import library .lib for the dynamic library and link this import library. For UNIX/Linux you must link the .os file directly.

3. The selected file will be displayed in the list of external libraries. The linked library will be displayed in the 4GL Project view.
4. The lookup mechanism used for dynamic C libraries is the standard lookup mechanism for the platform. See your system documentation for the details.
 - a. In Windows the path to the .dll file must be added to the PATH environment variable of your system.



- b. In UNIX/Linux you need to add the path to the .os file in the LD_LIBRARY_PATH environment variable.

Installing new features and updating

LyciaStudio supports automatic updates. You can also install some additional features which you need for comfortable work. LyciaStudio is compatible with most of the eclipse plug-ins. They can be installed using the eclipse update sites.

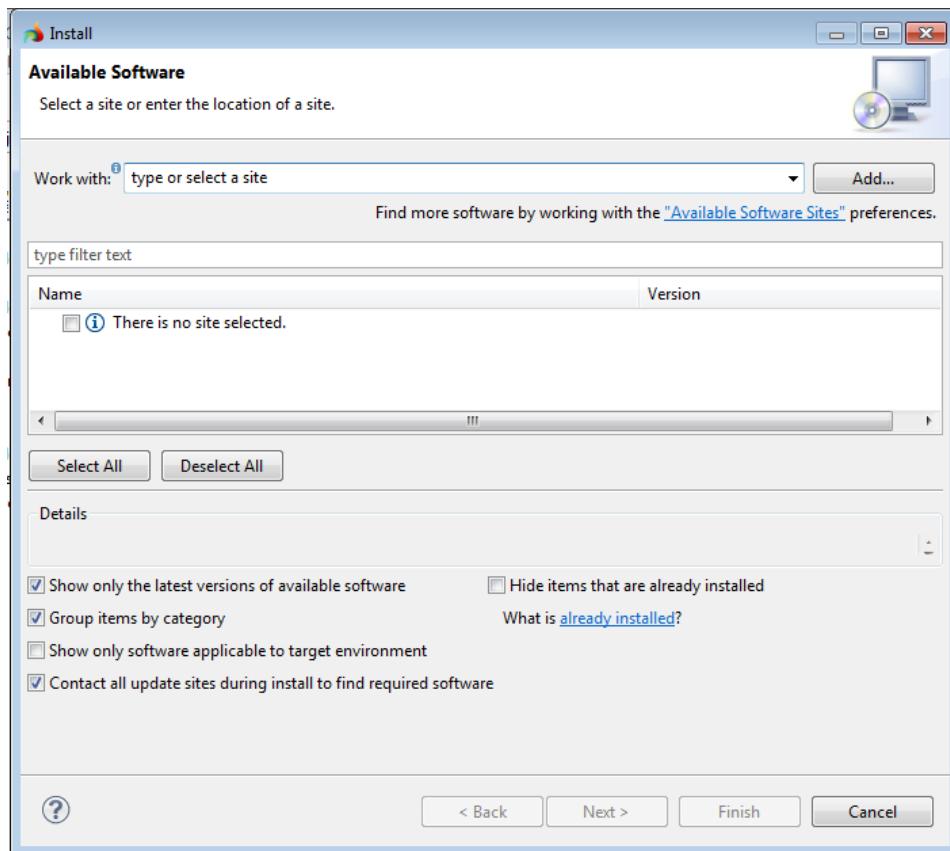
Installing a new plug-in

You can install a new plug-in to enhance the performance of LyciaStudio. For example, you can install the plug-in for working with the SVN repository, if you do not want to use the CVS repository, for which LyciaStudio offers a built-in client. For the details about using the CVS client within LyciaStudio see "[Working with Repositories](#)" chapter of this manual. Working with the SVN options offered by the plug-in is very much alike.

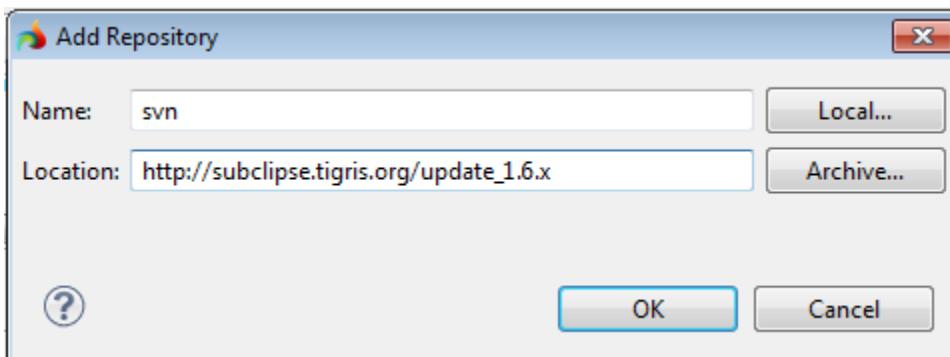
To install an SVN plug-in take the following steps:



1. Click **Help -> Install New Software** in the main menu.
2. You will be presented with the dialog which will prompt you for the site from which you want to download the plug-in:



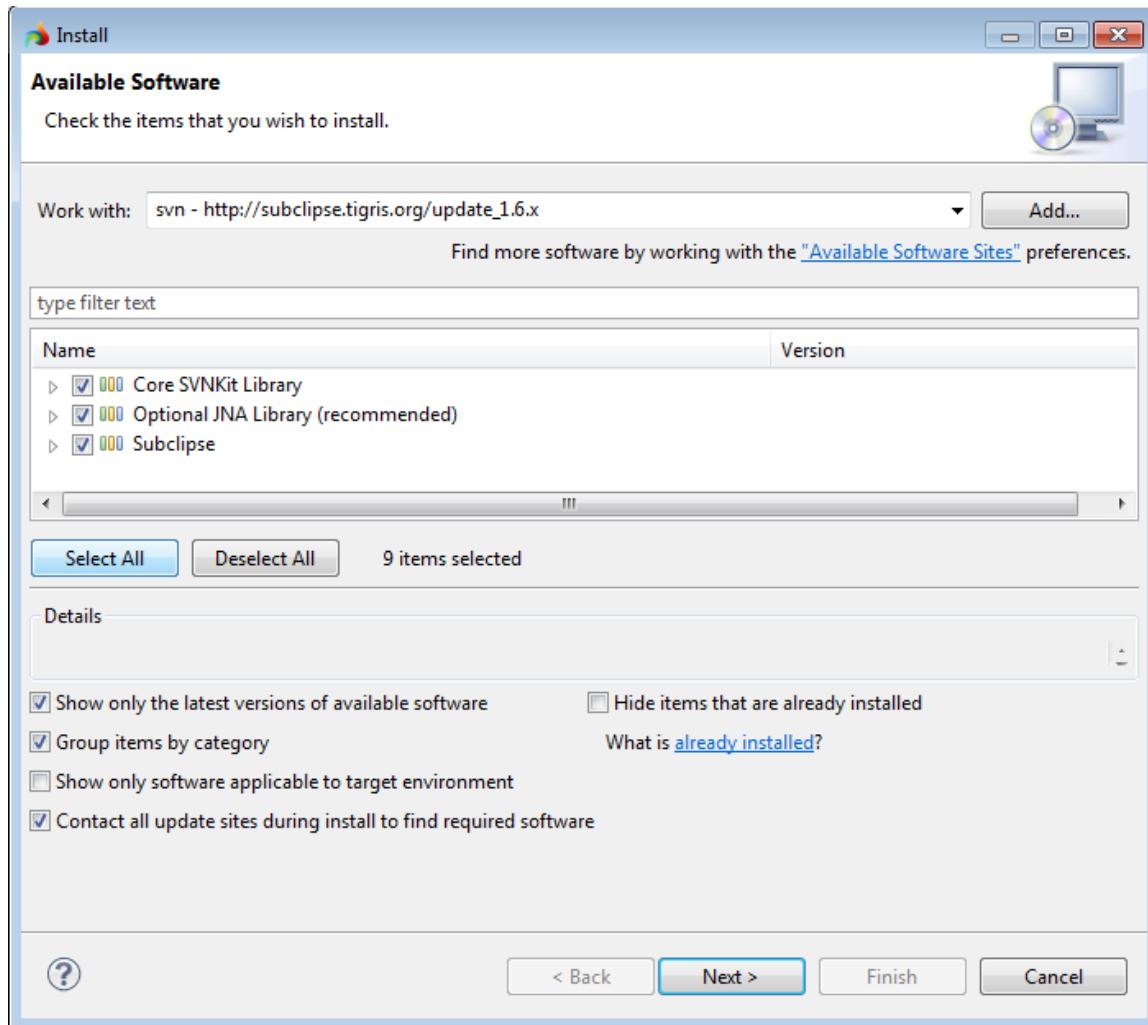
3. Lets presume you have no software sites configured. Click **Add...** to specify the update website from which you want to download the SVN plug-in.
4. In the dialogue box enter the name for the software site and its address:



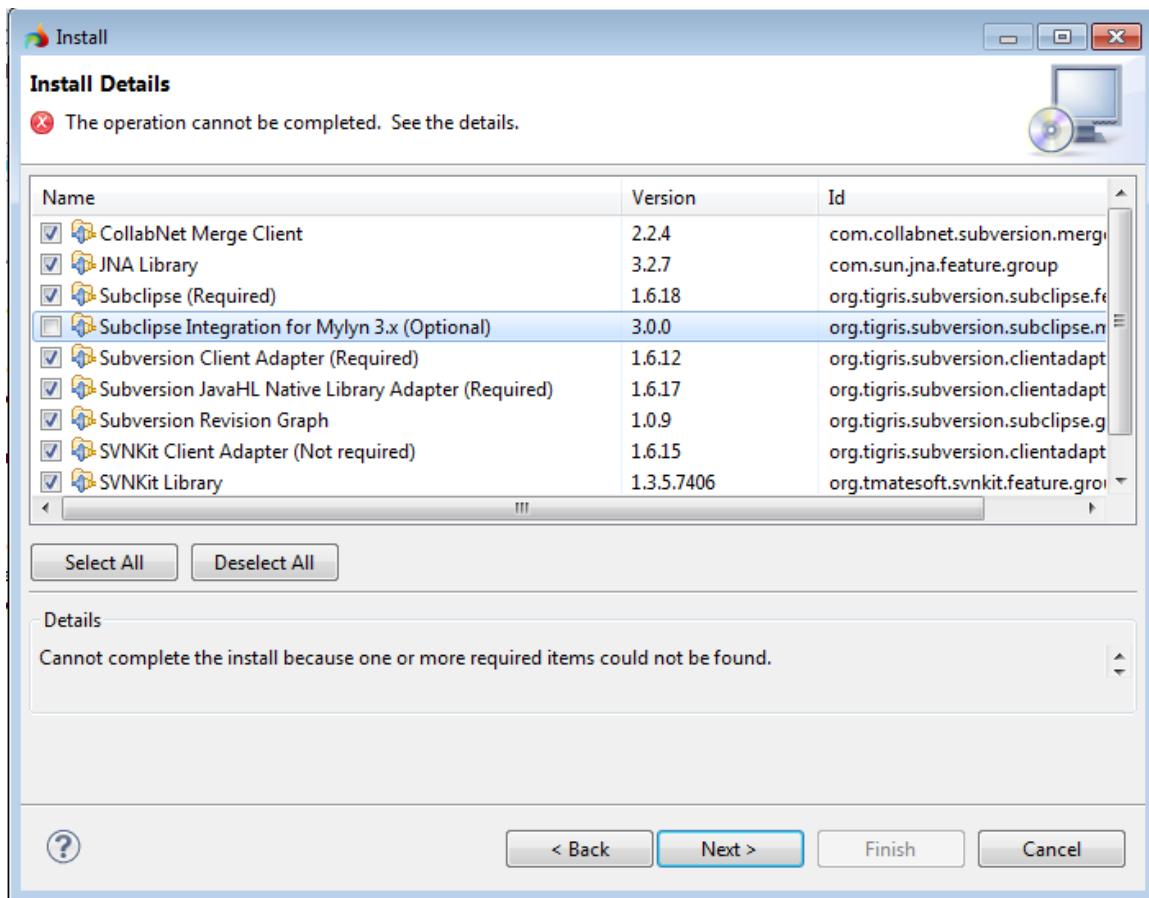


For the demonstration purpose we entered the name of the sbuclipse plug-in update site (http://subclipse.tigris.org/update_1.6.x). The address of the sbuclipse plug-in update site can be found on subclipse.tigris.org, at the Download and Install page. Click **OK**.

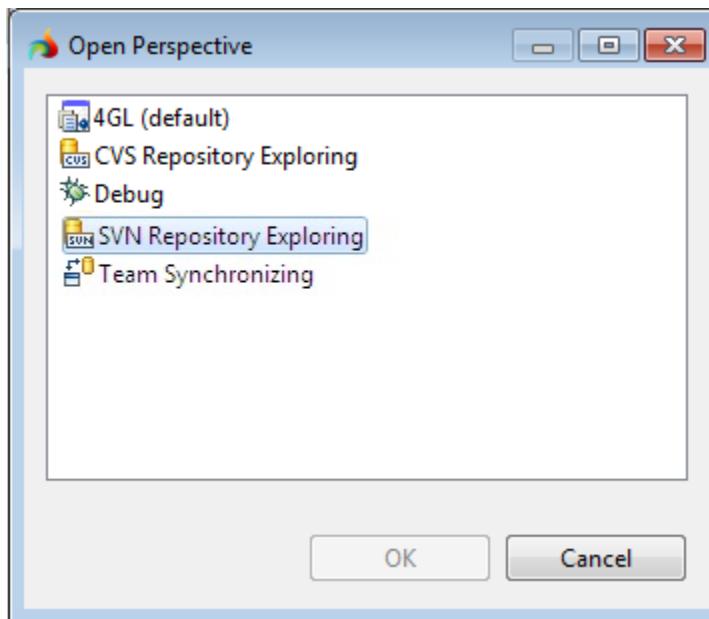
5. Then select the added site from the combo box at the top of the dialogue. Then you will see the features available for installation from the selected update site:



6. Check the features you want to install and then click **Next**.
7. The details about each component of the plug-in to be installed will be displayed in the next dialogue page. In the case with this svn plugin, you may need to remove the check from the Mylyn integration component in the list, as it is shown in the picture below, because LyciaStudio does not support this Eclipse component by default.



8. Click either **Next** button or **Finish** button, depending on which of them is active. If the plug-in requires the accepting of the license agreement, you will need to press **Next** and accept it, otherwise, the **Finish** button will be active.
9. The installation will begin. Once the new features have been downloaded and installed successfully, Lycia will offer you to restart the Studio in order that the changes took effect, or to apply the feature without the restart. It is advisable to make a restart.
10. After the feature is installed and applied, it will be integrated into the Studio. In our case the SVN Repository Exploring perspective will appear among the list of other perspectives, when selecting **Window -> Open Perspective -> Others...**



11. Using this perspective, you can check out, commit, update and use your SVN repository in any other way.

Please, note that you will also need to install the Subversive connectors which are not distributed with Subversive itself. The current version of Subversive has a connector discovery feature which makes it easy to find one. To do this, go to the subversive preference page at **Window -> Preferences -> Team -> SVN**. This preferences section will appear after the plug-in is installed.

Updating the installation

The Studio can be updated either automatically on regular basis or manually. For both methods you need the update site to be configured as described in the section above. As a rule, the update site for updating the Studio is already configured automatically the first time you launch the Studio. To view or change the update site go to **Window -> Preferences -> Install/Update -> Available software sites** preferences page. The default site used for update is <http://update.querix.com/lyciaide/1.0>.

To update the Studio automatically you need to schedule automatic updates. Then at the time set the Studio will automatically download the updates, or notify you that they are available. See [Scheduling automatic updates](#) for more details about scheduling the automatic updates.

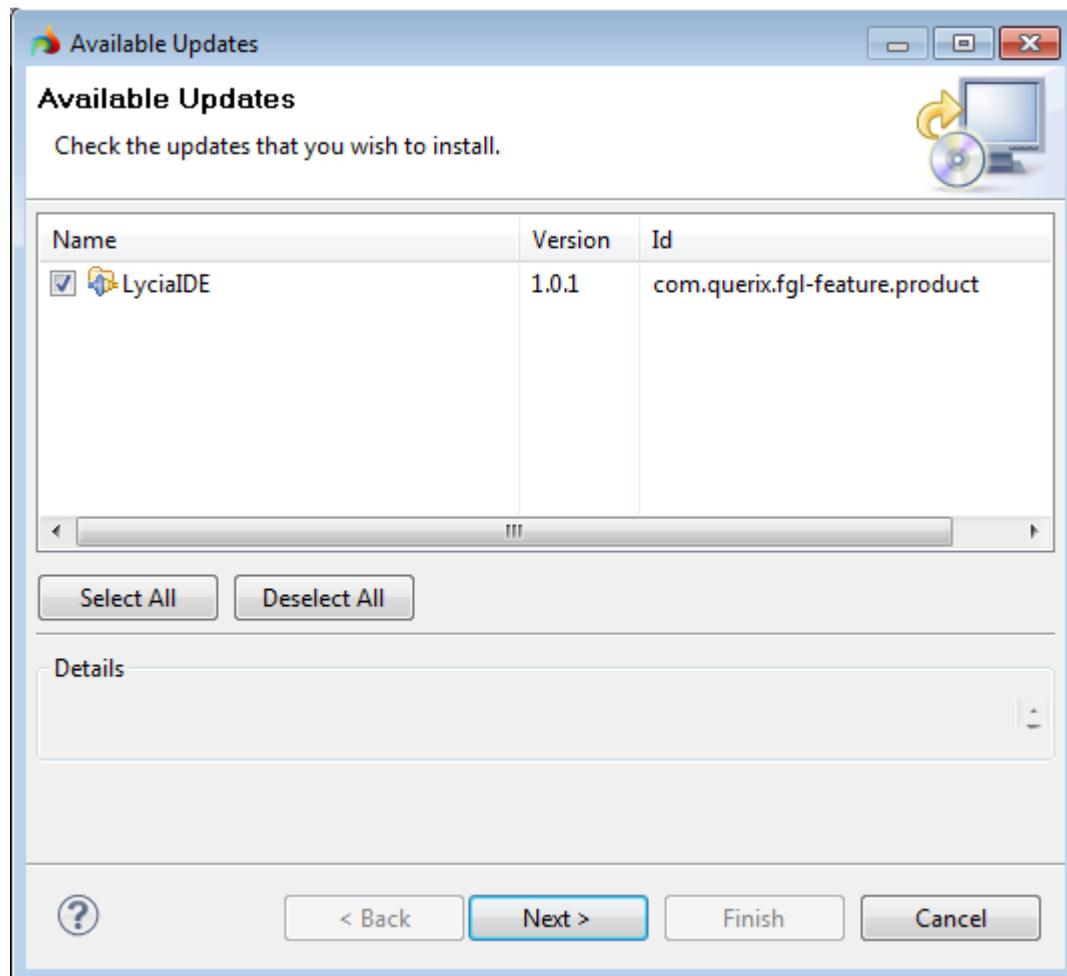
Manual updating

To update the installation manually you need to do the following:

1. Make sure that the update site is configured on [Available software sites](#) preferences page.
2. Go to **Help -> Check for Updates** menu option.



3. The Studio will search for the updates at the specified update site.
4. If any updates are found, you will be presented with the dialogue containing the updates. Click **Next** to update the installation:



5. The next page will display the details of the update. Click **Next**
6. The next page will contain the license agreement. You need to accept the agreement to be able to install updates.
7. Click **Finish** after accepting the agreement. The updates will be installed.
8. You need to restart the Studio for the updates to take effect.



Updating plug-ins

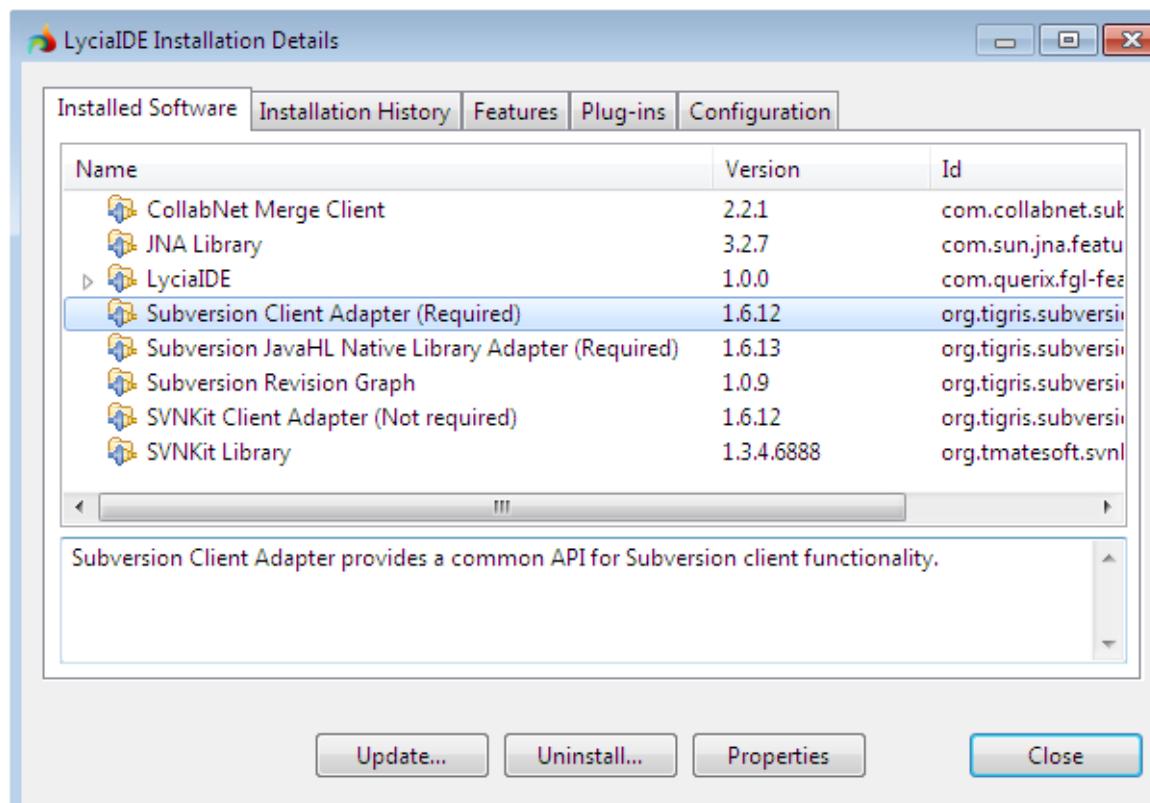
The same method is used to update the plug-ins you have installed. If the site for the plug-in update is configured in the Available software sites, it will also be considered when you select the **Check for Updates** option and the updates, if any, will be offered for installation.

Uninstalling plug-ins

After you have installed one or more plug-ins, you may want to uninstall them at some point. There are two ways to uninstall additional features: to uninstall the features individually, or to revert the installation to the state where they haven't yet been installed. You can uninstall only the additional plug-ins which you have installed as described above, you cannot uninstall any of the plug-ins which constitute the core of the Studio in such a way.

To uninstall a plug-in individually, make the following steps:

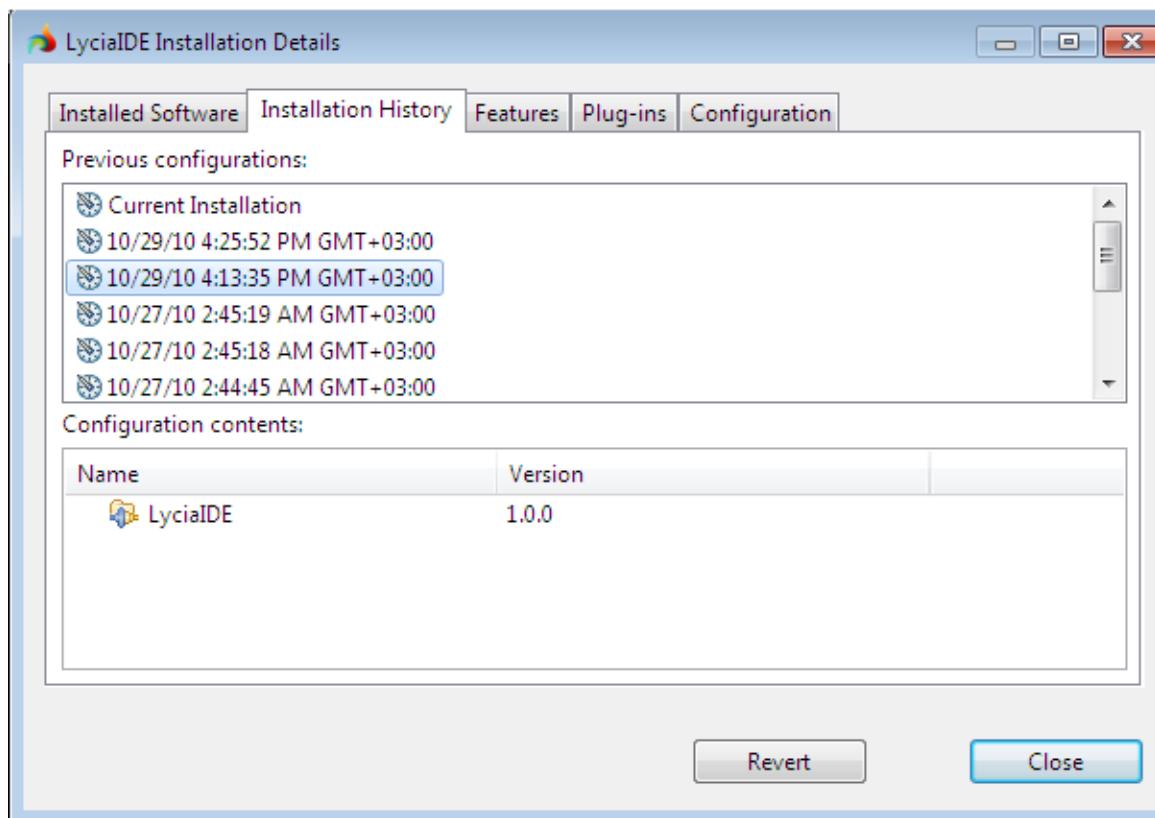
1. Go to **Help -> About LyciaStudio**.
2. In the About dialogue click on **Installation Details** button.
3. In the Installation details dialogue select the Installed Software tab.
4. Then click on the plug-in you want to uninstall and click **Uninstall...** button:



5. You will be asked to confirm the deinstallation. After the confirmation the plug-in will be removed. You will be prompted to restart the Studio for the deinstallation to take effect.

To revert the installation to any of the previous states, you need to do the following:

1. Go to **Help -> About LyciaStudio**.
2. In the About dialogue click on **Installation Details** button.
3. In the Installation details dialogue select the Installation History tab.
4. Select from the list in the upper pane the state to which you want to revert the installation. The features installed at that state will be displayed in the lower pane:



5. Click **Revert**.
6. The installation will be reverted to the selected state and the Studio will prompt you for restart.



Action Defaults

Action defaults allow the user to predetermine the appearance and behaviour of an action and to modify action views.

Besides the text, comment and action image, one have a possibility to define accelerators (up to four) which can trigger an action at runtime and to modify an action default view behaviour under certain circumstances setting values to its optional attributes.

The detailed description of all the possible action defaults attributes with their possible values are given in the next section of this chapter.

Action Default Attributes

When defining an action you must set one mandatory attribute (action identifier). There is also a number of optional attributes, that can be set for an action when modifying its appearance and behaviour.

All the attributes that can be applied to an action and their possible values are described in the table below.

Action Attributes	Description	Possible Values
Identifier	Identifies (names) an action	Alias of String
Text	Defines the text to be displayed in an action view (the button text, etc.)	Alias of String
ActionImage	Defines an image file to be displayed in an action view	Alias of ResourceId
Comment	Defines the default action help text	Alias of String
Accelerator1 Accelerator2 Accelerator3 Accelerator4	Defines the first accelerator keys, that triggers an action	Alias of KeyName
Statical	Defines whether a button can be viewed, even if a default action cannot be triggered at the moment	Alias of Bool. The default value is FALSE.
Validate	Defines whether the data validation takes place	Alias of Bool. The default value is FALSE (inactivates any data validation).
ShowIn Context	Defines whether an action can be	Alias of YesNoAuto:



Menu	rendered from the context menu	<ul style="list-style-type: none">✓ 'auto' makes the context menu option visible provided, that an action itself is visible (the <code>SetActionHidden()</code> method is invoked) and there is no other action view defined✓ 'yes' makes the context menu option always visible provided that an action itself is visible (the <code>SetActionHidden()</code> method is invoked)✓ 'no' makes the context menu option invisible <p>The default value is 'auto'</p>
Default View	Defines whether the default action buttons must be shown in the action defaults bar provided that the 'compat' compatibility property for an application is set	<p>Alias of <code>YesNoAuto</code>:</p> <ul style="list-style-type: none">✓ 'auto' makes an action button visible provided, that an action itself is visible (the <code>SetActionHidden()</code> method is invoked) and there is no other action view defined✓ 'yes' makes an action button always visible provided that an action itself is visible (the <code>SetActionHidden()</code> method is invoked)✓ 'no' makes an action button invisible <p>The default value is 'auto'</p>
Order	Specifies an action view position against another default action views	<p>Alias of INTEGER.</p> <p>In 4GL can be specified using the <code>fgl_setkeylabel()</code> function, one of the arguments of which is intended for setting/modifying the key label position.</p> <p>Is <i>Undefined</i> by default, that means that the button is added to an end of a list of action views.</p>



Action Defaults of Global Scope

In attempt to make the software development process as easy and obvious as possible, we have united and centralized the most often used actions in one configuration file to preserve their conventional appearance and behaviour throughout an application. So, there is no need to specify them every time you define the same action.

Global action defaults parameters are stored in the configuration file named SystemActionDefaults which has the .fm2 extension. The attributes predefined in it are applied to an action, if no other parameters are set to this action throughout an application. Though the action parameters defined in this file are of a global scope, they are of the lowermost priority.

The system action defaults file is included into the Lycia II Development Suite and is installed together with the Lycia Runtime module. After installing Lycia from the installer, both on Windows and Linux, this file can easily be viewed and modified (by default, it is located in C:\Program Files (x86)\Querix\Lydia II Development Suite 6.2\Lydia\etc).

The file has the following syntax (the same syntax must be used when defining an action parameters in an XML source form file or creating your own action defaults file):

```
<Action Identifier="ident_val" Text="text_val"
Comment="comment_val" Statical="val" Order="val" Validate="val">

    <ActionImage Uri="uri_val"/>
    <Accelerator1 KeyValue="key1_val" ControlModifier="yes"
AltModifier="yes" ShiftModifier="yes"/>
    <Accelerator2 KeyValue="key2_val" ControlModifier="yes"
AltModifier="yes" ShiftModifier="yes"/>
    <Accelerator3 KeyValue="key3_val" ControlModifier="yes"
AltModifier="yes" ShiftModifier="yes"/>
    <Accelerator4 KeyValue="key4_val" ControlModifier="yes"
AltModifier="yes" ShiftModifier="yes"/>
    <ShowInContextMenu>YesNo</ShowInContextMenu>
    <DefaultView>YesNo</DefaultView>
</Action>

[...]

</Actions>
```

where:

1. *ident_val* is an action name,
2. *text_val* is the text for an action view,
3. *comment_val* is an action help text,
4. *val* is one of the possible values for the corresponding parameter,
5. *uri_val* is an uri to an image file,
6. *key1-4_val* is an accelerator key value.

Here is the extract from the SystemActionDefaults.fm2 file to illustrate how the parameters for an action default are set globally:

```
<Actions>
```



```
<Action Identifier="interrupt" Text="Interrupt"  
Comment="Interrupt">  
    <Accelerator1 KeyValue="Interrupt"/>  
    <ShowInContextMenu>Yes</ShowInContextMenu>  
    <DefaultView>Yes</DefaultView>  
</Action>  
  
</Actions>
```

As it can be viewed from the example under, the Interrupt action has several globally predefined attributes. If no other parameters are set to it within an application, in an application it appears as follows:

- an action view is shown in the context menu and default view bar,
- an action can be triggered with the help of the PAUSE/BREAK key,
- an action view is labeled as "Interrupt" with the same text as a comment.

Accelerator Attribute

The accelerator attribute is an optional one. If it is not defined within an application, the default accelerator keys predefined in the configuration file are set by the runtime system.

There is a possibility to set up to four accelerator keys which are able to trigger the same action. These attributes can be defined globally in the System Action Defaults file, within the form files or in the 4GL source files. To do this you should specify the Accelerator1, Accelerator2, Accelerator3 or Accelerator4 key values for the accelerator attribute.

When there is an evident necessity of an action to be triggered by no accelerator key, you should specify the 'none' value for an accelerator key value parameter.

This table lists all the keyboard accelerator key names for the actions defaults defined globally in the System Action Defaults file described in the previous section of this chapter:

Action Default Identifier	Accelerator Key Value	Action Default Identifier	Accelerator Key Value
append	CTRL-F1	upfield	UP
insert	F1	downfield	DOWN
update	Undefined	firstrow	HOME
delete	F2	prevrow	UP
accept	ESCAPE	nextrow	DOWN
cancel	PAUSE/BREAK	lastrow	END
help	ALT-W	nextpage	NEXT PAGE
close	ALT-F4	prevpage	PRIOR PAGE
next	Undefined	nexttab	CTRL-TAB
previous	Undefined	prevtab	CTRL-SHIFT-TAB
first	Undefined	selectnextrow	SHIFT-DOWN
last	Undefined	selectprevrow	SHIFT-UP
print	Undefined	highlightnextrow	CTRL-DOWN
find	CTRL-F	highlightprevrow	CTRL-UP
findnext	CTRL-G	selectmenu	SPACE ENTER



editcopy	CTRL-C	nextmenu	RIGHT
editcut	CTRL-X	prevmenu	LEFT
editpaste	CTRL-V	expand	RIGHT
nextfield	TAB RIGHT ENTER	collapse	LEFT
prevfield	SHIFT-TAB LEFT	copyvisiblecolumn	Undefined
copycell	Undefined	copyvisibletable	Undefined
copyrow	Undefined		

where:

RIGHT, LEFT, UP, DOWN	the keys of the arrow keyboard group;
HOME, END, NEXT PAGE, PRIOR PAGE	the keys of the navigation keyboard group;
CTRL, SHIFT, ALT	the modifier keys;
F1-F12	the functions keys;
A-Z	the letter keys;
Undefined	no Accelerator Key Value defined.

Precedence of Action Defaults Parameters

As it was mentioned above, the action parameters can be set by the user explicitly or predefined in the global action defaults. The attributes defined on different program levels may override or complement each other.

The priority of this or that attribute set on different levels for the same action is of high importance, when merging takes place.

The final attribute value, that affects an action view, appearance and behaviour depends on the following precedence (the levels, where actions can be set, are listed in the descending order):

1. the [field](#) level;
2. the [sub_dialog](#) level;
3. the [dialog](#) level;
4. the [form_field](#) level;
5. the [form](#) level;
6. the [interface](#) level provided that the `LoadActionDefaults()` method is used;
7. the [application](#) level;
8. the [configuration](#) file (of the global scope).

Field Level

The field scoped action can be set in the 4GL source file, and possess the highest precedence of all the others.

Here is an example illustrating how an action can be set for the specific field:

```
MAIN
...
    CALL fgl_setactionlabel("myAct", "newActTitle")
```



```
INPUT f1, f2 FROM field1, field2
BEFORE INPUT
    CALL fgl_dialog_setactionlabel("myAct",
"newNewMyActTitle")

ON ACTION "myAct"
    DISPLAY "myAct"

ON ACTION "fieldAct" INFIELD field2
    DISPLAY "fieldAct"
END INPUT
END MAIN
```

Subdialog Level

An action may be defined in a subdialog, in an ON ACTION clause:

```
DIALOG
    INPUT f1, f2 FROM field1, field2

    ON ACTION "myAct"
        DISPLAY "myAct"

    END INPUT
END DIALOG
```

Dialog Level

The dialog scoped actions can be defined in an ON ACTION clause:

```
DIALOG
    INPUT f1, f2 FROM field1, field2
    END INPUT

    ON ACTION "dialog_act"
        DISPLAY "dialog_act"
    END DIALOG
```

Besides, you can modify an action appearance and behaviour by means of several dialog methods. The attributes set in such a way influence actions defined in a dialog (including all the subdialogs):

- `fgl_dialog_setkeylabel()`

Usage:

```
fgl_dialog_setkeyLabel("f2", "Login", "my_icon", 1, FALSE)

# sets the label and icon name for the action "F2" view, the
action "F2" view position (order attribute) and statical attribute
```



- SetActionHidden()

Usage:

```
DIALOG.SetActionHidden("interrupt", TRUE)  
  
# makes the action view for the interrupt action visible
```

- SetActionActive()

Usage:

```
DIALOG.SetActionActive("my_act", FALSE)  
  
# disables the "my_act" action
```

Form field Level

An action for a specific form field can be set by means of the Actions property object available from the Form Designer, Design tab.

Moreover you can add it and set its attributes directly to the XML source relating to the corresponding form using the same syntax as in the System Action Defaults configuration file:

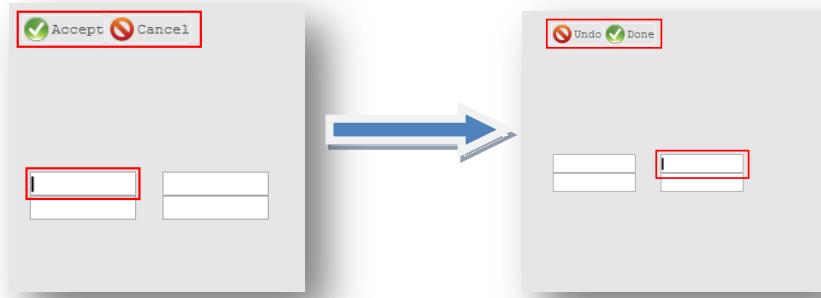
XML source:

```
<form xmlns="http://namespaces.querix.com/2011/fglForms">  
  <textfield identifier="f002" visible="true" enable="true"  
  zOrder="0" fieldTable="formonly">  
    ...  
    <Actions>  
      <Action Identifier="accept" Text="Done" Comment="Done"  
      Order="2">  
        </Action>  
      <Action Identifier="cancel" Text="Undo" Order="1">  
        </Action>  
    </Actions>  
  </textfield>
```

In the extract below the action attributes are set for the textfield with the *f002* identifier, so, the action view for this specific field differs. The view is changed when the *f002* field gets focus.



From the following screenshots you can trace the changes in the actions view taking place at runtime - the label and the order of the Accept and Cancel title bar buttons have been changed:



Form file Level

A form scoped action (as well as a form field scoped one) can be defined by means of the Actions property (on the form level) available in Form Designer, Design tab.

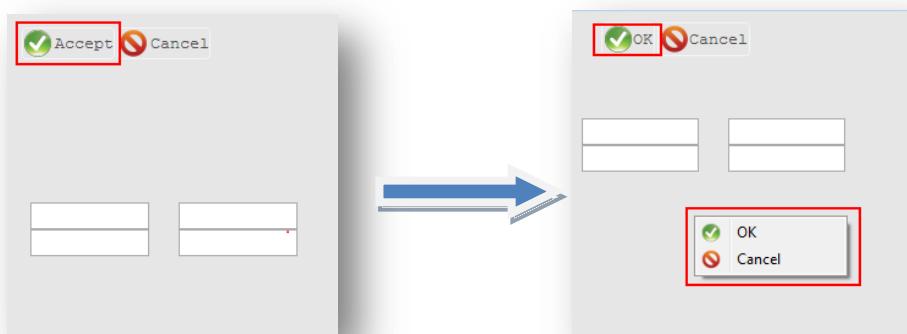
Moreover you can add them directly to the XML source file relating to the corresponding form using the same syntax as in the System Action Defaults configuration file:

XML source:

```
<form xmlns="http://namespaces.querix.com/2011/fglForms">
    ...
    <Actions>
        <Action Identifier="accept" Text="OK">
            <ShowInContextMenu>Yes</ShowInContextMenu>
        </Action>
    </Actions>
</form>
```

The XML source extract above illustrates how the action attributes (the Text and ShowInContextMenu attributes in particular, can be set on the form level).

On the screenshot below you can trace the changes in the application that will take place at runtime:





Interface Level

The action parameters on the interface level are set by means of the LoadActionDefaults() method which is intended for loading the file with the user predefined action parameters. This file should have the same syntax and extension as the System Action Defaults configuration file.

The attributes set in such a way override the action defaults parameters predefined globally in the configuration file.

Usage:

```
ui.Interface.LoadActionDefaults("action_file")
```

Application Level

Application scoped actions are written down to an application file with the .fm2 extension. It is named automatically as <app_name>Actiondefaults.

Application scoped action attributes can be set by means of several methods

- fgl_setkeylabel();
- fgl_setactionlabel().

Usage:

```
fgl_setkeyLabel("f2", "Login", "my_icon", 1, TRUE)  
  
# sets the label and icon name for the action "F2" view, the  
action "F2" view position (order attribute) and statical attribute  
to TRUE.
```

Configuration File Level

The system action parameters predefined in the configuration file possess the lowermost priority.

Below, there is an extract from the System Action Defaults file, illustrating how the attributes are set for the Accept and Cancel actions defaults which appear automatically when the INPUT statement is invoked.

So, if no other attributes for these actions are set explicitly within an application, the following settings will be applied to them:

```
<Action Identifier="accept" Text="Accept" Comment="Accept"  
Order="140000">  
    <ActionImage Uri="qx://embedded/accept.png"/>  
    <Accelerator1 KeyValue="esc"/>  
</Action>  
<Action Identifier="cancel" Text="Cancel" Comment="Cancel"  
Order="150000" Validate="no">  
    <ActionImage Uri="qx://embedded/break.png"/>  
    <Accelerator1 KeyValue="Interrupt"/>  
</Action>
```



The screenshot below illustrates how the attributes set in the configuration file influence the Accept and Cancel action views at runtime:



Action Attributes Merging

When any action appears within an application, the automatic merging of its parameters takes place.

And the principles of the action attributes precedence described in the previous section are of the highest importance, when the program merges the action attributes set on different program levels.

To illustrate how attribute merging works, let consider the following example of obtaining the finite parameters for the *delete* action:

***SystemActionDefaults.fm2* file:**

```
<Action Identifier="delete" Text="Delete" Comment="Delete"  
Order="130000" Validate="no">  
    <ActionImage Uri="qx://embedded/delete.png"/>  
    <Accelerator1 KeyValue="f2"/>  
</Action>
```

***UserActionDefaults.fm2* file** loaded to an application by means of the LoadActionDefaults() method:

```
<Action Identifier="delete" Text="Delete text" Comment="Delete"  
Validate="yes">  
</Action>
```

XML source, the form level:

```
<Action Identifier="delete" Text="MyDelete" Comment="Delete text"  
Order="3" >  
    <ActionImage Uri="qx://newDelete.png"/>  
    <Accelerator2 KeyValue="A" ControlModifier="yes" />  
    <ShowInContextMenu>Yes</ShowInContextMenu>  
</Action>
```

4GL source file, the dialog level:

```
fgl_dialog_setactionLabel("Delete", "MyDelete", 1)
```



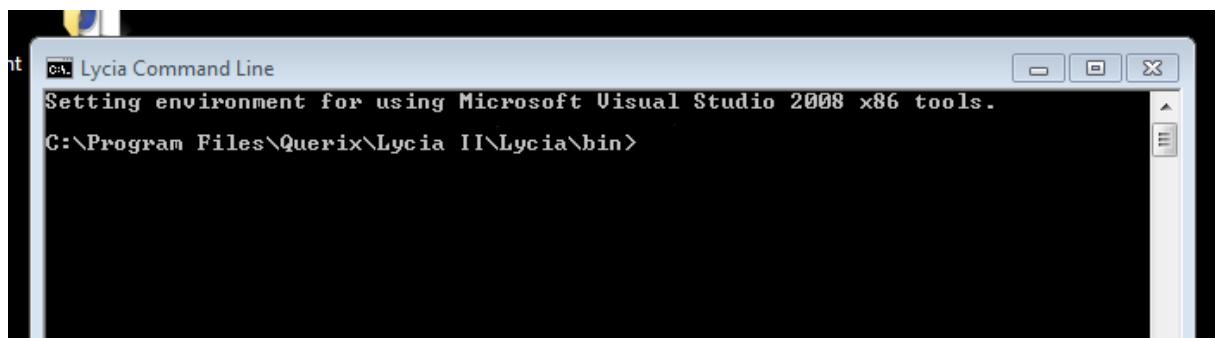
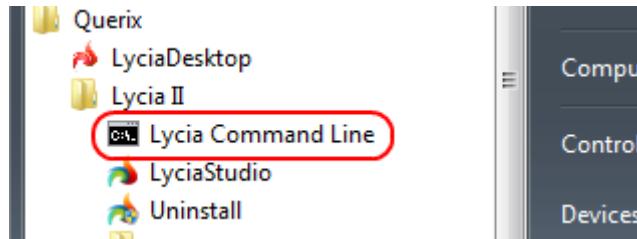
The table below contains the finite attribute values for the *delete* action which are obtained in the result of the merging of the parameters set on different program levels in the extracts given below:

Action Attribute	Finite Value	Level
Identifier	Delete	
Text	Delete	Dialog Level
Action Image	newDelete.png	Form Level
Comment	Delete text	Form Level
Accelerator1	F2	Cfg file
Accelerator2	CTRL-A	Form Level
Accelerator3	None	None
Accelerator4	None	None
Statical	FALSE	Default value
Validate	Yes	Interface level
ShowInContextMenu	Yes	Form Level
DefaultView	Auto	Default value
Order	1	Dialog Level



Command line tools

In Windows the command line environment is invoked from the **Start Menu -> Querix -> Lycia** folder, as shown below:



On Unix, before working with the command line tool, the environment must be set. This can be done as follows:

```
cd <directory where Lycia is installed>  
. environ
```

Take care to note the space between the "." and the "environ".

Lycia has several types of the command line tools, they are:

- Tools for running and compilation 4GL files and applications: qbuild, qexpt, qfgl, qform, qlink, qmsg and 4make. These tools are described further in this chapter.
- Tools for web services compilation: wsmdc, wslink, and wsact. These tools are described in details in the "[Web services](#)" chapter of this guide.
- The auxiliary tools which are used to compile and link C files: esqlc, [qxcc](#) and [qxld](#). These are not actual tools, they are just labels which launch a C compiler. On Windows they launch Microsoft Visual Studio and on UNIX/Linux they run gcc. The qxcc command is



used to compile files and the qxld command is used to link them. The esqlc command is used to compile the C files with embedded SQL code.

- The tool for exporting the database schema: qexpt.

qbuild

Syntax: qbuild [options] ProjectRoot [Target] ...

Where 'options' refers to one or more of:

-?	[--usage]	Display usage information
-V	[--version]	Show version
-v	[--verbose]	Verbose output
-D	[--define] arg	Define property
-C	[--configuration] arg	Configuration
-M	[--build_mode] arg (=build)	Build Mode

qbuild is used to compile the projects or their elements created in LyciaStudio and located in the workspace. It produces one or more executable files or object files and places them into the output folder in the workspace. If only the path to the project is specified, the complete project will be built. If you specify the path to a specific resource (a file or a program) - only this resource will be built.

Usage:

```
qbuild C:\work\ws\project4gl
qbuild C:\work\ws\project4gl program4gl
```

Flag -C is optional and must be followed by the name of the build configuration you want to use for building a project.

Flag -M is optional too, if it is absent the building mode is used by default. If it is present, it must be followed by the "build" keyword. If you want to perform cleaning instead, specify this flag followed by the "clean" keyword.

For building:

```
qbuild -M build c:\users\me\workspace\cms-micro
qbuild c:\users\me\workspace\cms-micro
```

For cleaning:

```
qbuild -M clean c:\users\me\workspace\cms-micro
```



qexpt

Syntax: qexpt [options] database

Where 'options' refers to one or more of:

-o [--output] arg (=.)	Output directory
--format arg (=fgl)	Output format (fgl, xml, sql)
--stdout	Direct output to stdout
-v [--verbose]	Verbose output
-V [--version]	Print version
-? [--usage]	Show usage information
-h [--help]	Show help information
-d [--database-driver] arg	Database driver
--database arg	List database schemas to export

This tool is used to export the database schema together with database data. This enables a quick and clean database migration in case of necessity.

Here is an example which loads the database schema into "test1" folder in SQL format from an Informix database called "stores":

```
qexpt -o /home/informix/Desktop/test1 --format sql -d informix --database stores
```

qfgl

Syntax: qfgl [options] file...

Where 'options' may be as follows:

-? [--usage]	Display this usage information
-V [--version]	Show version
-v [--verbose]	Verbose output
-d [--database-driver] arg	Set database driver for compilation
-e [--encoding] arg	Source file encoding
-o [--output] arg	Output path
-s [--web-service]	Web service compile
--java-option arg	Pass option to JVM

The qfgl command is used to compile a 4gl source file into a p-code module. To compile a 4gl source code file, the following command would be called:

```
qfgl a.4gl
```

which would output a p-code module called a.4o

```
qfgl -o b.4o -d sserver a.4gl
```



would output a p-code module called b.4o, and the 4gl file will be validated against a SQL Server database.

For compiling files using the UTF-8 encoding, -e flag must be followed by "utf8" value.

	<p>Note: the -d option is only used for validation purposes during compilation. The compiled program can be run against any database server available, either by running the application with the -d option, or by setting LYCIA_DB_DRIVER environment variable</p>
---	---

qform

Syntax: qform [options] form-file

Where 'options' refers to one or more of:

--db arg	Set database driver for compilation
--e arg	Set form file encoding
--p arg	Set output path
--scrccompat	Set grid panel as the root form container for SCREEN layout
--v	Show version
--?	Display usage information

The qform command is used to compile form files from the command line. From the command line, it compiles .per files to .pic files, .per files to .4fm files, and .4fm files to .pic files.

The v flag instructs the form compiler to display the version number and then exit, without compiling anything.

The e option followed by "utf8" value will allow you to compile the form in the UTF-8 encoding.

qlink

Syntax: qlink [options] file...

Where 'options' is one or more of:

-? [--usage]	Display this usage information
-V [--version]	Show version



```
-o [ --output ] arg           Output file
-d [ --database-driver ] arg Database driver
-v [ --verbose ]              Verbose output
--verify                      Verify output
--errors arg (=default)      Verify function resolution
-m [ --merge ]                Merge
-l [ --lib ] arg              Dynamically Link library
```

The qlink command is used to link files together. To link 4gl files together, the following command would be called:

```
qlink -o x.4a a.4o b.4o c.4o
```

where a.4o, b.4o and c.4o are statically linked together to form the p-code module x.4a

```
qlink -o x.4a a.4o b.4o -lib c.4o
```

would mean that a.4o and b.4o are statically linked into x.4a, whereas c.4o is dynamically linked to x.4a. x.4a contains the two p-code modules and the runner code allowing it to be executed directly from the operating system.

The --verify option checks for duplicates of function declarations and for function calls for which the function was not defined. If it is omitted, the program will compile normally even if there are duplicated functions or calls for missing functions. If this option is included, the warning about the duplicated functions names and locations will be given, but the program will still compile. In both cases you will get a runtime error, if during the program execution a duplicated or a missing function is called.

qmsg

```
Syntax: qmsg [options] input-file output-file
```

Where 'options' refers to one or more of:

```
-? [ --usage ]                  Display usage information
-v [ --version ]                Show version
-v [ --verbose ]                Verbose output
-e [ --encoding ]               Input-file encoding
-d [ --define ]                 Define symbols
-i                           Path to the connected files esql
```

qmsg is the message compiler. It generates a compiled message file of the name specified by `output-file` from an ASCII source file of help messages specified by `input-file`. Since there is no convention on message file names, Lycia does not require any specific file extension format.



qxcompat

The qxcompat has been specially created for the compatibility reasons. It can convert the whole project at a time or separate files from a specific project to the corresponding format, so that it can be successfully compiled and executed.

Syntax:

```
qxcompat --input-file <file_name> --output <output>  
or  
qxcompat <file_name>
```

where:

--input-file <file_name>	a file with one of the following extensions: .4ad, .4fd, .4fm, .4pw, .4sm, .4st, .4tb, .4tm
--output <output>	a directory (for projects), a form file, etc.

Here is the list of the correspondence of a source file format to a converted one:

Input file extension	Output file extension
Project	
.4pw	is included into .fglproject
Action Defaults	
.4ad	.ad2
Graphical form	
.4fd	.fm2
.4fm	
Start menu	
.4sm	.sm2
Styles	
.4st	.qxtheme
Toolbar	
.4tb	.tb2
Top menu	
.4tm	.tm2

To illustrate the qxcompat command line tool operational principles, let us consider some usage examples.



Example 1.

```
qxcompat --input-file project_name.4pw --output output_folder_name
```

where:

- `project_name.4pw`

a project file with the following content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Workspace defaultApplication="000000001"
version="1.0">
    <Group label="Source">
        <Application id="000000001"
label="test_conversion">
            <File filePath="input_toolbar.4tb"/>
            <File filePath="input_topmenu.4tm"/>
            <File filePath="input_sourcefile.4gl"/>
            <File filePath="input_formfile.per"/>
        </Application>
    </Group>
</Workspace>
```

- `output_folder_name`

a folder where output should be located

As a result of the `project_name.4pw` project file conversion, the new Lycia 2 project will be created in the directory specified as `output`. Apart from the default Lycia project content, the project folder `output_folder_name` will contain all the files listed in `project_name.4pw`, converted where a file format requires, as they are listed below:

<code>input_toolbar.4tb</code>	-->	Output:
<code>input_topmenu.4tm</code>	-->	<code>input_toolbar.tb2</code>
		<code>input_topmenu.tm2</code>
		<code>input_formfile.per</code>
		<code>input_sourcefile.4gl</code>

Example 2.

```
qxcompat file_name.4tm
```

If the only argument follows `qxcompat`, it is regarded as input-file. The following conversion will take place (the output file will be saved in the same direction):

Input:		Output:
<code>file_name.4tm</code>	-->	<code>file_name.tm2</code>



4make

Syntax: 4make [options] target_name

4make is used to compile a 4gl program on Unix only. 4make can be used to compile a single 4GL file to a P-Code executable in the form

```
4make myfile.4gl
```

4make can also be used to compile a program comprising multiple sources, based on a definition within a .def file.

```
4make program_name
```

where 'program_name' refers to a TARGET declaration in a .def file. For example:

```
TARGET=myprogram
SOURCES=file1.4gl
    file2.4gl

FORMS=form1.per
    form2.per
```



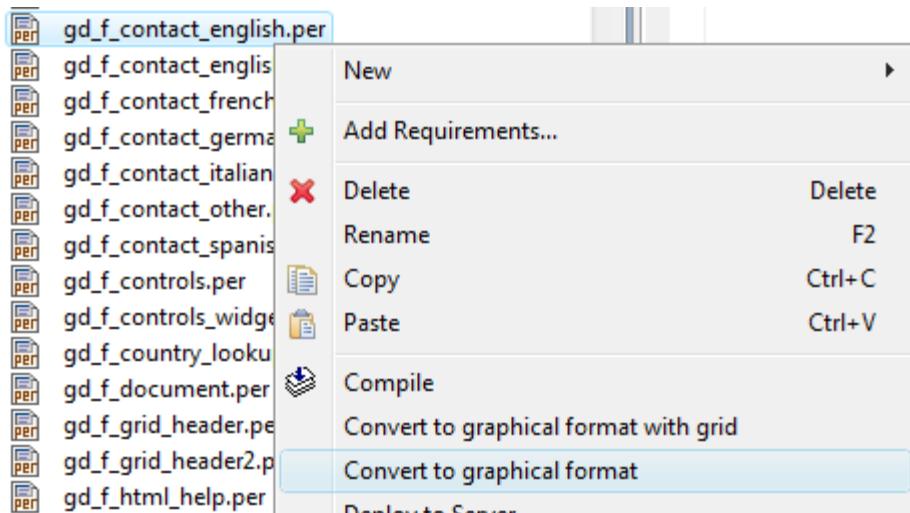
Using the form editor

Lycia incorporates a new graphical form editor to allow developers to design or enhance their form files into a project with ease. There is still the option to design form files in a classic 4GL text format (.per files) as in previous revisions. To edit files graphically (.fm2 files) they will need to be either created originally in this format, or converted using Lycia's conversion tool, which will preserve all conditional data and form details and keep the form looking just as before.

Converting form files

You can convert .per and .4fm form files into .fm2 format. Select the form file that requires converting and right-click on it and select **Convert to graphical format** or **Convert to graphical format with grid** from the menu. The difference between these two options is in the root container that the converted form will get. The first option will create an .fm2 form with CoordPanel as a root container. The form with grid has a GridPanel as a root container. The GridPanel improves the behaviour of the form and form objects during the resizing.

The Console Panel will update and inform that the conversion is complete and the file in the 4GL Project View panel will change from the previous extension to the new one.

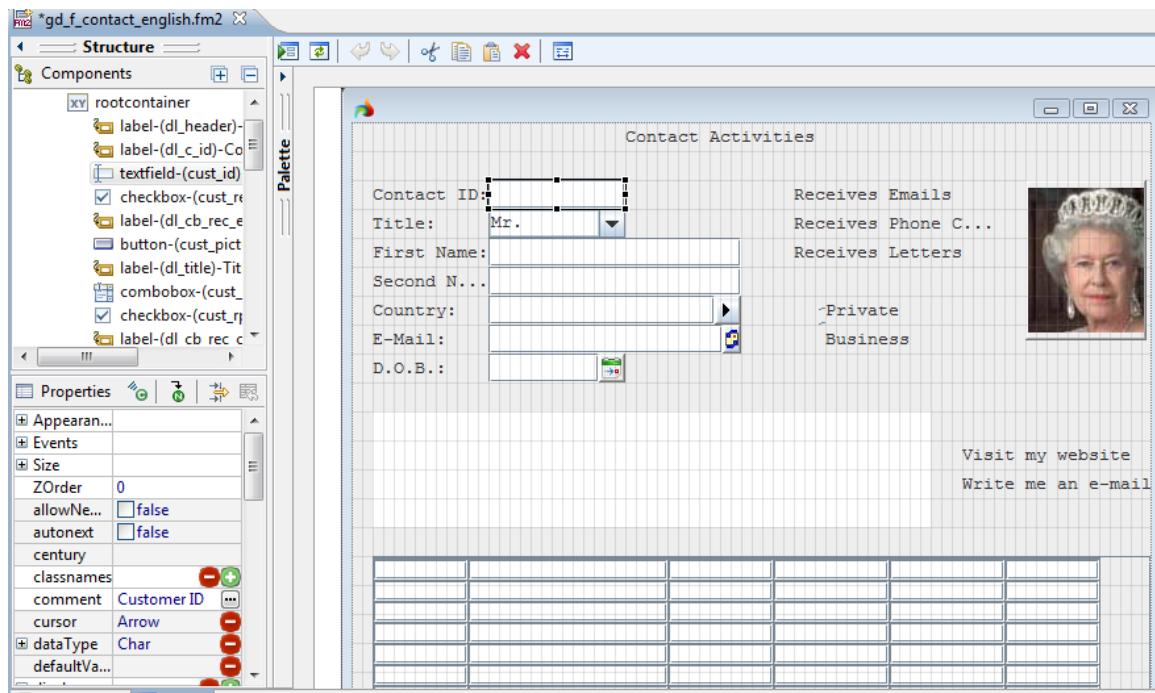


You can convert form files only in one direction: from text and older .4fm graphical format to the new .fm2 format. You cannot convert new graphical form files back to older graphical and text format.

Although it is possible to create and manipulate both .per and .4fm form files, they still will be automatically converted to .fm2 during the compilation.



Above shows how the colours form file from the colours demo project would look once converted to a graphical format. As you can see, by default a grid is overlaid onto the form so positioning of fields is quick and simple, and any colour attributes applied to a field can be seen instantly in the Properties tabs below. Below is a comparison to the classic 4GL editor, so you can see the difference.



Above shows how the contacts form file from the GUI demo project would look once converted to a graphical format. As you can see, by default a grid is overlaid onto the form so positioning of fields is quick and simple, and any colour attributes applied to a field can be seen instantly in the Properties tabs at the left part of the editor. Below is a comparison to the classic 4GL editor, so you can see the difference.



The screenshot shows a software interface titled "gd_f_contact_english.per". The window is divided into several sections:

- HEADER:** Shows the file name "gd_f_contact_english.per" and the extension ".per".
- DATABASE:** Set to "formonly".
- SCREEN:** A code block containing 4GL script. It defines a screen with a header section and a body section. The body section contains various fields and their properties, such as "f_cust_id", "f_cust_title", "f_cust_fname", "f_cust_lname", "ff_cust_country", "ff_cust_email", "ca_cust_dob", "f_memo", "hl_url", "hl_cust_email", and "fb_input", "fb_error", "fb_message", "fb_cancel", and "fb_help".
- ATTRIBUTES:** A section at the bottom of the code editor.

Forms

A form file is a file describing an object used for input and interaction with the 4GL program. A form file has a number of properties which can be adjusted.

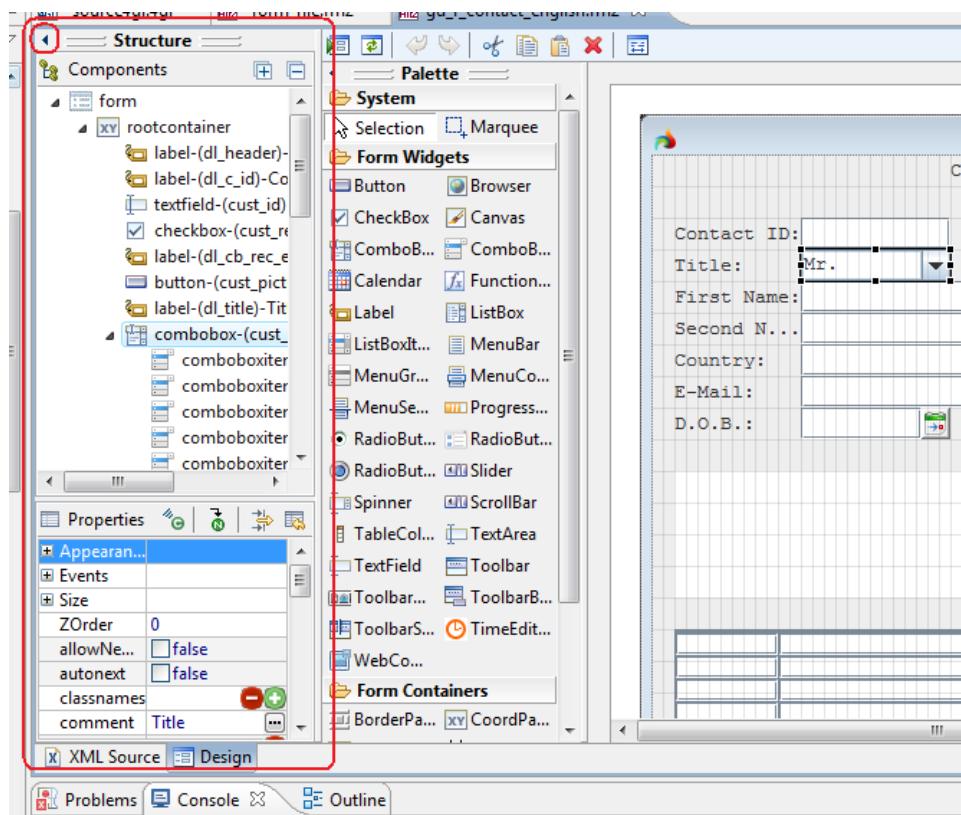
The right click on the form calls a menu that has the following useful options:

- Layout – allows to change the root panel keeping to the existing widgets layout as much, as it is possible
- Delete tags – allows to totally delete some properties from the form. E.g., if the “items” tag is deleted, all the “Items” properties in the form will be cleared, including the Root panel “Items”, so that the form will become empty. It is also possible to delete such parameters as minsize, location, displaymodes etc.

General Form Properties

New Lycia forms have hierachic multilevel parent-child structure. The Main parent object of is the form itself. The form direct children are tool- and menu bars, records and a root panel. The root panel is the object that contains all the other form widgets, and some of these widgets may also have child elements.

The Design view includes a Structure panel, which displays two elements – the form Components and the form Properties views. The Structure panel is placed at the right part of the Design view and can be hidden by pressing the button at the top left corner of the Structure panel:



As it can be seen in the screenshot above, the Components view displays the form components structure in a tree, so that it is easy to keep the trace of the parent-child relations among the form components. It is possible to select form objects not only from the Design area, but also by clicking on them in the Components view.

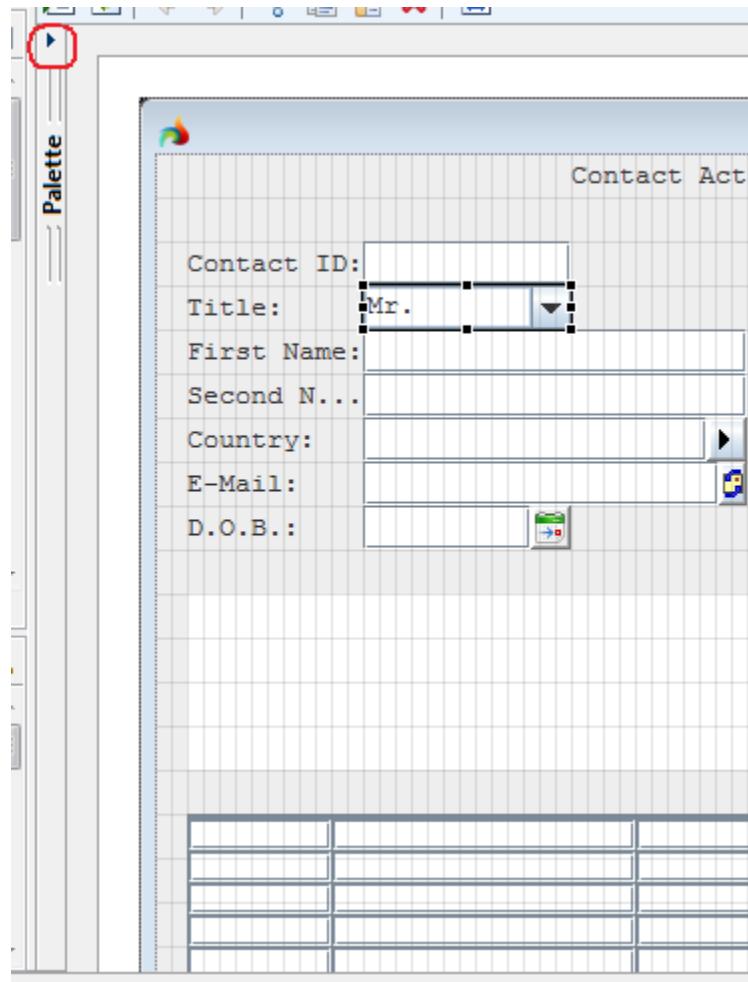
The Properties view is placed under the Components view and allows to see and to change the properties of the currently selected object. The view is empty if no object is currently selected. If several objects are selected, only their common properties will be available.

The set of the properties in these groups depend on the currently selected object. Some of the properties are available for all the available form objects or for all members of a corresponding object group (containers or widgets). The other properties are object-specific. The properties in the Properties view are organized into groups according to their purpose. The properties allow to set up the objects appearance, behaviour, interaction with the user and application at runtime, etc.



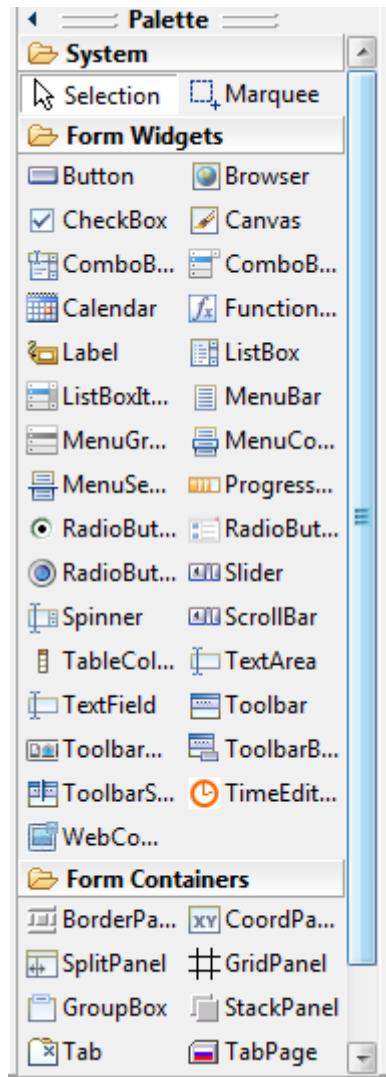
Palette

To add a new widget to a form or to use the selection tools, the widget palette needs to be used. The palette contains all available form containers and widgets. Usually it is opened automatically together with a graphical form file. However, if it has not been opened automatically, it can be found on the left-hand side of the form editor and unfolded as indicated below by the red circle:





Once active, the palette looks as follows:



To use the palette, the developer simply clicks on the needed widget once, then clicks where they would like the widget positioned upon the form. They are then free to customise the widget as required: change the size of a widget, its position etc. Each of these widgets will be explained in detail in the sections below.

To create more than one widget of the same type select the widget and hold the **Shift** key. In such case each time you click within the form editor area, a new widget of the selected type is created. To be able to change the properties of a widget, you need to switch to the Select tool. If the widget button is not stuck, the cursor will automatically change to the Select tool after a single widget of the selected type is placed onto the form.



Widgets on the palette make up the majority of the contents of a form file, and allow user interaction with data using a variety of layouts and looks when text input is not appropriate. Whilst having their own properties, they also have common properties, which will be listed afterwards.

The Form has two main types of objects - containers and widgets. A container is an area that nests the form widgets. Different containers have different widgets layout and manipulation behaviour.

Containers

When the form is created, you need to select the root container it will be using. All the other containers and widgets can only be placed inside the root container. The choice of the root container influences the resizing and layout possibilities for the form at runtime.

Border Panel

The objects placed to the Border Panel, are automatically located along the panel borders. When you select a place where a widget is to be placed, the border areas are highlighted. When the widget is dropped, it takes all the available space along the selected border and gets its default height or width:

Therefore, if there is a widget already placed at one border, the edge of the widget placed to the neighbour border is defined by the edge of the previously placed one:

When an object is placed to the central part of the border panel, it automatically fills all the rest of the container:

Container specific properties: padding

Coord Panel

The location of the objects within the Coord Panel is determined by x and y coordinates, which specify the location of the object top left corner on the horizontal and vertical axis, respectively.

The object with coordinates XCoord = 0, YCoord = 0 will be placed at the top left corner of the parent Coord Panel container.

Container specific properties: none



Grid Panel

In Grid Panel Container, the objects are placed within a grid. One object can take several cells, and the grid structure allows to adjust the elements sizes and location easily. The grid lines are abstract, i.e., they are visible during the panel manipulations and not visible when the grid panel objects is inactive and in runtime.

When the first object is added to the grid panel, a conditional grid appears. When the object is dropped, the grid size and lining is adjusted so that it is placed to the bottom right corner.

After that, it is possible to change the number of grid rows and columns.

Container specific properties: *GridItemLocation (for child elements), gridLength*

GroupBox

The GroupBox container is used to create a titled frame around a form widget. The Group Box can contain only one object, which can be a widget or any other panel with its own children.

Container specific properties: *Font, ForeColor, Padding, title, Title Justification*

ScrollView

The main feature of the ScrollViewer is that in can include more actual space than it is limited by the container edges. The container can have only one child, but this child may be represented by another container with its own set of fields..

If the ScrollViewer content is wider than the container itself, the corresponding scroll bar automatically appear so that the user can scroll the content to see the necessary part.

Container specific properties: *Padding, HorizontalScrollbarVisibility, VerticalScrollbarVisibility*

Stack Panel

The Stack panel is used to create ordered stacks of objects. They are automatically aligned and placed under each other or next to each other, depending on the panel orientation.

The panel can nest both containers and widgets.

Container specific properties: *Orientation, Padding*



Tab Panel

The tab Panel creates a space in the form where the several tabs can be placed and later be easily switched. It does not create the tabs themselves, but only allocates space for them. The tabs are created with the Tab Page containers that should be added as tab Panel children.

Container specific properties: *Enable Multiline, Padding, TabPagePlacement, OnSelectedTabPageChanged event*

Tab Page

The Tab Pages are the containers that include other objects and can be switched. The Tab Pages should be placed within the Tab Panel container.

The tab page body can include one child, which is typically represented by another container. The content of a tab page does not depend on the content of other tab pages

Container specific properties: *Font, ForeColor, Image, OnSelectTabPage Event*

Table

The Table allows to create tables for data input and display.

A form table has a hierarchical structure, in which the Table container is the main Parent object.

The Table container is filled with table columns, each of them having a form widget inside.

Table identifier is linked to the identifier of the screen record nesting the table column fields. Therefore, editing one of these IDs automatically changes the other.

Container specific properties: *Cell Padding, GridColor, HorizontalScrollBarVisibility, Identifier, IsMultiselect, IsVirtual, Margin, Padding, RowCount, RowHeight, RowResizable, TextAlignement, VerticalScrollBarVisibility, Visible, VisibleToolbarGroups.*

Table Column

The table Column is a child object of a table container and is used to nest a widget that becomes an element of a screen array.

Each column can include only one form widget. All the fields added to the table columns are automatically added to an automatic screen record linked to the table. The changes in the columns order and number are reflected in this screen record immediately.



Container specific properties: *Font, ForeColor, ColumnLength, Grid Length Type and Value, Text, Aggregate properties, Total Aggregate properties,*

Tree Table

The Tree Table container is a container that allows to create tree tables.

As well as a standard table, the Tree table consists of columns, each having a field in it. The Tree table includes the columns that are displayed, the columns that regulate the parent-child relations between the nodes, and the columns that specify additional settings - e.g., identify, whether the parent node is expanded or collapsed by default. The purpose of the columns is specified by means of special Tree Table properties. The values of these properties should be the names of the table columns, in which the corresponding node info is stored:

- ColumnId - sets the specific ID of the node
- ColumnIsNode - specifies a name for a column, listing the Boolean values indicating whether the node can have children(TRUE) or not(FALSE), irrespectively of whether the parent-to-be element currently has children.
- ColumnParentId - sets the ID of the node parent, making the node itself a child of the node with the given ID
- Column Edit - specifies the column to which the parent node values are displayed. When the property is empty, the first column is treated as the Edit one.
- Column Expanded - sets a column in which contains Boolean values that indicate whether the corresponding node is expanded (TRUE) or collapsed (FALSE)
- ColumnImage - specifies a name for a column listing the paths to the pictures to be displayed near the node names

Container specific properties: *ColumnEdit, ColumnID, ColumnIsNode ,ColumnParentId, ColumnExpanded, ColumnImage*



Form Widgets

Dynamic Form elements are widgets on a form which can be changed during program execution.

Button

A button widget can be placed on a form to record user input in the form of clicking on a button. It can contain text or it can be a clickable image. The text and the image are changeable. The button click can be mapped to a key event or an action event, defined on the form.

Widget specific properties: *IsToggleButton, Text, AllowNewLines, Image, OnInvoke event*

Browser

The file browser widget embeds a fully functional browser into the form. The file browser will attempt to render the source specified in its field value. You can use the browser to view web pages, files, images and folders on your computer. The file browser is also useful to merge 4GL applications with Web-based applications

Widget specific properties: *Text(the address for the browser to load), Events*

Canvas

The Canvas widget is used to display interactive SVG images.

Widget specific properties: *Image, Events*

Check Box

The Check Box widget is used to place check boxes on a form. They are used to select one or more values from a range of options. They can also be used for True/False or Yes/No values, by checking the box only for a positive response to the static text statement alongside it.

Widget specific properties: *CheckedValue, UncheckeckedValue, OnCheck and OnUncheck events, Title Image*

Combo Box

A combo box will usually display a drop-down-list with a button on the right-hand side to make the list appear. The list can be populated statically within the form definition using the



ComboBoxItem widget. It is also possible to manipulate combobox values at runtime by using the fgl_list_xxx() functions.

Widget specific properties: *Editable (allows text input into the combo box possible),
Initializer, MaxDropdownHeight, MaxDropdownWidth*

Calendar

The Calendar widget can be used to place a calendar function within a form field that needs a date to be entered. The date can be entered into the text field after selecting it from the popup window, else a date can be manually added. The format for the date is taken from the environment variable DBDATE, unless otherwise specified.

Widget specific properties: *Century, Format*

Function Field

The function field widget allows the developer to introduce a field where the user can click on the button when the field is in an INPUT state. This button can have an icon associated with it. The button should be used to trigger a key or action event, which is defined in the Properties section.

Widget specific properties: *Image, OnInvoke event*

Label

The Label widget is a form widget typically used to display data that cannot be manually changed by the user on runtime. This can be some text or an image..

Widget specific properties: *isDynamic, Image, Text, TextAlignment, AllowNewLines,
Forecolor, OnMouseClick event*

Listbox

The Listbox widget is used to display structured list of values the user can choose during the input. The whole list is displayed to the form and cannot be changed at runtime..

It is possible to select several items at once. The Listbox widget is filled with Listbox items that can also be taken from the Palette

Widget specific properties: *EnableMultiSelection, OnSelectedItemChange event, Text
Alignment*



Radio Button

Radio buttons allow you to create a group of items from which only one option can be chosen. You can have one single or several radio button items in one radio button group but all items in the group will write to the same variable.

Radio Button widget allows to add radio buttons to the custom location on the form, unlike the RadioButtonList widget, where all the items are grouped in one place and definite order.

Widget specific properties: Value, GroupIdentifier, OnCheck/OnUncheck events, Title, Image, TextAlignement

Radio Button List

RadioButtonList widget is a container that allows you to create a set of radio buttons grouped automatically under one identifier. The widget is filled with radioButtonListItem objects that create the set of the available values

Widget specific properties: Orientation, OnSelectedItemChange event, Text alignment

Progress Bar

The Progress Bar widget is used to display a numeric value within the specified range. It is typically used to display graphically the level of progression of some process or some level of fulfilment.

Widget specific properties: Orientation, Maxvalue, Minvalue, Step

Slider

The Slider Bar widget is an graphical widget that allows to input or display a numeric value by changing the position of a slider on a scale.

Widget specific properties: MajorTick, MinorTick, Maxvalue, MinValue, orientation, ShowTicks, OnValueChanged event

ScrollBar

The Scrollbar widget is used to manipulate numeric values within the specified range. It can be used to input values, but is more likely to make the application perform some actions when the user moves the scrollbar slider (OnScroll event).



Widget specific properties: Maxvalue, MinValue, LargeStep, SmallStep, Orientation, OnScroll event

Spinner

The Spinner widget is a graphical widget that allows entering numeric values both manually or using the spinner arrow buttons..

Widget specific properties: Step, maxValue, minValue

Text Field

A text field is used for user input. It can be mapped to represent a field in a database, can be set to be a no-entry field, the type for the field and many attributes such as the field colour, font and can be set, amongst many other features and settings.

Widget specific properties: none

Text Area

As with the Text Field, Text Area is used to input and display text data of different formats. The value inputted to the text field can include any printable characters. The only feature that distinguishes Text Area from Text Field is that Text Area allows the data string to be split into several lines. This option is activated by setting the AllowNewLines property value to True.

Widget specific properties: AllowNewLines, UseTabs, Horizontal/VerticalScrollBarVisibility, TextAlignment

Time Edit Field

The TimeEditField widget is displayed like a spinner field with three numeric buttons separated with a colon symbol.

The spinner is used to change the hours, minutes and seconds values, depending on the part of the time value in which the cursor is currently set.

Widget specific properties: none



Web Component

A WebComponent widget is a form widget that can serve as a seat for external objects, or front-end plug-in mechanisms.

The WebComponent properties are used to specify the object to be used and to set it up. The settings allow to establish communication between the application and the web-component..

Widget specific properties: *ComponentType, ComponentProperties*

The graphical form editor usage

To get acquainted with the graphical form editor let us create a simple 4GL project containing a graphical form file. The application will look like a librarian card. The form placed to it will consist of two tabs. One of them will be designed to get and save reader's personal data, the other one will be a kind of a reader's ticket with a table to make records concerning the reader's book taken history there.

In order to create the application stay focused upon the following steps execution:

1. Create a 4GL project called "graphical_form_demo".
2. Create a 4GL program within this project called "form4gl".
3. Create a 4GL source file called "source4gl".
4. Copy the following source code and insert it into the "source4gl" file:

```
MAIN
DEFINE
    my_form WINDOW,
    my_record RECORD
        fname CHAR(15),
        lname CHAR(15),
        age INT,
        location CHAR(30),
        reg_date DATE
    END RECORD,
    books_list DYNAMIC ARRAY OF RECORD
        f001 CHAR (10),
        f002 CHAR (10),
        f003 CHAR (10)
    END RECORD

OPEN WINDOW my_form AT 1,1 WITH FORM "form_file"
ATTRIBUTE (BORDER)
```



```
DIALOG
--Inputs first name, last name and age to the text fields,
location to the RadioButtonList and date to the Calendar located
in the General tab

INPUT my_record.* FROM app_info_rec.*
END INPUT

--Inputs data to the table located in the Records tab

INPUT ARRAY books_list FROM tab_rec.*
END INPUT

--Make the program to save the updated information when the user
presses the button "Save"

ON ACTION(save)

--Opens a dialog box when the user presses the button "Save"
CALL fgl_winmessage("Save","The information is saved","info")

--Make the program to quit when the user presses the button "OK"

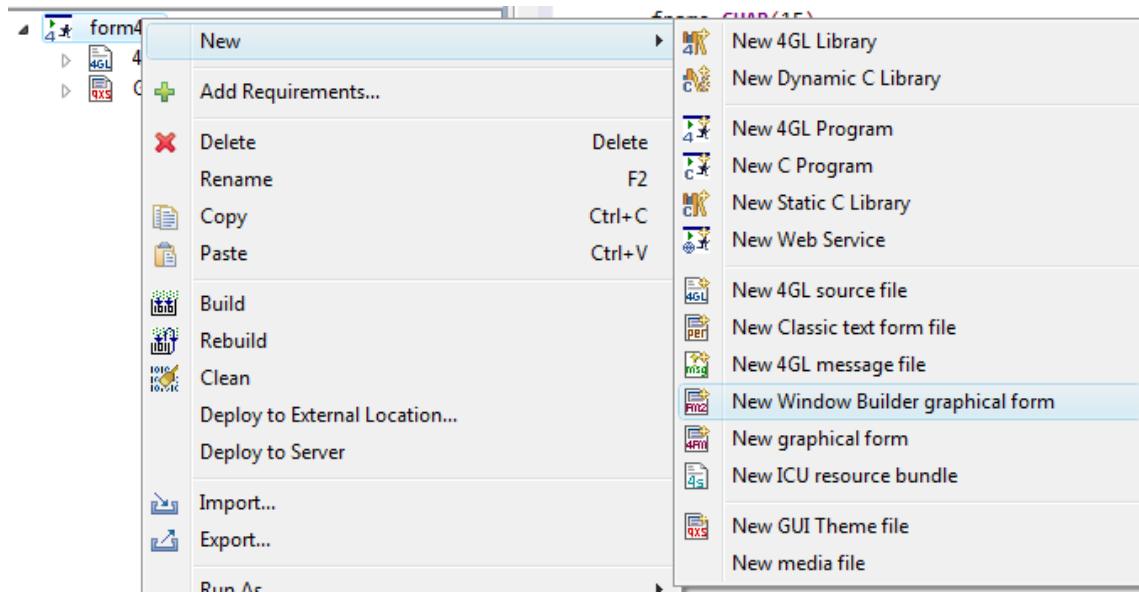
ON KEY(esc)

--Opens a dialog box when the user presses the button "OK"
CALL fgl_winmessage("OK","The program will be closed in a
second","info")

EXIT DIALOG
END DIALOG
END MAIN
```

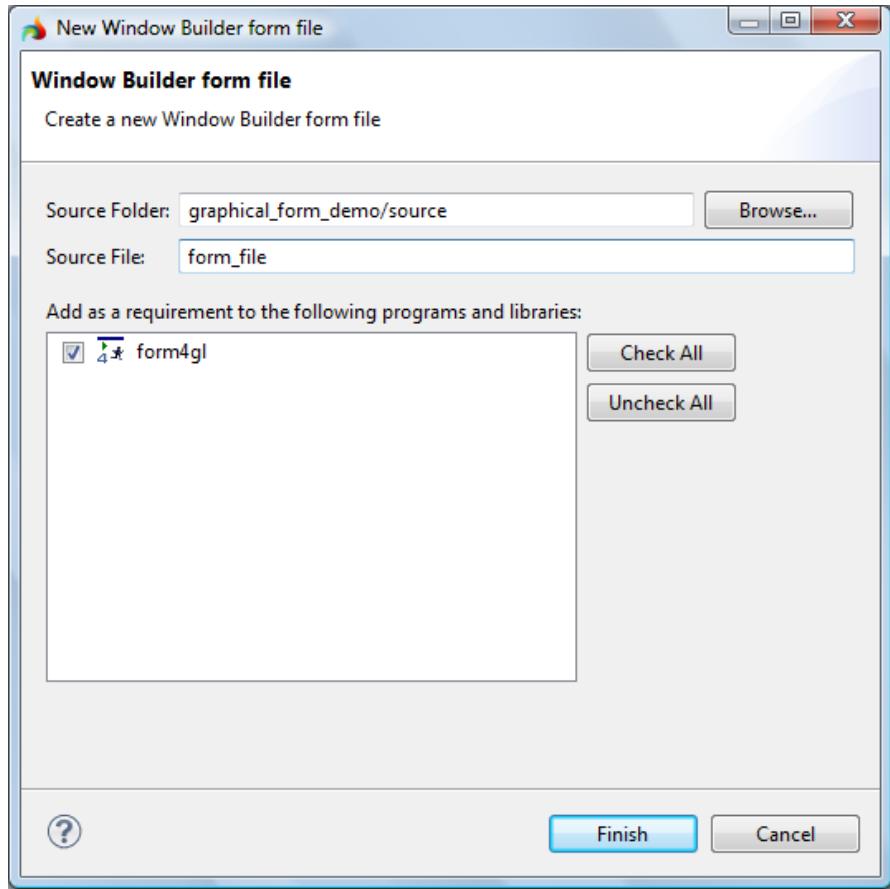


5. Right click the "form4gl" program in the 4GL Project View.
6. Select the "New -> New Window Builder Graphical Form" option from the context menu:

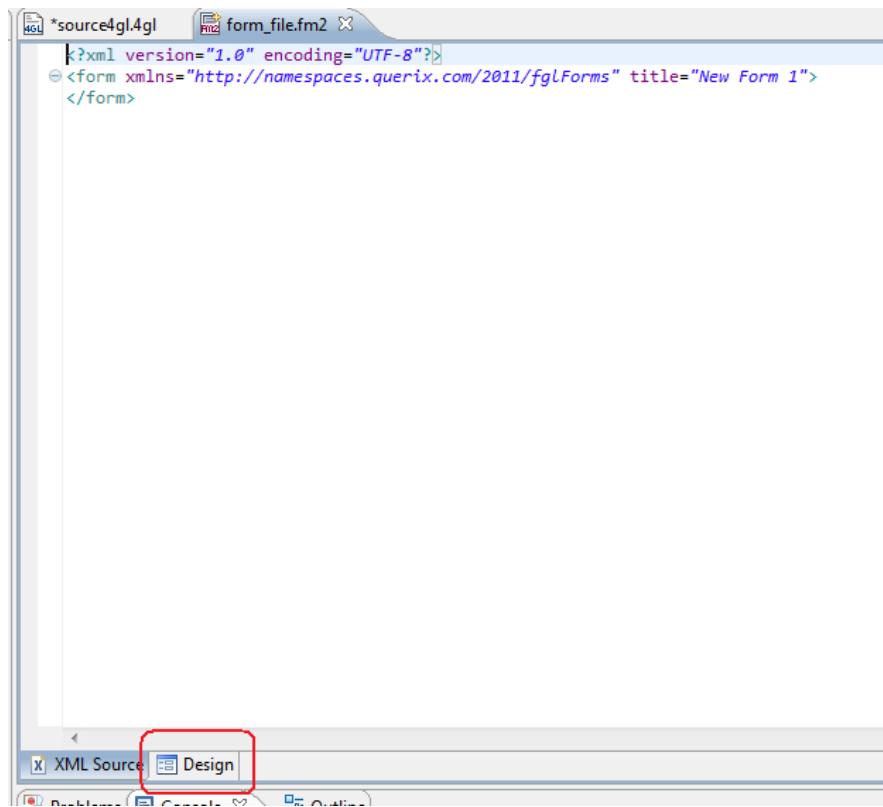




7. Create a new form file called "form_file", click **Finish**:



The newly created form file will be opened in the XML-source editor where you can see the basic form template. In the process of the form creation the form source code changes can be traced there. Click the **Design** tab at the bottom part of the editor to switch to the graphical designer view, where the form content can be manipulated graphically.



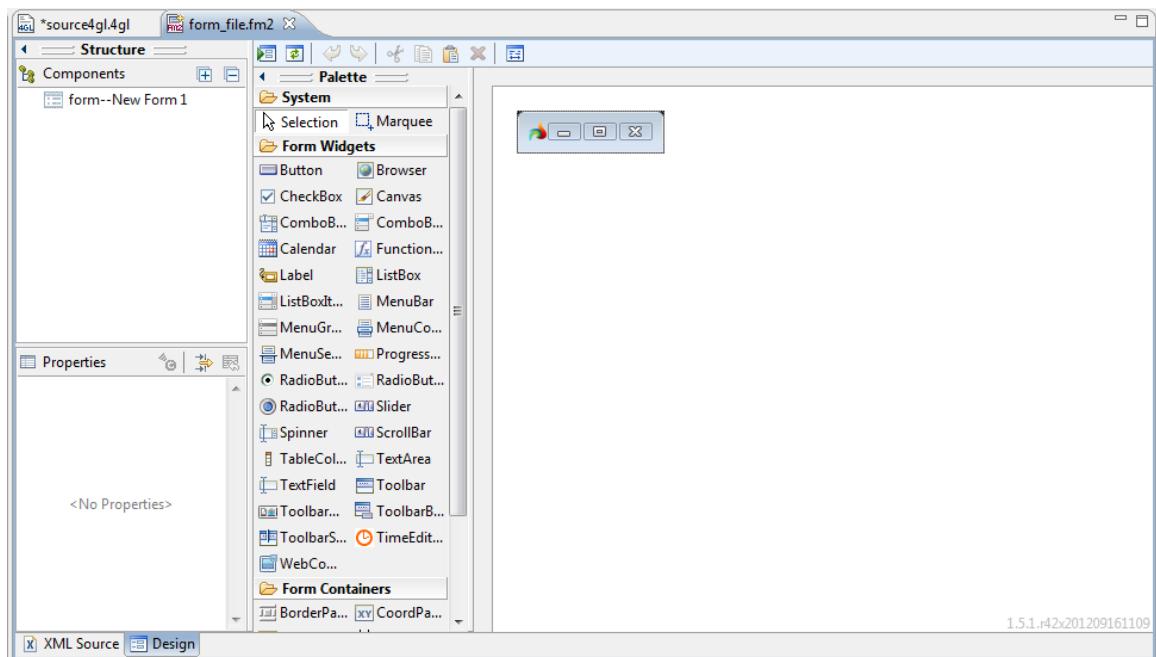
A screenshot of the Lycia Form Editor interface. The window title is "form_file.fm2". The XML code in the editor pane is:

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://namespaces.querix.com/2011/fglForms" title="New Form 1">
</form>
```

The bottom navigation bar shows tabs for "XML Source" and "Design". The "Design" tab is highlighted with a red rectangle.

It is not recommended to manipulate the form by changing its source code manually. This can result in errors, for example, in screen records naming and content as changing field identifiers in XML does not lead to automatic changes in the field references. There are may also be other issues that are difficult to trace.

The initial layout of the **Design** view is given in the screenshot below:



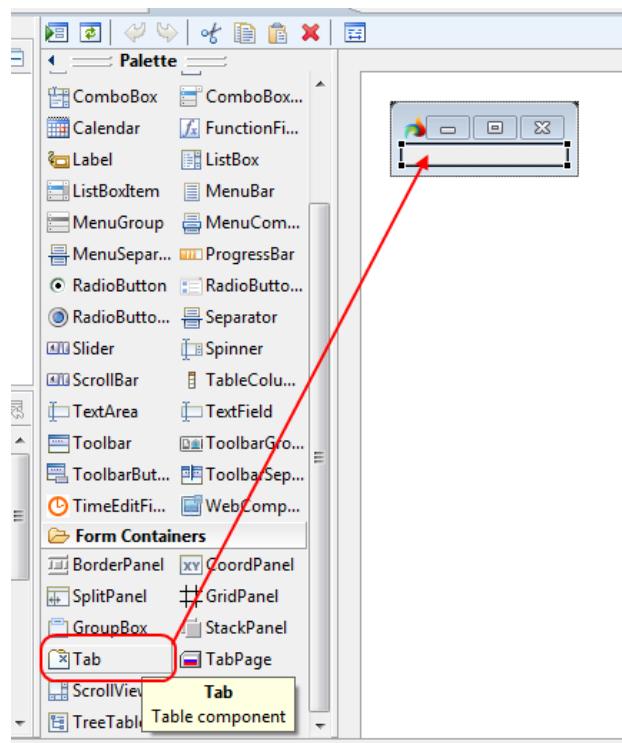
Creating a Form

To get acquainted with the widgets available and their specific properties, follow the instructions given in this section to populate the "form_file" of the "form4gl" program.

To set a non-default form title, select the form in the **Components** view and change the value of the **Title** property.

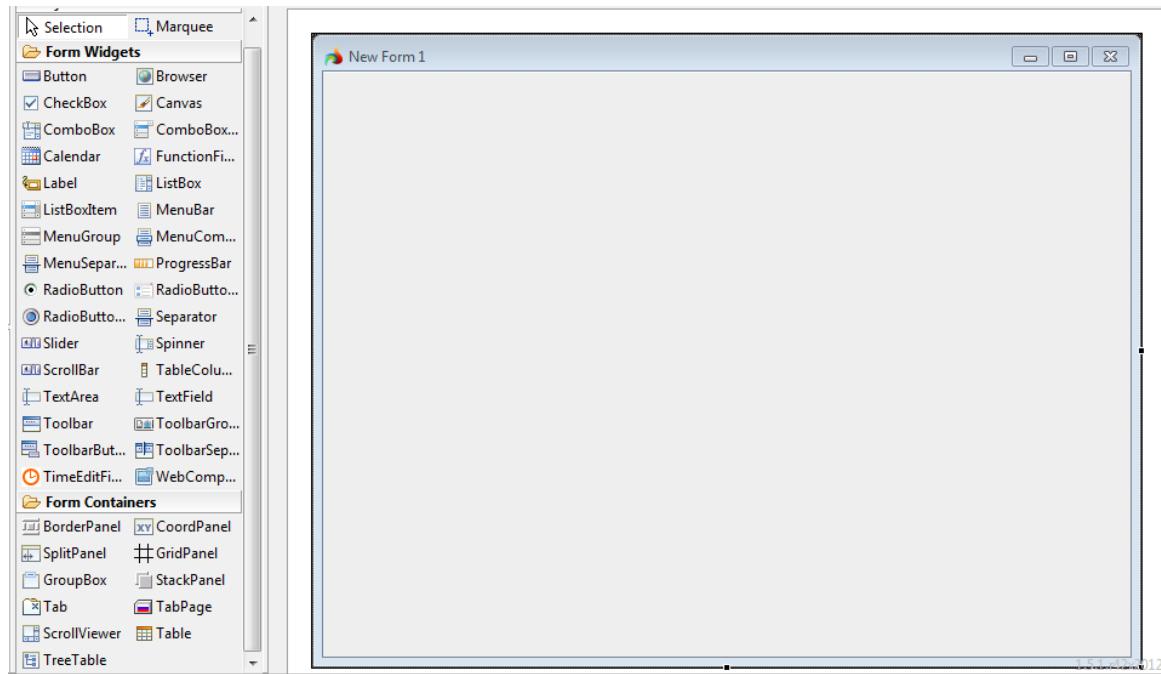
Selecting the Root Container

Each form should include a root container which will become a seat for all the other form elements. As it was mentioned earlier our form will have two tabs, therefore, the **Tab** container should be added to it as the root one:



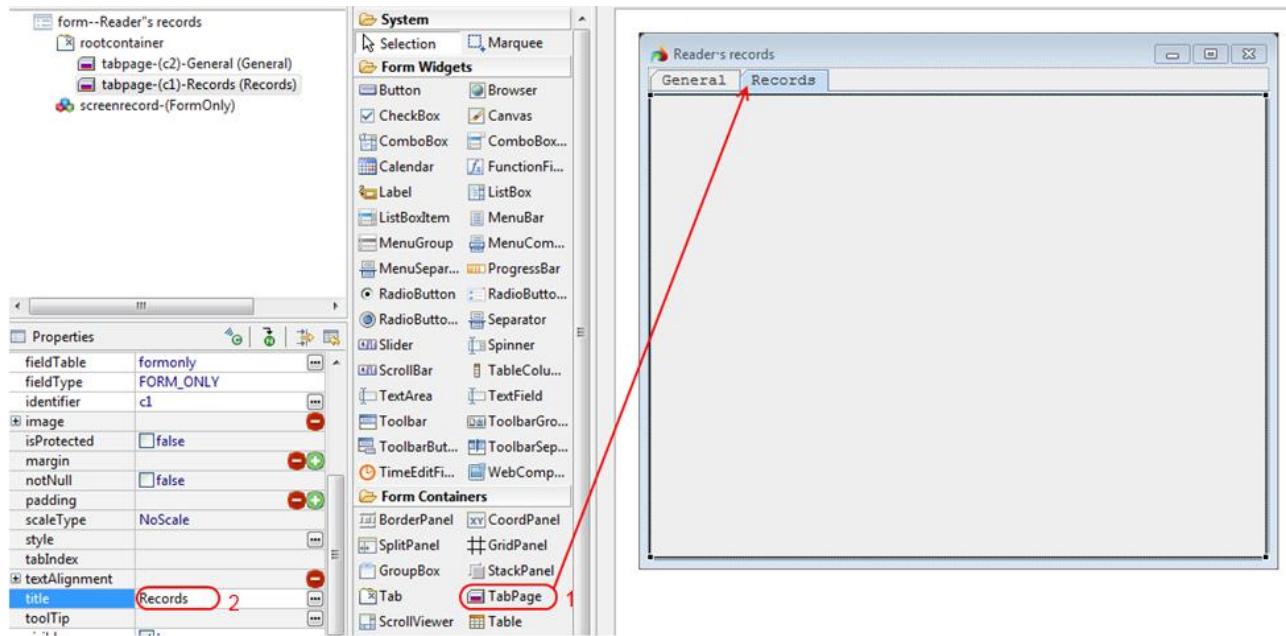


After the root container has been added, assign the desired height and width to the form by dragging its edges:





The tab container should nest tabs, in which the form widgets will be placed. Add two tab pages to the tab container (1). To change the tab header, type in a new title to the **Title** property: let it be *General* for the first tab and *Records* for the second one (2):



The **Tab** container should include another container to fit the necessary elements. In our example, we add a **Grid** container to the *General* tab and a **Stack** panel to the *Records* tab. The instructions of their adding are given further in this chapter.



Adding a Grid Container and a Button to the General Tab

1. Select the General tab in the **Structure** view form tree:

The screenshot shows the Lycia Form Editor interface. On the left is the **Structure** view, which displays a tree of components under "form--New Form 1": "rootcontainer", "tabpage-(c2)-General (General)" (highlighted with a red box), "tabpage-(c1)-Records (Records)", and "screenrecord-(FormOnly)". In the center is the **Properties** panel, showing properties for the selected tabpage component, such as "image", "isProtected" (set to false), "margin", "notNull" (set to false), and "padding". On the right is the **Palette** view, which is divided into two main sections: **Form Widgets** and **Form Containers**. The **Form Widgets** section lists various UI elements like Button, CheckBox, ComboBox, etc. The **Form Containers** section lists BorderPanel, SplitPanel, and GridPanel.

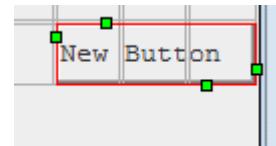
2. Select the Grid Panel container in the **Palette** view, **Form Containers** folder and add it to the tab.
3. Add a button to the bottom left corner of the tab from the **Form Widgets** folder:

The screenshot shows the General tab of the form. A **GridPanel** component has been added to the tab's content area. A **Button** component from the **Form Widgets** palette has been placed at the bottom-left corner of the grid. A red box highlights the **Button** in the palette, and a red arrow points from the palette to the button on the tab.

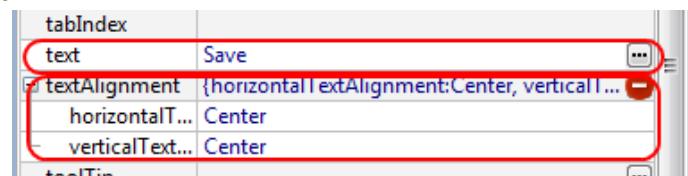
The position of the first element added to the grid container defines the container bottom left corner. You can later manually add or delete rows and columns if needed.



4. The new element (the button in our case) takes only one cell of the grid on default. Expand it by dragging its edge so that it takes three cells in a row:



5. To change the button label use the **Text** property field in the **Properties** section. Double Click on the button can also be used for this purpose.
6. Set the button **horizontal** and **vertical text alignment** properties to *Center* to line up the label position:



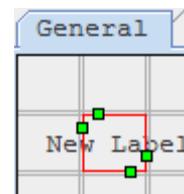
7. Go to the **Events** properties group and set up the **OnInvoke** event handler: select **ActionEventHandler** and specify the **ActionName** as *Save*. This will make the application to save the data entered by the user to the **General** tab fields at runtime:



Adding Labels and Text Fields

Now, when the container is ready and set up, we can add some fields to the form. We add labels to display text and fields of different types to input or display data.

1. To add a label, select the **Label** widget in the **Palette** view and click on the target cell of the grid container:



Drag the right edge of the label to the right so that the label takes three cells in a row. Double click the label to change its text:

A screenshot of the Lycia form editor interface. It shows a grid-based layout with a single label component labeled "First name:". The label has a red border and is positioned in the top-left corner of the grid.

You can also change the label identifier in the properties view, to make the further reference easier:

A screenshot of the Lycia properties view. The left pane shows a tree structure with nodes like "tabpage-(c2)-General (General)", "content-3", and "label-(l_fname)-First name:". The right pane shows various properties for the selected label. The "identifier" field is highlighted with a red box and contains the value "l_fname". Other visible properties include "fieldTable" (set to "formonly"), "fieldType" (set to "FORM_ONLY"), and "gridItemLocation" (set to "gridHeight:1, gridWidth:...").

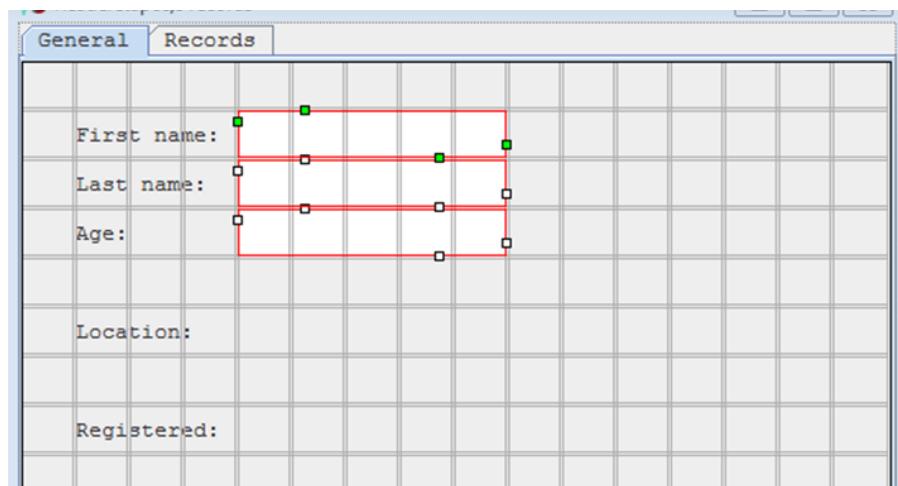
2. Add the other labels to the form:

A screenshot of the Lycia form editor interface showing a grid with several labels added: "First name:", "Last name:", "Age:", "Location:", and "Registered:". The "Registered:" label is currently selected, indicated by a red border around its grid cell.

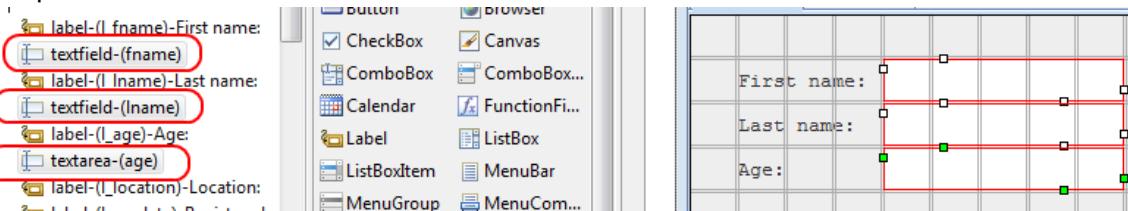


If needed, you can change the labels text alignment by setting the **TextAlignment** properties. You can manipulate several labels (or other objects of the same or different type) simultaneously and set up the same values for their common properties. To do it, select several objects (by clicking on them with the **Shift** or **Ctrl** button being pressed or with the help of the **Marquee** tool:)

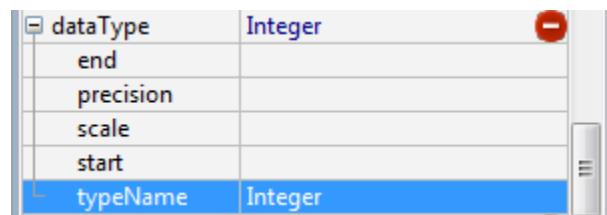
3. Select the **Text Field** widget in the **Palette** and add it to the necessary place in the form. Adjust the field size the same way you did it for the labels and the button:



4. Set up the fields **identifiers**:



5. The Age field may accept only numeric values. Therefore, it is necessary to set up the field data type. Select the field needed, unfold the **Data Type** property and specify the **Type Name**:



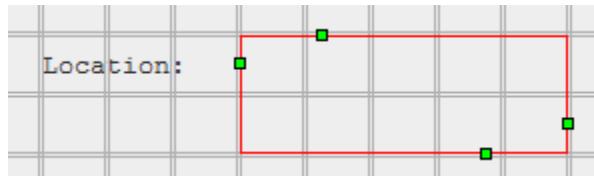
If you need to specify a field of such data types as DECIMAL or INTERVAL, you will also have to set up the accepted precision/scale or start and end values.



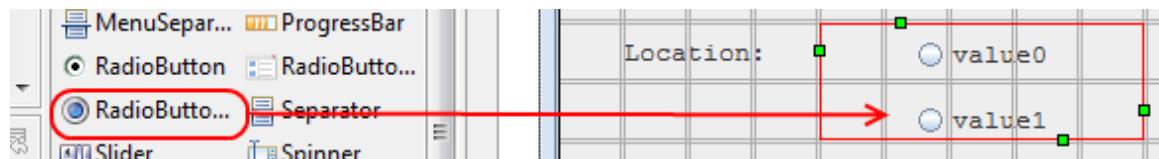
Adding RadioButtons

RadioButtons may prove useful when it is necessary to create a field for restricted input, where the user is able only to select among available variants. To create a RadioButton group, perform the following steps:

1. Add a *RadioButtonList* widget and set its **identifier**, **width** and **height**. For normal display, the height in rows should correspond to the number of radio button list items you want to add. To have two radio buttons, we create a *RadioButtonList* with two rows:

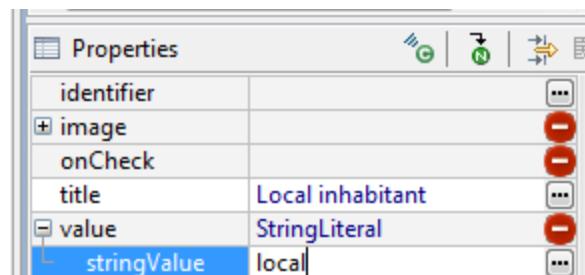


The *RadioButtonList* widget is empty and transparent by default. It should be filled with *RadioButtonListItems* that are also selected from the **Palette**:



2. Select *RadioButtonList* items in the **Structure** view one after another and set up their properties:

- Title - The text of the label that will be displayed to the item.
- Value - The value that will be passed to the corresponding variable when the item is selected during the input. The Value property consists of the data type and value sub-properties:

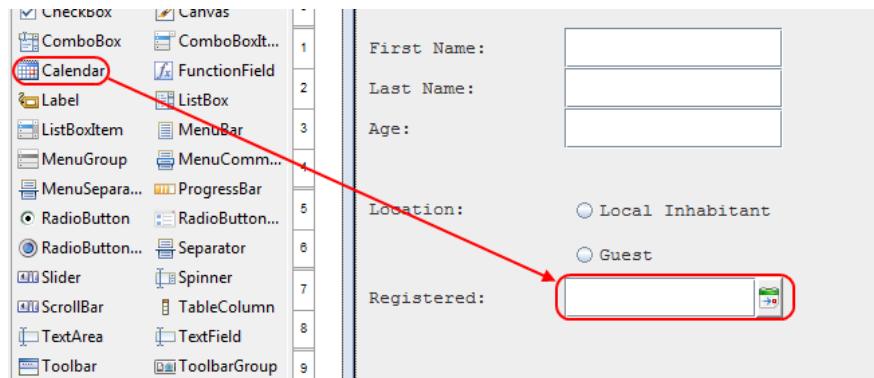




Adding Calendar

If the user is expected to input some DATE type variable it is reasonable to include the *Calendar* widget to the form which will facilitate inputting allowing to select the date with the help of the mouse.

To add a *Calendar* widget to the form select it from the **Form Widgets** folder in the **Palette** view and click on the target cell of the **grid** container (next to the Registered label). Adjust the field size the same way you did it for the labels, the button and the text fields:



To make the further references easier set up the *Calendar* widget identifier, let it be "reg_date":



Adding a Picture

As we are creating some kind of an application form it is logical to place the applicant's photo there.

So, to add a picture to the form, please, perform the following steps:

1. Add a **Label** to the right upper form area (next to the *fname* textfield).
2. Adjust the field size to it (in our case the label is 4 cells in height and the same in width).
3. Double click the **Label** to remove its text.



4. Set up the label identifier, let it be `app_photo`:

The screenshot shows the Lycia Form Editor interface. On the left, there's a tree view of form components under 'content-c'. A red box highlights the 'label-(app_photo)' node. Below it, the 'Properties' panel shows the 'identifier' field set to 'app_photo', also highlighted with a red box. On the right, the 'General' tab of the form grid displays several fields: 'First Name', 'Last Name', 'Age', 'Location' (with radio buttons for 'Local', 'Inhabitant', and 'Guest'), and 'Registered'. A red box highlights the 'Label' component in the list of available components.

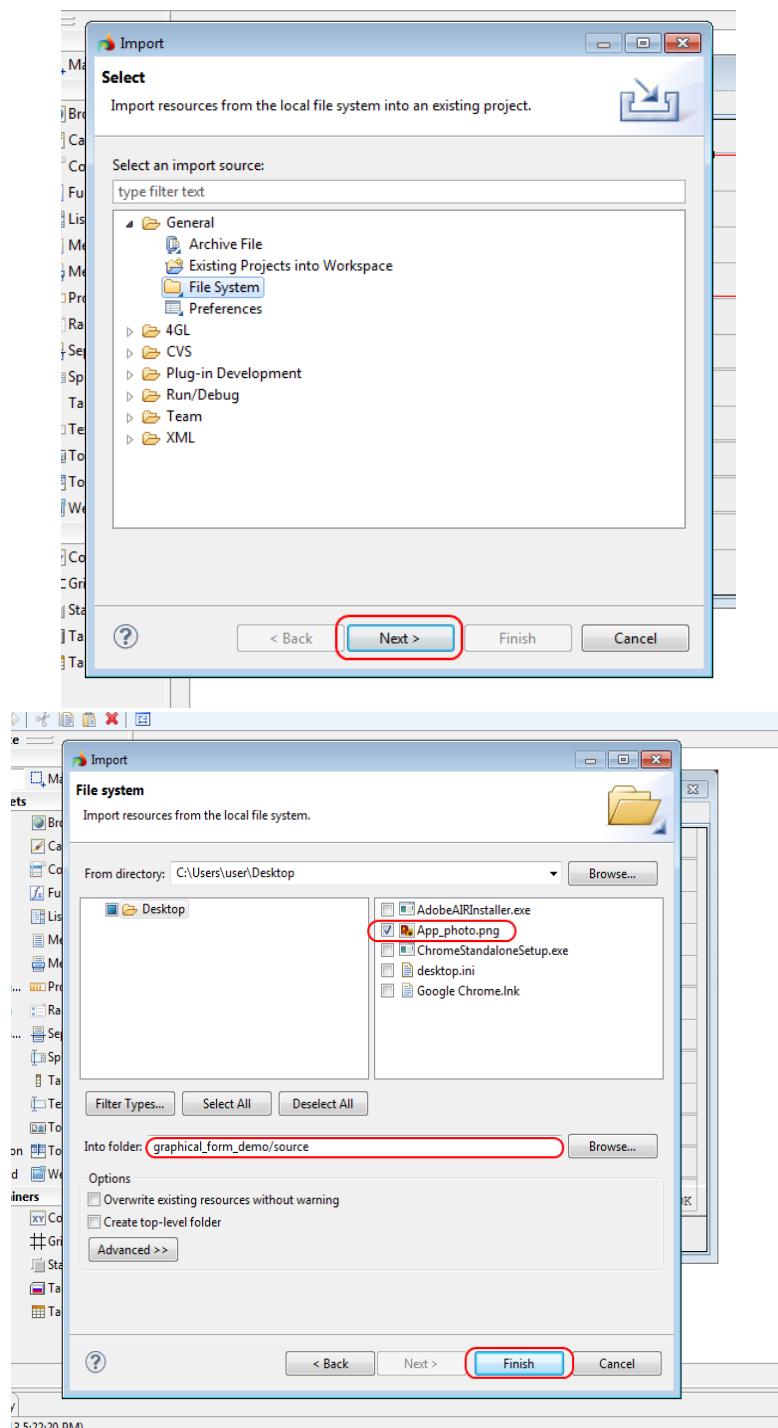
5. Import the picture from the local file system into your project:

- select the "Import" option from the context menu (right click on the application):

The screenshot shows the Lycia Project Explorer. A context menu is open over a project item named 'form1'. The 'Import...' option is highlighted with a red box. Other options in the menu include 'New', 'Delete', 'Rename', 'Copy', 'Paste', 'Build', 'Rebuild', 'Clean', 'Deploy to External Location...', 'Deploy to Server', 'Export...', 'Run As', 'Debug As', and 'Team'.

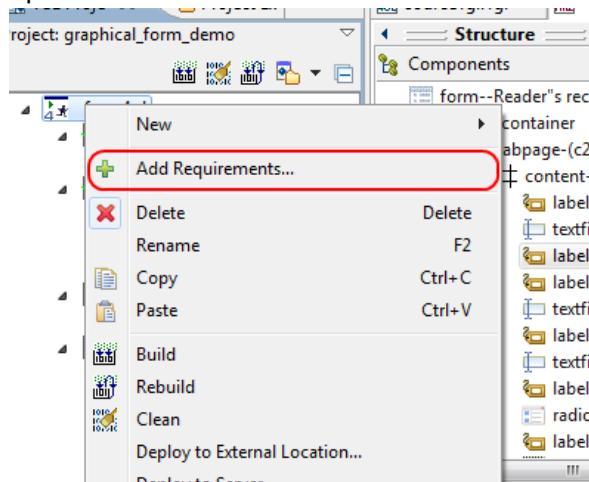


- select an import source, indicate the directory and the target folder:

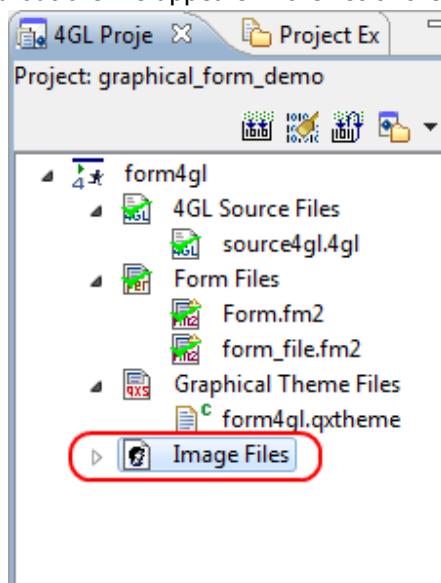




- right click on the application. Select the "Add requirements" option from the context menu and add the required file:

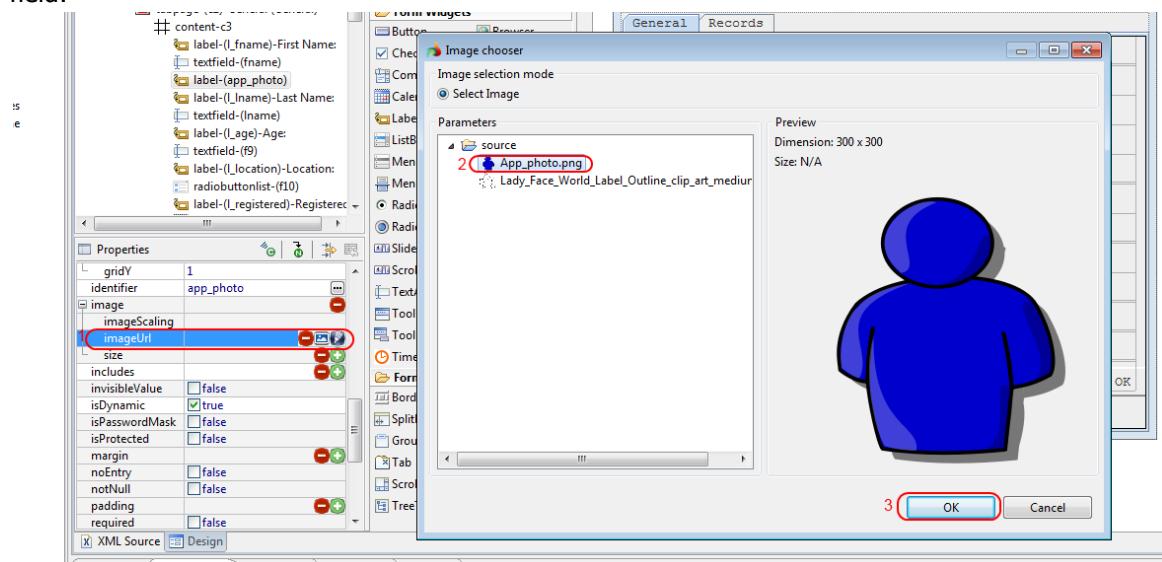


When this action is carried out the file appears in the list of the application files:

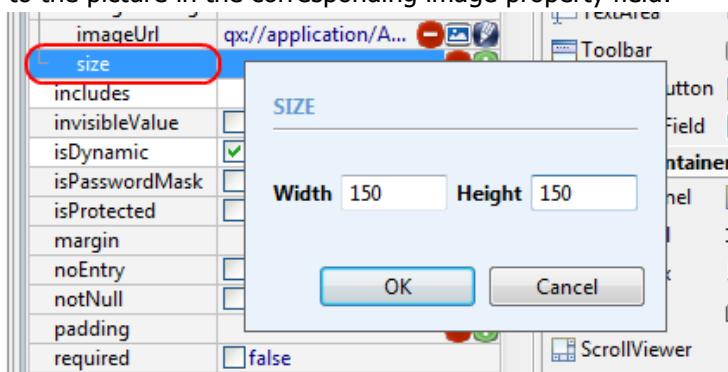




- add the picture to the form: left click on the image button in the **imageUrl** property field:



- adjust the size to the picture in the corresponding image property field:





All the actions carried above will result in the following form view:

The screenshot shows a software interface for creating a screen record. At the top, there are two tabs: 'General' (which is selected) and 'Records'. The main area contains the following fields:

- First Name:** An input text field.
- Last Name:** An input text field.
- Age:** An input text field.
- Location:** A radio button group with two options:
 - Local Inhabitant
 - Guest
- Registered:** An input text field with a small calendar icon to its right.

In the upper right corner of the form area, there is a large blue placeholder icon of a person. In the bottom right corner of the entire window, there is a 'Save' button.

Creating a Screen Record

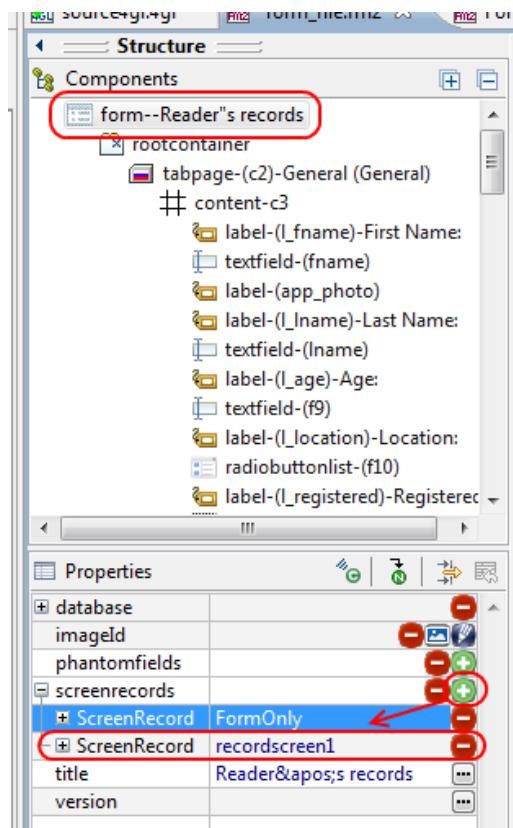
Now let's create a Screen record of `fname`, `lname` and `age` fields, `location` RadioButtonList and `reg_date` Calendar.

To add a new screen record, perform the following steps:

1. Select the form from the **Structure** view form tree.

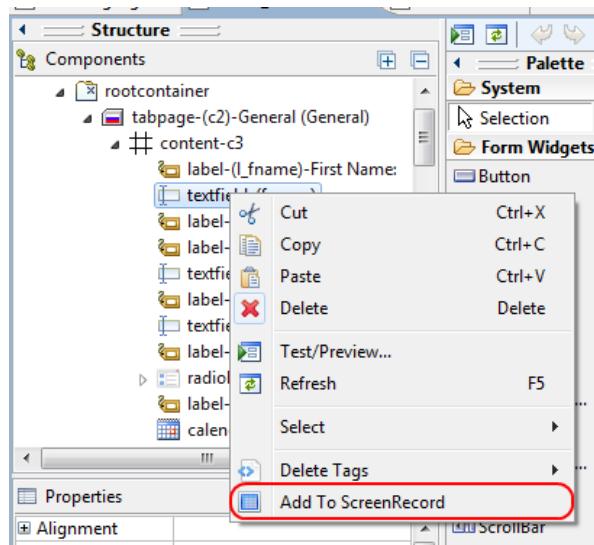


2. Press the **Add** button to the right from the **screenrecords** property. This will create an empty screen record with a default identifier:



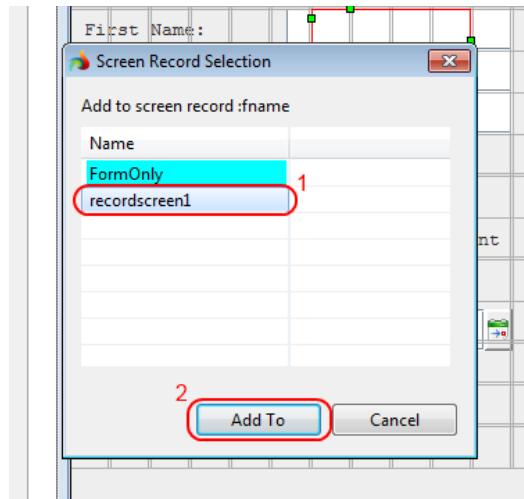
3. add an *fname* field to the screen record:

- right click on the *fname* field in the **Structure** view, select **Add to ScreenRecord** option:





- choose the screen record to which the field should be added in the dialog window appeared, press the **Add To** button:



- the field should appear in the selected **Screen Record list**:

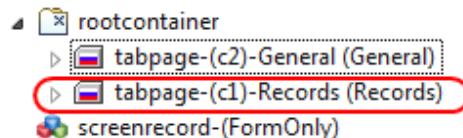


4. Add an *lname* and *age* fields, *location* radiobuttonlist and *reg_date* calendar to the screen record by selecting them at once from the **Structure** view. As a result all the fields added should appear in the **Screen Record list**.

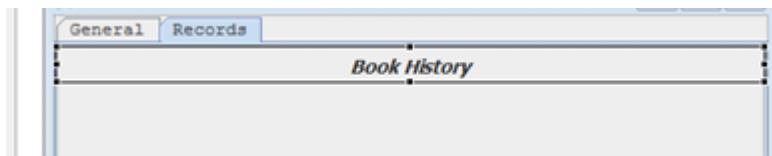
5. For convenience set the screen record **identifier** to "app_info_rec"

Adding a Stack Panel

- Select the Record tab in the **Structure** view form tree:

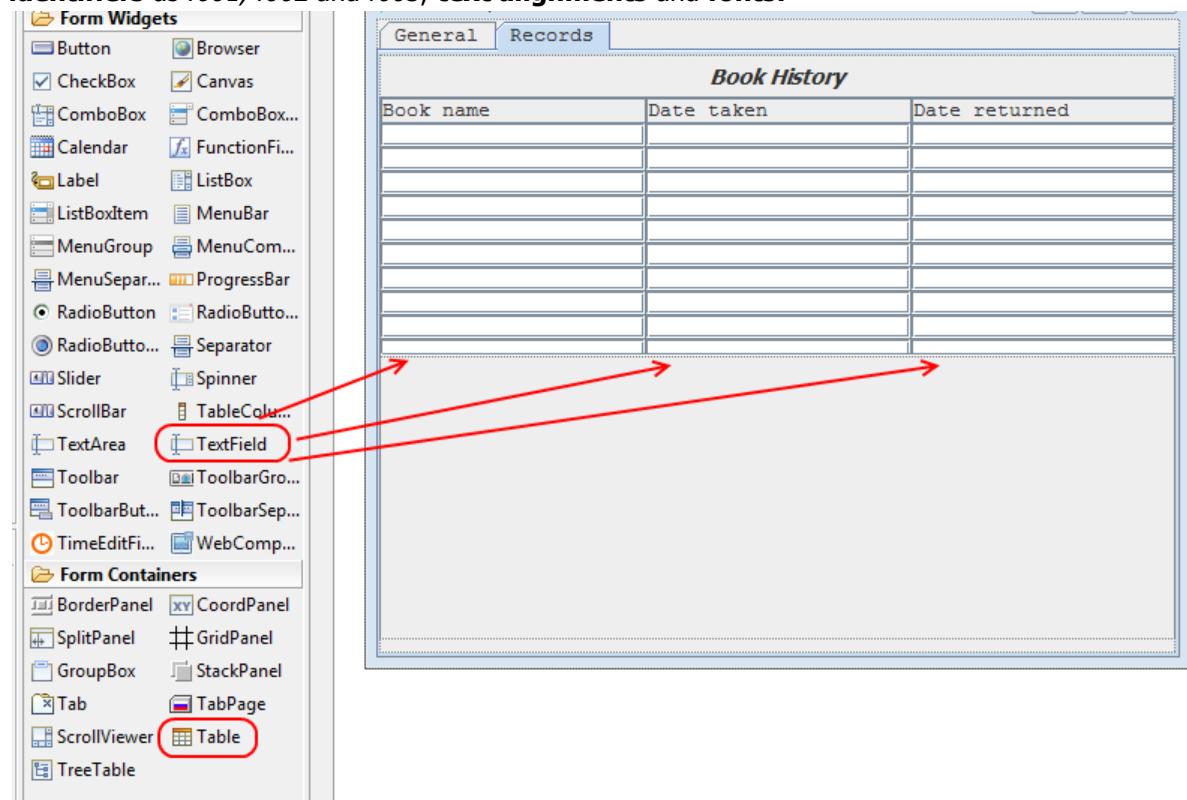


- Add a **Stack** panel to it from the **Form Containers** folder in the **Palette** view.
- Leave the orientation property vertical (the default stack panel orientation): this supposes that the objects placed within the **Stack** panel will be arranged one under another.
- Add a **Label** as a header. Set up its **identifier**, **Text Alignment** and **Font**:



Adding a Table

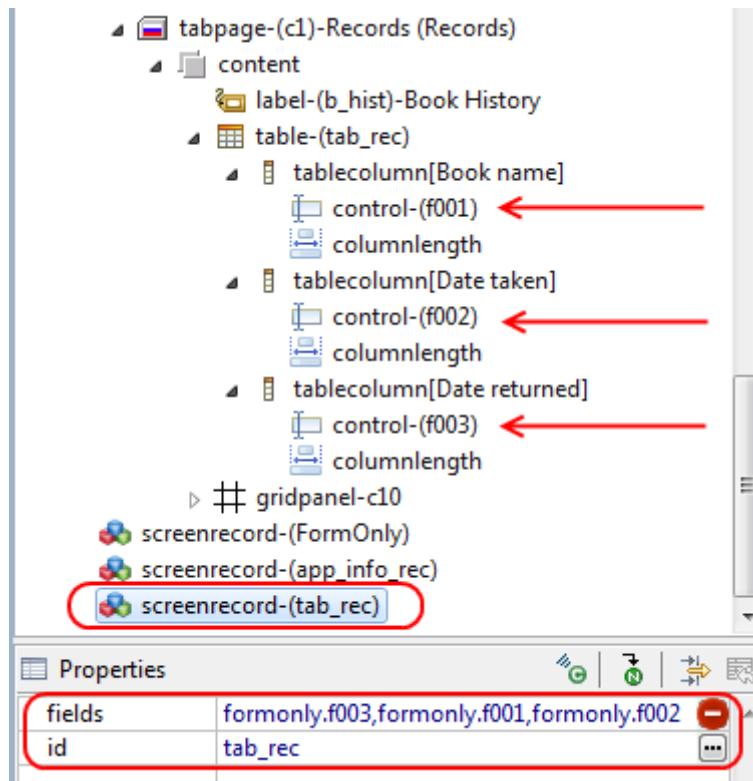
1. Add a **Table** to the **Records** tab from the **Form Containers** folder in the **Palette** view.
2. Adjust the table size by dragging its lower border.
3. Add three **Text Fields** to the table from the **Form Widgets** folder in the **Palette** view. This will result in three columns added to the table.
4. Change the column names to "*Book name*", "*Date taken*", "*Date returned*". Set up their **identifiers** as *f001*, *f002* and *f003*, **text alignments** and **fonts**:



Consider that Lycia window Builder automatically creates a screen record bound to the table and adds there all the fields added to this table.



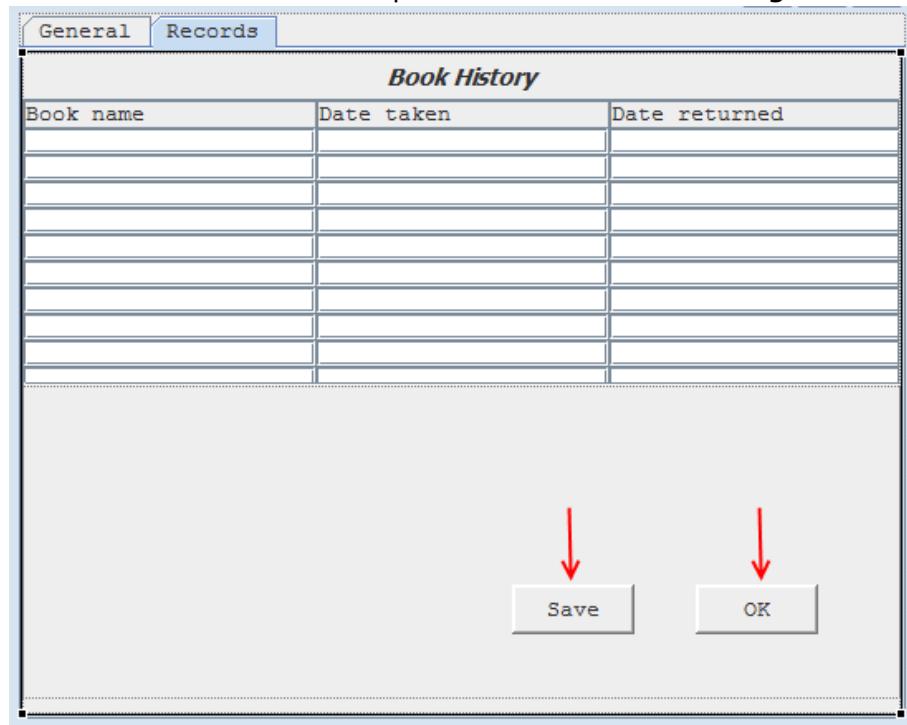
4. Set the screen record identifier to "*tab_rec*" for convenience as it will be used in the INPUT ARRAY statement in the source code:



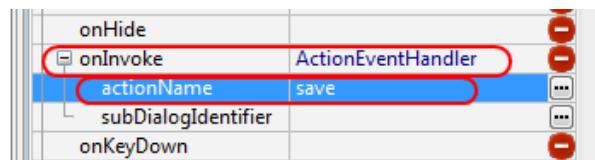


Adding Buttons to the Record Tab

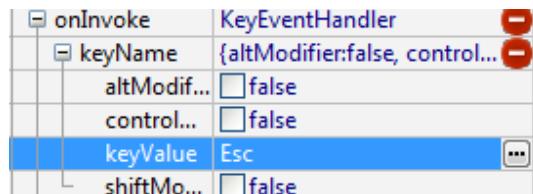
1. Add a **GridPanel** to the **Records** tab from the **Form Containers** folder in the **Palette** view.
2. Add two **buttons** to it from the **Form Widgets** folder in the **Palette** view. Change their names to "Save" and "OK". Set up their **identifiers** and **text alignments**:



3. For the "Save" button set an **OnInvoke** event to the **ActionEventHandler**. Define the **actionName** as *Save*. This will make the application to save the data entered by the user to the **Records** tab at runtime:



4. For the "OK" button set an **OnInvoke** event to the **KeyEventHandler**. Define the **keyValue** as *Esc* in the corresponding **On Key** block. This will bind the exit event with the *Esc* key. The keyValue will be later used for the INPUT statement in the source code:





Displaying a Form

Display the form with the help of the OPEN WINDOW WITH FORM statement. In the source code it will look as follows:

```
OPEN WINDOW my_form AT 1,1 WITH FORM "form_file"  
ATTRIBUTE (BORDER)
```

where

- *my_form* stands for the variable of the WINDOW data type defined in the main block,
- *form_file* is the name of the form file to be displayed in a newly opened window.

Adding a Dialog

DIALOG

```
--Inputs first name, last name and age to the text fields, location to  
the RadioButtonList and date to the Calendar located in the General tab  
  
INPUT my_record.* FROM app_info_rec.*  
  
END INPUT  
  
--Inputs data to the table located in the Records tab  
  
INPUT ARRAY books_list FROM tab_rec.*  
  
END INPUT  
  
--Make the program to save the updated information when the user presses  
the button "Save"  
  
ON ACTION(save)  
  
--Opens a dialog box when the user presses the button "Save"  
CALL fgl_winmessage("Save","The information is saved","info")  
  
--Make the program to quit when the user presses the button "OK"  
  
ON KEY(esc)  
  
--Opens a dialog box when the user presses the button "OK"  
CALL fgl_winmessage("OK","The program will be closed in a  
second","info")  
  
EXIT DIALOG  
  
END DIALOG
```



Using the Debugger

This chapter will explain how to use the debugger within Lycia. We will use an example program to take the developer step-by-step through the important processes involved. Any program compiled using Lycia can be debugged using the LyciaStudio.

Preparing for Debugging

To learn how to use the Lycia Debugger, we will create a demonstration program. Firstly, create a new project with the name 'debugging_sample' using the menu option **File -> New -> 4GL Project**, as demonstrated previously in Chapter 3. Then right-click the project in the 'Project Explorer' view, choose **New -> 4GL Program** to create a new 4GL program with the name 'sample'. After this, next right-click the project again and select **New -> 4GL Source File** to create a 4GL Source File with the name 'sample'.

In the source code text editor, enter following code for the file sample.4gl:

```
GLOBALS
DEFINE
    a INTEGER,
    b INTEGER,
    c INTEGER
END GLOBALS
DEFINE
    leave CHAR (2)
MAIN
LET a=1
CALL first_function() RETURNING b
LET c=b-a
CALL second_function()
IF c>0 THEN
    DISPLAY "C is equal to ", c AT 5, 2
    PROMPT "Press any key to exit" FOR leave
    EXIT PROGRAM
ELSE
    DISPLAY "C is equal to 0 or less than 0" AT 5, 2
    PROMPT "Press any key to exit" FOR leave
    EXIT PROGRAM
END IF
END MAIN

FUNCTION first_function()
DEFINE
    r_b INTEGER
PROMPT "Enter value for variable B: " FOR r_b
RETURN r_b
END FUNCTION
FUNCTION second_function()
DISPLAY "A is equal to ", a CLIPPED, 1 space, "- it is a constant value"
AT 2, 2
```



```
DISPLAY "B is equal to ", b CLIPPED, 1 space, "- it is the value you've  
entered" AT 3, 2  
DISPLAY "A has been subtracted from B to get C" AT 4, 2  
END FUNCTION
```

Save and build the above example file. Any program compiled using 'Lycia' can be debugged using the 'LyciaStudio', or by using the debug command line tools. A compiled program can be executed either in run mode or in debug mode:

- In run mode (standard method of compiling), the program executes but the execution will not be suspended or examined.
- In debug mode, execution can be suspended and resumed, variable values may be inspected or changed and expressions may be evaluated.



Debugging

The Lycia debugging tool lets the developer see what is going on "inside" a program whilst it executes step by step. There may be a situation where there is a problem in the code which cannot be revealed while compiling, but which occurs only during run time. Using the debugger, the developer can step through a program until a fault occurs and identify the cause. The debugger is included into LyciaStudio to help a developer solve such problems.

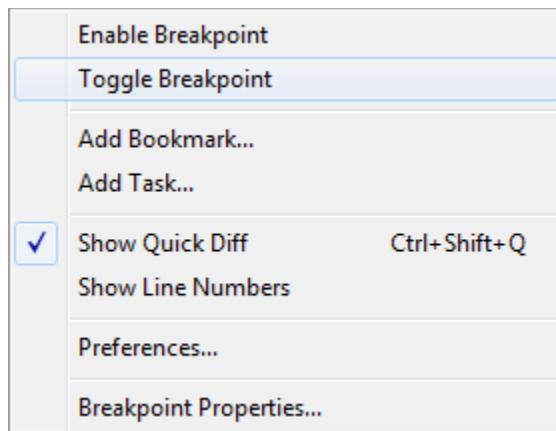
Now, as the program has been built, it can be run in debug mode. But to begin debugging, breakpoints need to be set.

Setting Breakpoints

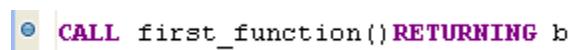
The points at which a developer wishes to suspend the execution of a program and begin executing it step by step are marked with the help of breakpoints. If a program is executed in debug mode, it will execute until it hits the first breakpoint. After this, the execution will be suspended and the 'Debug' view will be opened within the Studio, offering the developer the option to manage the debugging process and to view the debugging results.

A breakpoint suspends the execution of a program at the location where it is set. It can be set with the help of the context menu in the 'Editor' view on the left side of the LyciaStudio by default. To set breakpoints, follow these steps:

- 1) Right-click the left side of the 'Editor' in the grey vertical bar view next to line "CALL first_function() RETURNING b" (line 14 in the file). Select the option "**Toggle Breakpoint**" as shown below:



Now you have one breakpoint set as show below:



- 2) Set the second breakpoint next to the line "LET c=b-a" (line 16) as explained above.



-
- 3) Set the third breakpoint next to the line "CALL second_function()" (line 18) as explained above.

Now there will be 3 breakpoints set as shown below:

```
CALL first_function() RETURNING b
LET c=b-a
CALL second_function()
```

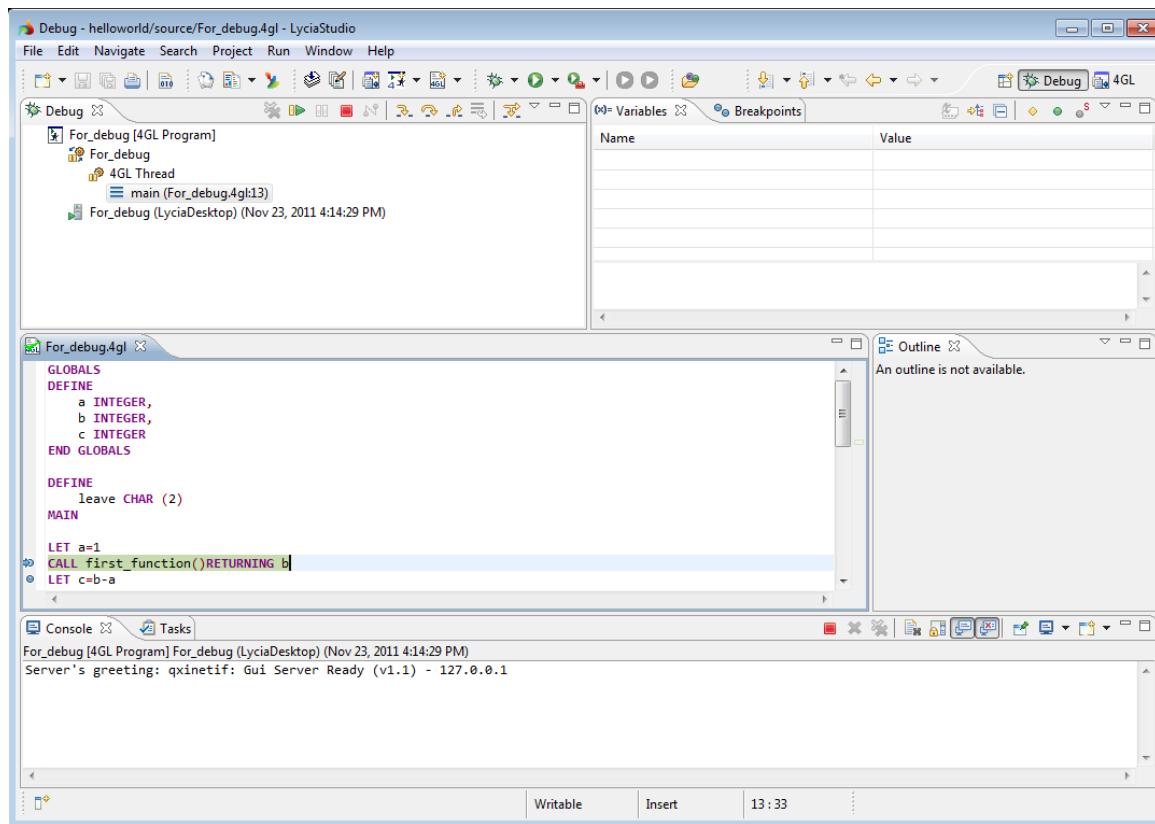
The developer can delete a breakpoint by right-clicking it and selecting the "**Toggle Breakpoint**" option once more. It is also possible to create a breakpoint by double-clicking the grey vertical bar at the left side of the 'Editor' view and to delete it by double-clicking it repeatedly.

Launching Program in Debug Mode

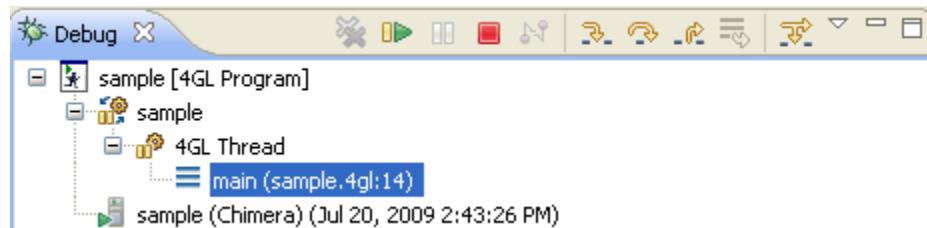
There are two ways to start debugging a compiled program using LyciaStudio:

1. Right-click the program in the '4GL Project' view and select the "**Debug As...**" option. Then choose one of the graphical clients you have installed or you may choose the character mode.
2. Use the Debug menu by pressing the  **Debug** button

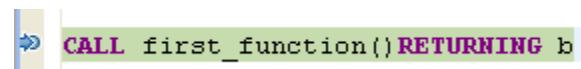
When the program hits the first breakpoint, the execution will be suspended and a pop-up window will appear asking whether you want to switch to the 'Debug' perspective. Press **OK** to switch to the 'Debug' perspective. It will be loaded as shown below:



The Debug perspective displays the stack frame for the suspended threads for each target that you are debugging in the 'Debug' view. Each thread within the program appears as a node in the tree. It displays the process for each target you are running. If the thread is suspended, the stack frames are shown as child elements.



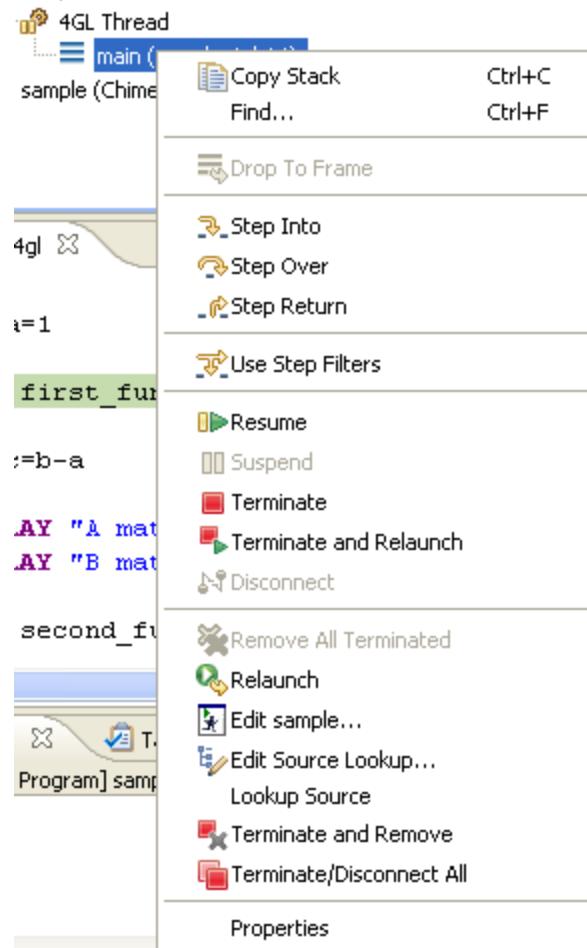
Currently, in our example, the 4GL thread is suspended and has one child element which is the result of the first breakpoint on line 14 in the main section of the 4GL Source File example. This breakpoint will also be correspondingly highlighted in the 'Editor' view as shown below:





Executing a Program Step by Step

The tools necessary for step by step execution can be found either in the context menu of the current thread called by right-clicking a 4GL Thread, or its child elements, as shown below.



The same menu option can be found on the toolbar of the 'Debug' view.



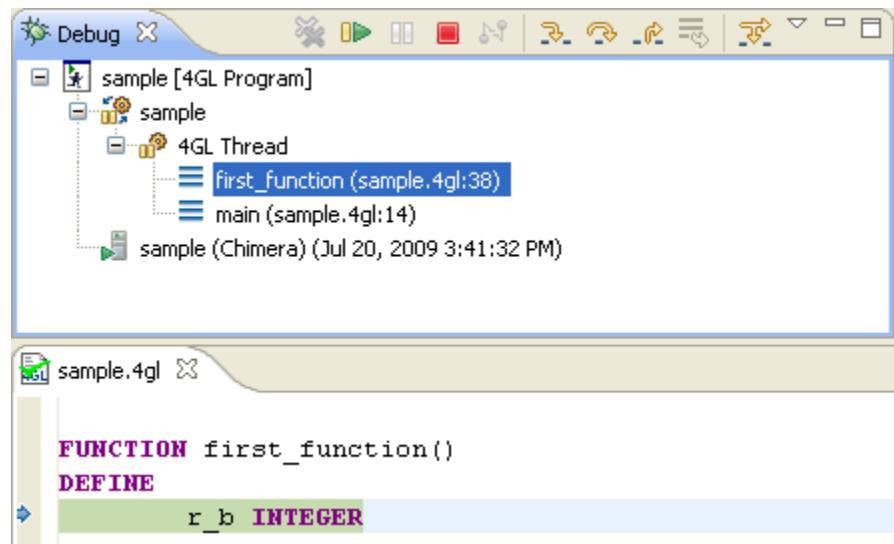
There are 3 main functions on the debugging toolbar which control how the program is executed. The following commands can be used to move through the code step by step:

- 1) The **Step Into** command (depicted by the icon, or key press **F5**) is used to step into the method called by the currently executed line of code, and executing it as expected.
- 2) The **Step Over** command (depicted by the icon, **F6**) is used to step over the method called by the currently executed line of code without entering or executing it.



- 3) The **Step Return** command (depicted by the icon, **F7**) is used to return from a method which has been stepped into.

Pressing F5 or the **Step Into** button will step into the highlighted statement. This means that the function called by the currently executing CALL statement will be stepped into and executed.



	<p>Note: When using GUI clients with the debugger, they cache the screen information and buffer and such actions may not appear on the debugger straight away. The screen is refreshed when there is user interaction so another step further than expected may be required.</p>
--	---

This action will move the cursor to the function which is called in the highlighted line. Another stack frame will appear within the 4GL Thread list, because the cursor has moved from the main section to one of the sub-functions.

If you press the **Step Into** button again, the execution of the program will come to the point where input from user is required, at which point the code can no longer be stepped through until the input is performed. After the required data has been input, it is possible to continue stepping into the code.

If the developer does not wish to step through the function further, you can press F7 or the **Step Return** button. This function is used to return from a function which has been stepped into, thus you do not need to step through the whole function. After using **Step Return** the cursor will be carried to the line following the statement that has called the function.



```
CALL first_function() RETURNING b
LET c=b-a
```

Even though we return from the function, the remainder of the code inside the function that has just been left will be executed normally and the value returned.

Now the cursor is at the second breakpoint. It is a simple statement and does not call a function, so next select either **Step Into**, **Step Return** or **Step Over** and the cursor will move on to the next line, where the third breakpoint is located:

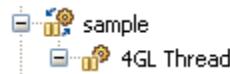
```
CALL second_function()
```

Press F6 or select **Step Over** button to move to the statement that follows the breakpoint without stepping into the function.

```
CALL second_function()
IF c>0 THEN
```

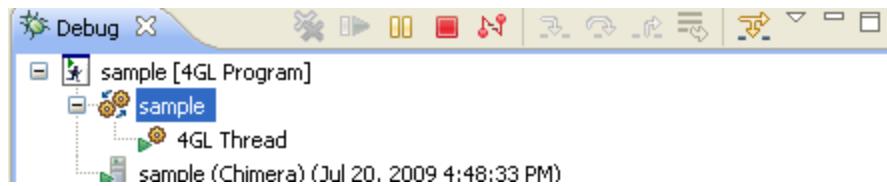
Even though the function was never stepped into and the cursor is not moved into the second function, it will still be executed normally.

Now that there are no more breakpoints, you can press the **Resume** button (depicted by the ➤ icon) and the program will execute freely until the end. If you press **Resume** on any line between two breakpoints, the program will execute freely until the next breakpoint is hit, at which point the execution will be suspended. The buttons **Resume** and **Suspend** (depicted by the ⏸ icon) toggle between continual and step by step execution of a program. If a Program is in the suspended mode, the pause marks are displayed in the 'Debug' view and the **Resume** button is enabled, as shown below:



	Note: You can execute a program step by step only if the program is in suspended mode.
--	---

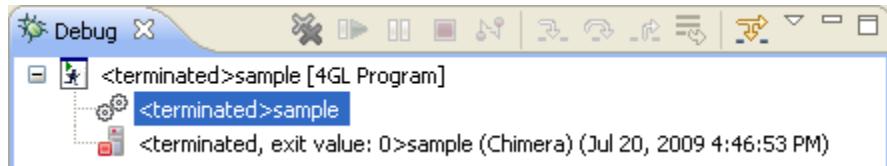
If the program runs freely, then play marks are displayed in the 'Debug' view and the **Suspend** button is enabled. There are no stack frames and if they have been displayed previously, they will now be removed.



Terminating and Relaunching a Program

When debugging, the developer can step through the code until the end of the program, resume the execution of the program, or terminate the program.

If the developer wants to terminate the execution of the selected program, use the **Terminate** option (depicted by the icon) in the context menu or on the 'Debug' view toolbar. This will terminate the selected program and the 'Debug' view will indicate that the program is no longer running.

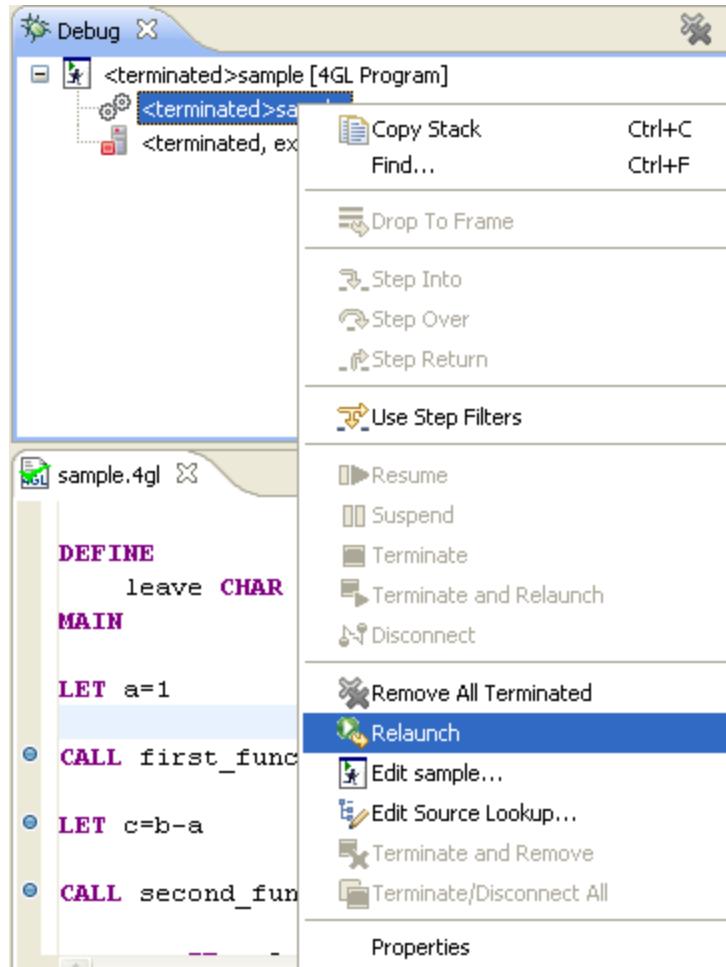


You must relaunch the program if you wish to continue debugging by creating a new instance of the program and debugging that.

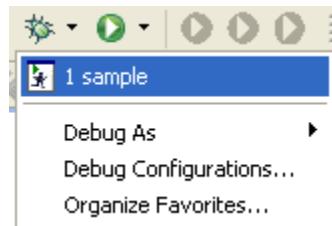


To relaunch the terminated program:

- 1) Choose the **Relaunch** option (depicted by the icon) in the context menu:



Or select it from the list of the programs that have been recently launched in debug mode. To do so, open the 'Debug' menu:

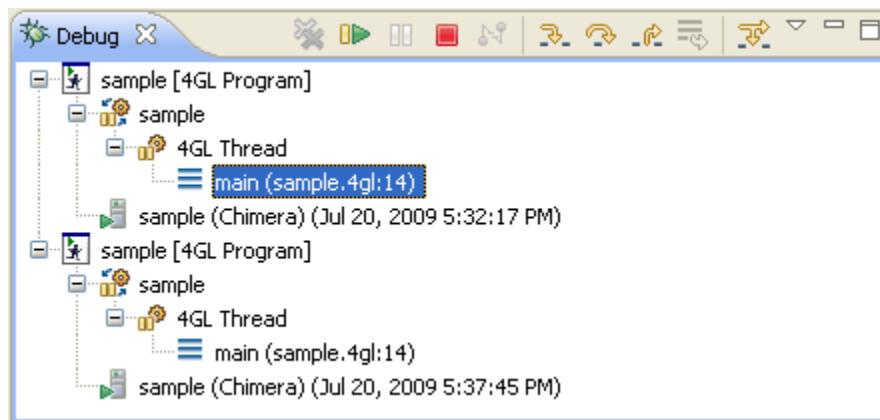




The program will be launched using the debug configurations which will be discussed later in this chapter. To relaunch the last program that was launched in debug mode, press the **Debug As...** button directly.

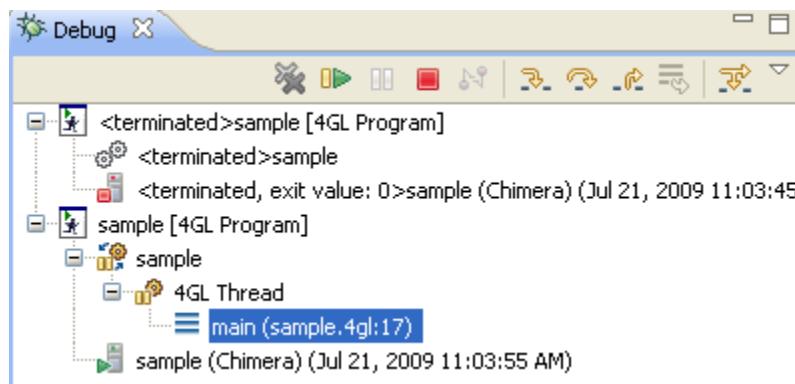
- 2) In the main menu select **Run -> Debug History** to launch one of the programs that have been debugged recently.

If you try to relaunch a program while it is still running, a new instance of the program will be launched and executed independently. You can step through the code of the currently selected instance of the program without affecting the execution of the other instances of the program.



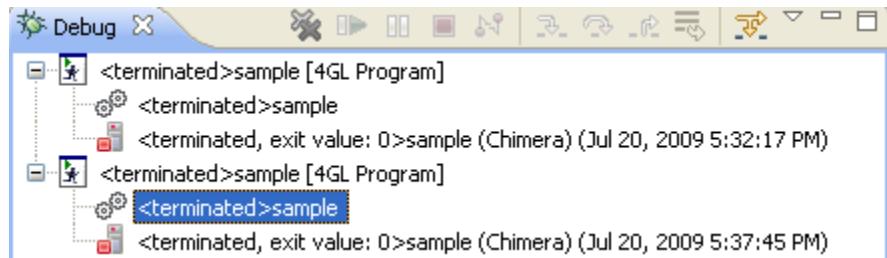
The program is executed from the start, disregarding the point at which it has been terminated or suspended.

To terminate a program and relaunch it, select the **Terminate and Relaunch** option (depicted by the icon) in the context menu. It will terminate the selected program and run a fresh instance of this program:



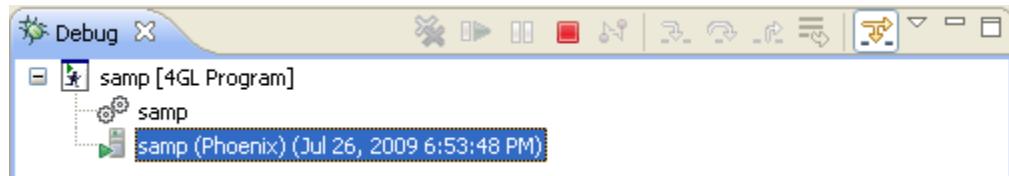


To terminate the execution of both instances of the program which are now running, select the **Terminate/Disconnect All** option (depicted by the icon) in the context menu. This will terminate all instances of all the programs that are currently running in debug mode:



To remove all terminated instances of the programs from the 'Debug' view, select the **Remove all Terminated** option (depicted by the icon) either from the context menu or using the 'Debug' view toolbar. The **Terminate and Remove** option (depicted by the icon) in the context menu can be used to terminate one running program and to remove it from the 'Debug' view.

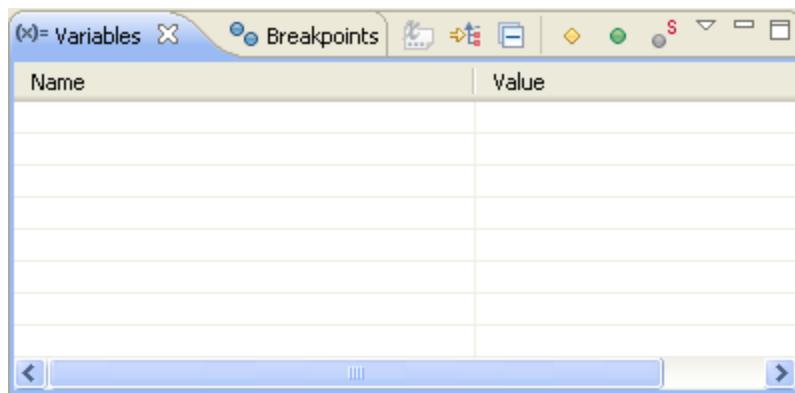
The **Disconnect** button (depicted by the icon) terminates the debug session and resumes all suspended threads. The program continues to execute, but it cannot be suspended and executed step by step, the only action available is termination of the program:



Variables View

The 'Variables' view in the LyciaStudio is used to monitor the values of the variables whilst the program is running in debug mode. When the 'Debug' perspective is opened, it can sometimes appear empty, as shown below. The variables view can be empty in the following situations:

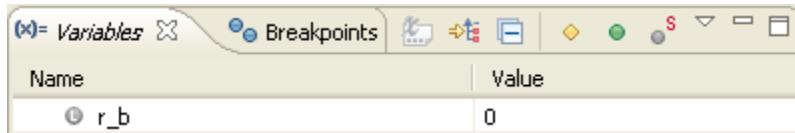
- when the currently selected thread is terminated;
- when there are certain circumstances met regarding a filter - for example, If all filters (module, global, special build in etc) are de-selected and there are no local variables



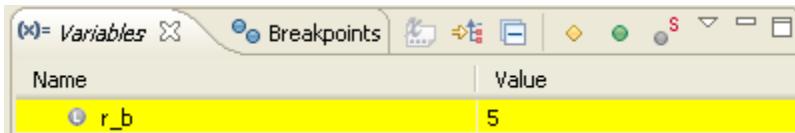
Monitoring Values of Variables

The 'Variables' view is empty to begin with because no variable type is declared. Defining the types of variables to display can be selected by pressing buttons on the toolbar of the view (depicted by the icon). If none of these buttons are selected, the 'Variables' view displays only the local variables when they are dealt with after their declaration.

Now try relaunching the program and **Step Into** the line next to the first breakpoint. Step through the function (using F5) until you get to the PROMPT statement (line 34). Pay attention to the 'Variables' view, as it now contains one variable 'r_b' which is a variable local to the first function and it has zero value:



Press the **Step Into** button once more and the program will now require input for this local variable. Enter '5' into the prompt field and press the Enter key. The value of the local variable will change to '5' and it will be highlighted in yellow, which indicates that the value of the variable has been changed:



If you return to the main section of the source file or **Step Return** from the function, the local variable will no longer be displayed.

To see other types of variables, use the corresponding buttons on the toolbar of the view. Each button corresponds to a specific type of variable:

- 1) Press the button; when this button is pressed, the built-in variables are displayed in 'Variables' tab, as shown below:



Name	Value
\$status	0
+ \$sqlca	RECORD
\$sqlstate	
\$int_flag	0
\$quit_flag	0
\$eflastkey	0
\$engine_sql_code	0
\$db_type	0
\$q4gl_client_type	qx_chimera

Note: the execution of the sample program does not influence the values of the built-in variables, so they will remain unchanged throughout the execution, and you may press the button again to hide the built-in variables.

- 2) Press the button; when this button is pressed, the module variables are displayed in 'Variables' view. We have only one module variable in the source file, as it is of character type no value is displayed for it at this point:

Name	Value
leave	

- 3) Press the button; when this button is pressed, the global variables are displayed in 'Variables' view. We have 3 global variables displayed below, and we also have the module variable displayed below them, as we have not pressed the button once again to hide it:

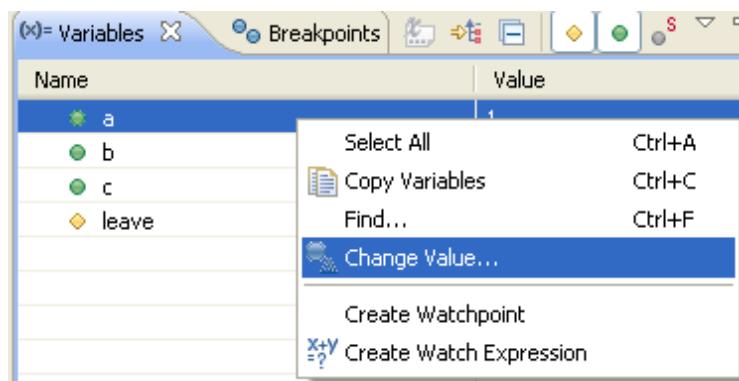
Name	Value
a	1
b	5
c	0
leave	

Notice that variable 'c' has zero value because at the point where the program is currently suspended, it has been declared but no value has been assigned to it. Variable 'a' has the value of 1. It is a constant value which is assigned to this variable before the first breakpoint. Variable 'b' has the value of 5 which has been returned from the first function.

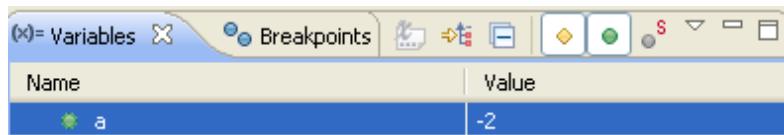


Changing the Value of a Variable

There is a way to change the value of any variable during run time without editing the source code, independently of any user interaction or code changes. Right-click the variable 'a' and choose the **Change Value** (depicted by the icon) option:



Enter '-2' as the value for variable 'a' in the pop-up window and press **OK**. The 'Variables' tab will now display the following:



The program will continue to execute using the new value for variable 'a', though according to the source code, its value still remains unchanged and is equal to 1.

As you step through the code, corresponding values are assigned to the variables including local variable 'r_b'. All 3 buttons can be selected and all 4 types of variables (built-in, global, module and local) can be displayed at the same time.

You can also change value of a variable by clicking on the variable you wish to edit in the variables list. A cursor will then appear and the developer can enter the new value in this field.

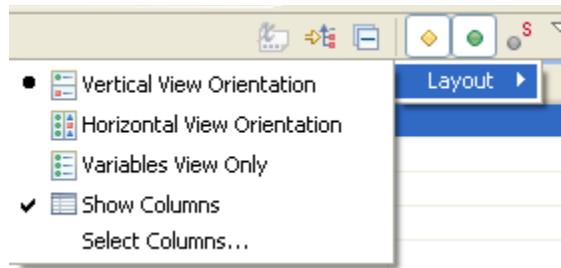
If you enter a value that is not compatible with the data type of the selected variable, an error dialog will appear. Note that it is possible to enter a number for the variable of CHAR or VARCHAR data type, because when a number expression is assigned to a CHAR variable, it is converted to a string.



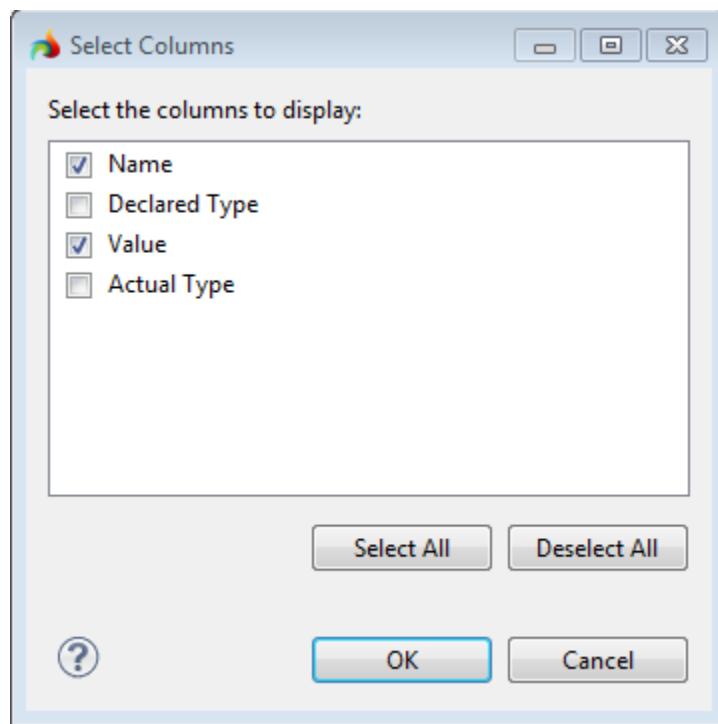
Note: The value you specify will be evaluated as a 4GL expression. It can include variable names; these will be substituted with the value of the given variables. e.g. if the value of x is 5, then x + 2 will be evaluated to be 7. For this reason, literal strings **must** be quoted, e.g. "customer_name", otherwise strings will be treated as variable names.

Customising the Variables View

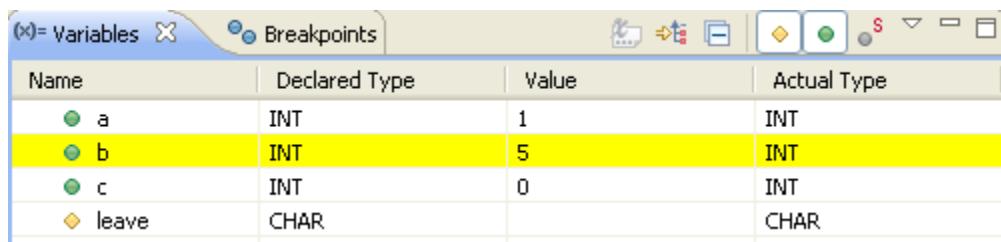
You can customise the look of the 'Variables' view by pressing the **View Menu** button (depicted by the  icon):



If the horizontal or vertical view orientation is chosen, one more field is added to the 'Variables' view, which displays the value of the currently selected variable. When the **Show Columns** option is selected, the **Select Columns** option is displayed. By default, 2 columns are displayed: 'Name' and 'Value', which can be customised with the help of the **Select Columns** option. If you choose this option, the following pop-up window will appear:



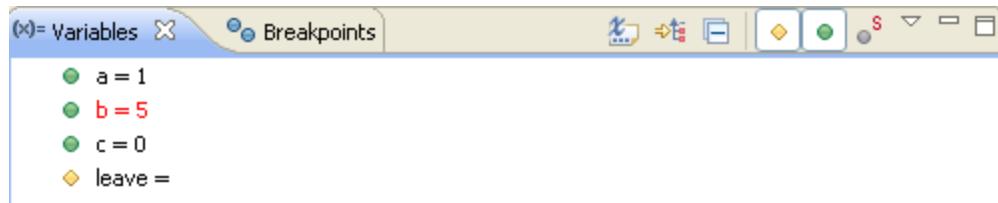
Next, press the **Select All** button and then press **OK**. Now your 'Variables' view will display four columns:

A screenshot of the Eclipse IDE's "Variables" view window. The window title is "(x)= Variables". It shows a table with four columns: "Name", "Declared Type", "Value", and "Actual Type". There are four rows in the table, each representing a variable: "a" (Value 1, Actual Type INT), "b" (Value 5, Actual Type INT), "c" (Value 0, Actual Type INT), and "leave" (Actual Type CHAR). The "Value" column for row "b" is highlighted with a yellow background. The "Variables" tab is selected in the top bar, and there are other tabs for "Breakpoints" and "Registers".

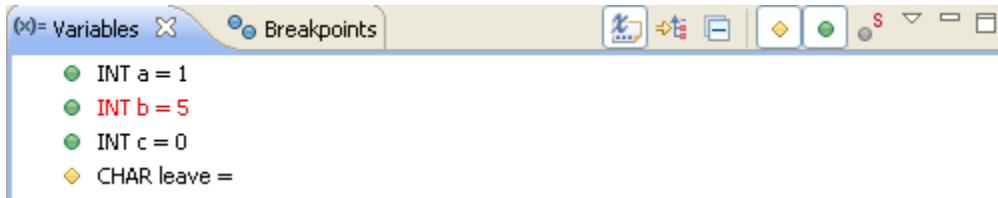
Name	Declared Type	Value	Actual Type
a	INT	1	INT
b	INT	5	INT
c	INT	0	INT
leave	CHAR		CHAR



If the **Show Columns** option is not selected, variables are displayed in one column as shown below:

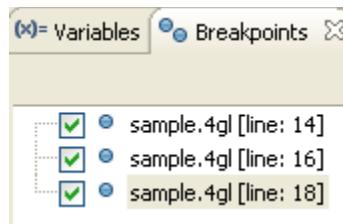


In this mode, it is not possible to choose a column to display the data type of the variables, yet it is still possible to see variables data types by pressing the button, which will result in placing data types next to the names of the variables as shown below:



Breakpoints View

Breakpoints are displayed in the vertical ruler of the 'Editor' tab and in the 'Breakpoints' tab. The 'Breakpoints' tab offers detailed information on the breakpoints:



Here we have 3 breakpoints. Each breakpoint contains information regarding its location.

Enabling and Disabling Breakpoints

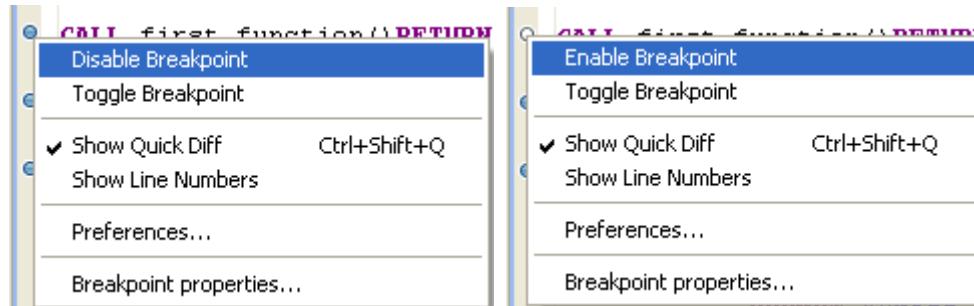
Breakpoints can be enabled and disabled:

- An enabled breakpoint causes a thread to suspend whenever the breakpoint is encountered. Enabled breakpoints are represented by a blue circle .
- A disabled breakpoint will not suspend a thread. Disabled breakpoints are represented by a white circle .

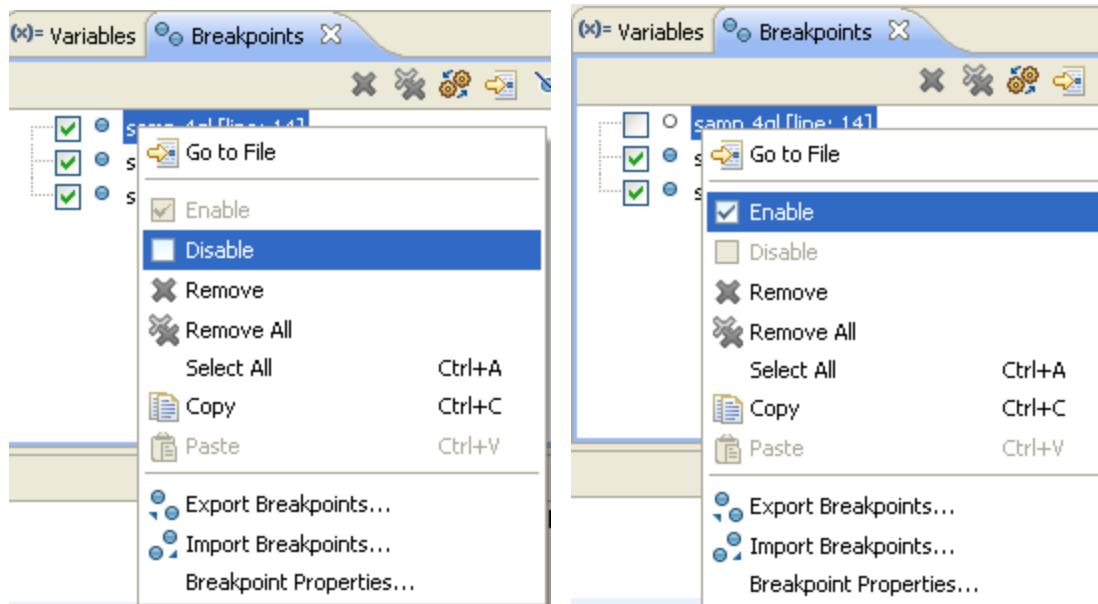


There are several ways to enable or disable breakpoints:

- Right-click a breakpoint in the 'Editor' view and select either the **Disable Breakpoint** option or the **Enable Breakpoint** option, depending on whether the breakpoint is currently enabled or disabled, as shown below:



- Right-click a breakpoint in the 'Breakpoints' view and select either the **Disable Breakpoint** option or the **Enable Breakpoint** option, depending on whether the breakpoint is currently enabled or disabled:



- Select/ deselect the check box next to a breakpoint.
- Enable or disable breakpoint using the **Breakpoint Properties...** option (see the 'Breakpoint Properties' section)

The 'Breakpoints' view offers a number of other commands:

- Press the Skip button (depicted by the icon) to skip all active breakpoints. The breakpoints will not be disabled, but will be skipped during run-time and will not cause any threads to suspend until the session terminates.

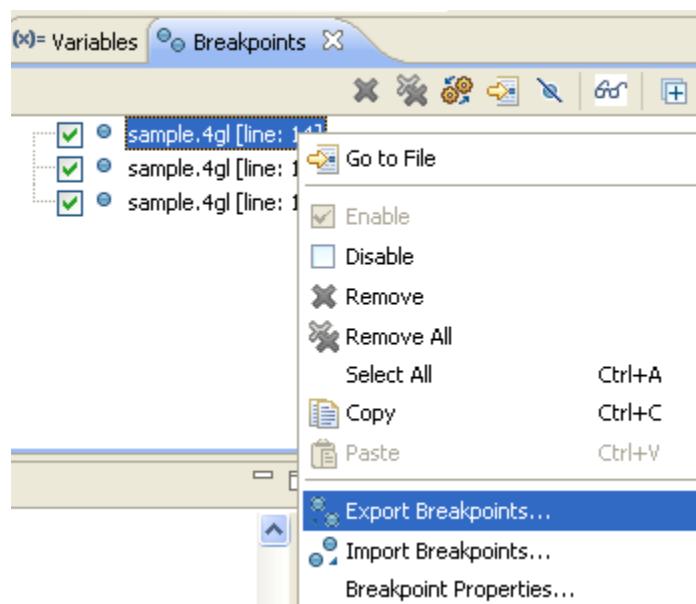


- Select the Go **to** File command (depicted by the icon) to open the associated source file for the breakpoint, make it active and highlight the location of the breakpoint. If the resource is already open, it is made active and the breakpoint location is highlighted. From our example, attempt to close the 'Editor' view which displays the source file, select the third breakpoint and press the Go **to** File button. The source file will be opened in the 'Editor' view and the line next to the third breakpoint will be highlighted. The same result may be achieved when double-clicking a breakpoint in 'Breakpoints' view.
- To delete a selected breakpoint, use the **Remove** command (depicted by the icon) in the context menu of the breakpoint or on the toolbar of the tab. To delete all breakpoints, use the **Remove All** command (depicted by the icon).

Exporting and Importing Breakpoints

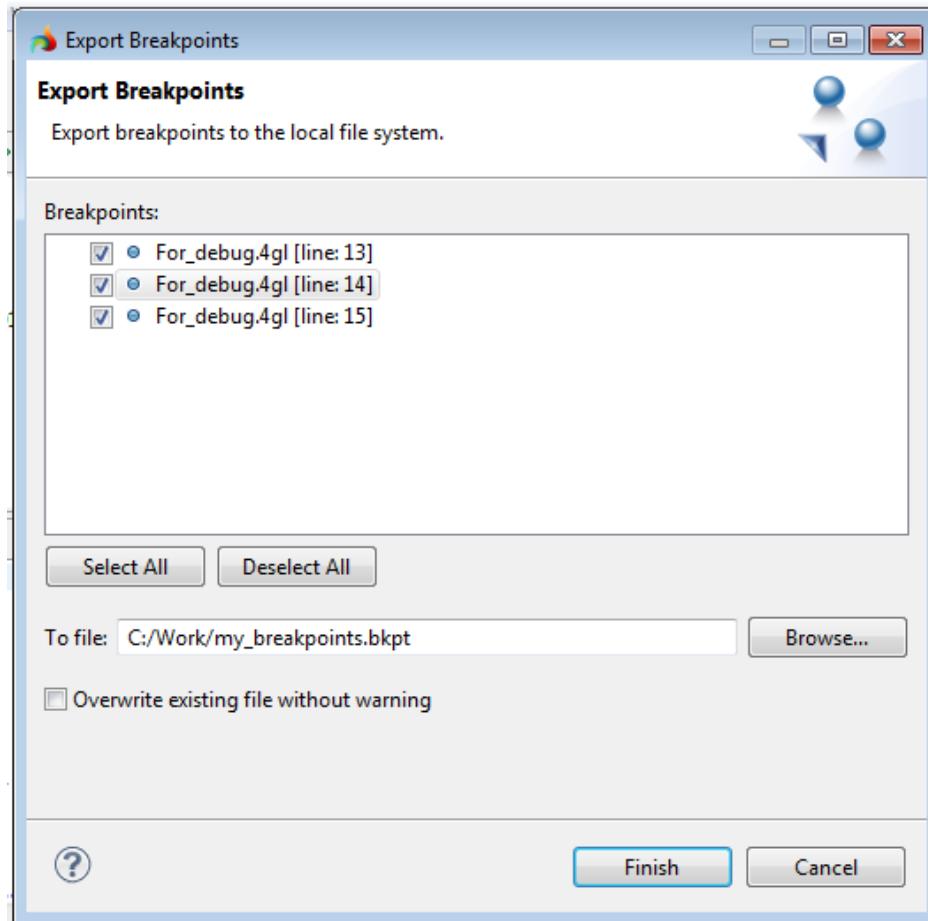
It is possible to export currently set breakpoints into a file and then import them from that file when needed at a later date.

Open the context menu of any of the breakpoints and choose the **Export Breakpoints...** (depicted by the icon) option:



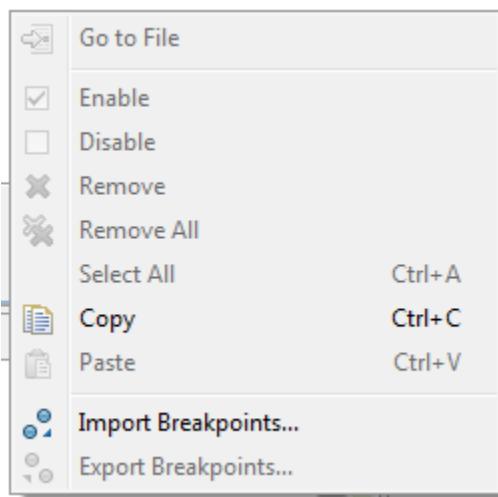


The export dialog box will be displayed as follows:

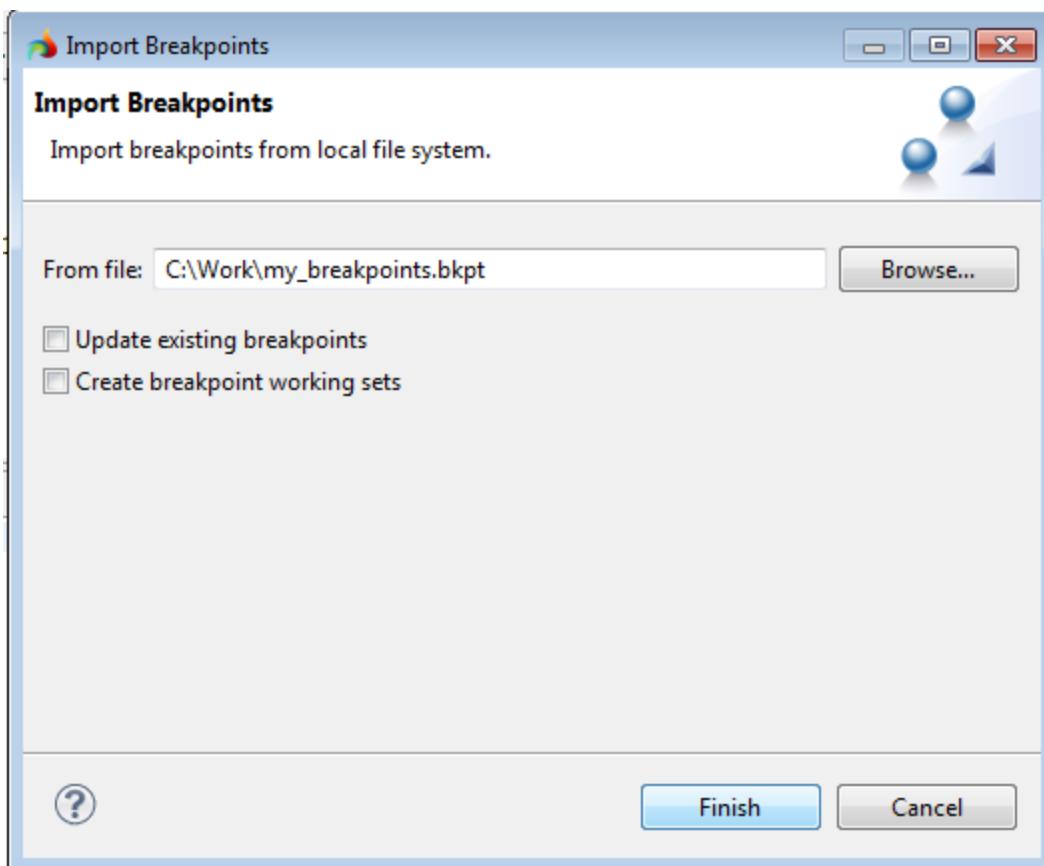


Press the **Select All** button to select all the breakpoints, press the **Browse...** button, select the directory to which you would like to export the breakpoints, type the name of the file and press **Finish**. The file with the extension .bkpt will be saved to the selected directory.

Now delete all the breakpoints from the source file using the **Remove All** command. After that, select **Import Breakpoints...** option (depicted by the icon) from the context menu of the 'Breakpoints' view as shown below:



Within the import dialog box, press the **Browse...** button to select the file to which has just had the breakpoints exported to it, and press **Finish**.



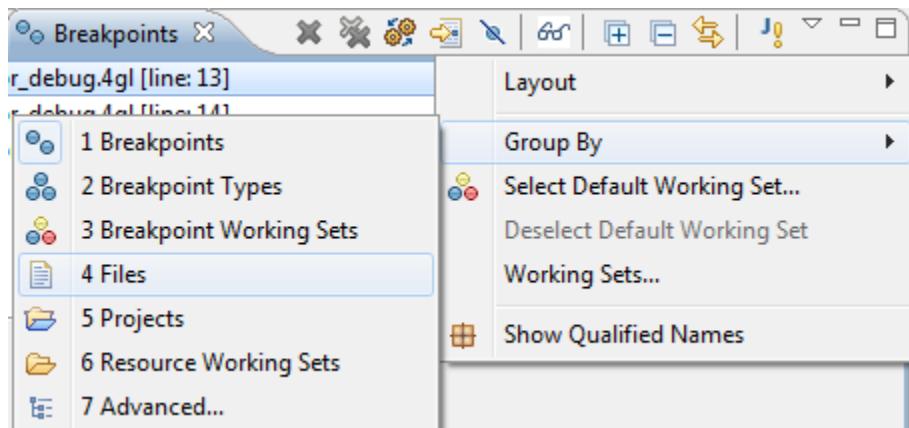
You will see all the breakpoints in their same places as before.



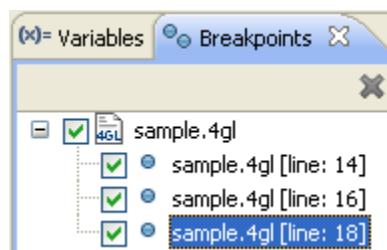
Grouping Breakpoints

In large projects, you may have many breakpoints for different source files and it is easier to work with them if they are organized in some manner. The 'Breakpoints' view has options which help to organize breakpoints. These are grouping and creation of working sets and can be accessed by pressing the **View Menu** button (depicted by the icon):

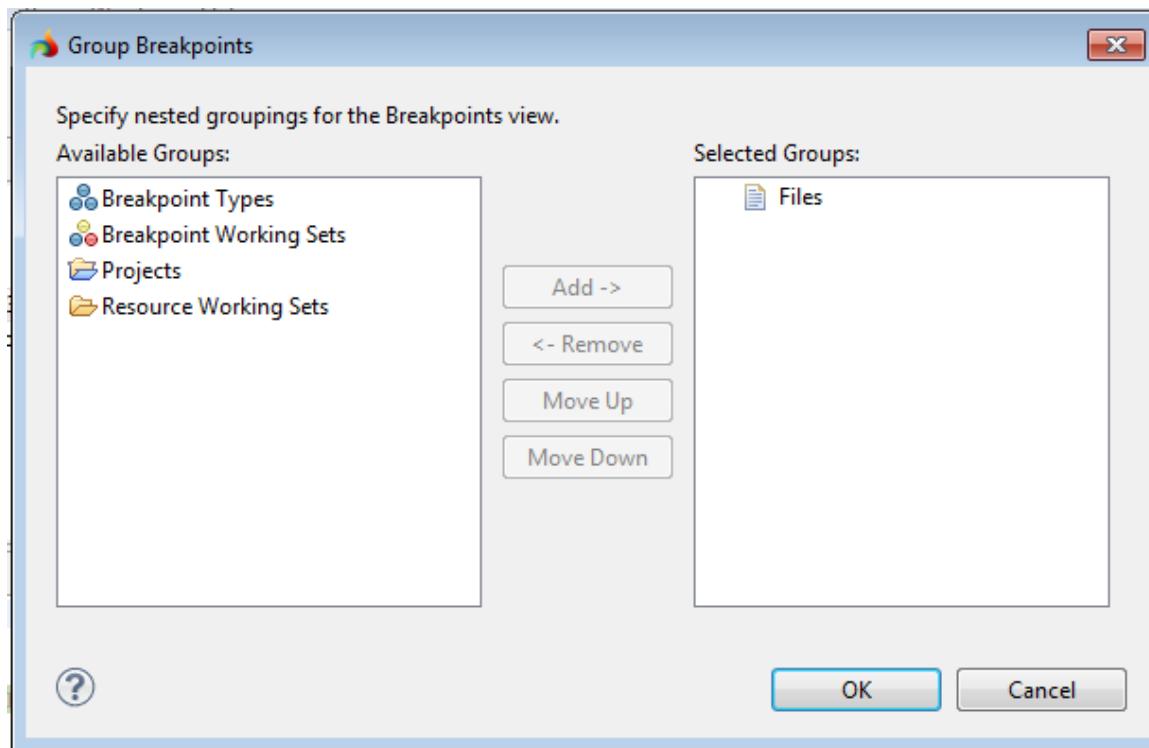
- The **Group By** option groups breakpoints by their source files, projects, breakpoint types, etc. By default, the breakpoints are grouped 'by breakpoints', which means that breakpoints are not grouped by any common feature.
 - 1) To group breakpoints by source files, select the **Group By** option and choose **Files** as shown below:



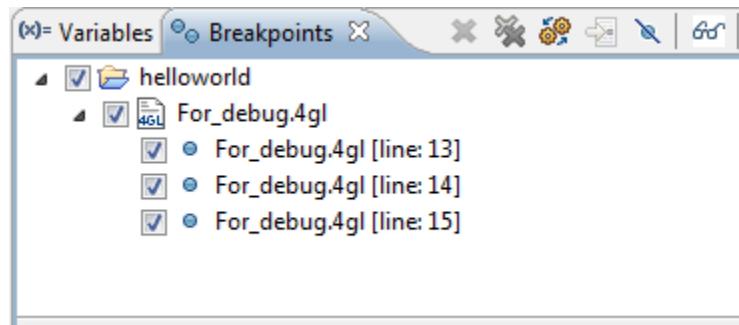
- 2) This will result in grouping the breakpoints by the source files to which they belong. If you deselect the main checkbox next to name of the group, all dependent checkmarks will be removed and all the breakpoints in this group will be disabled. Likewise breakpoints can be grouped by project, by working sets, etc.



- 3) It is possible to make complex groups for breakpoints. For example, you can make up a group displaying only breakpoints for a certain source file within a certain project, which is very convenient when working with a number of projects. Select the **Advanced...** option from the menu shown above to create a complex group. First add 'Projects' and after this, add 'Files' and press **OK**.



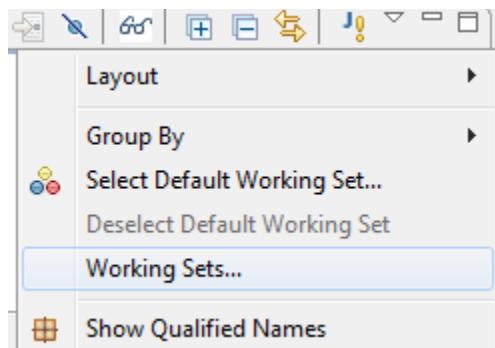
Now you have your breakpoints grouped by the projects and by source files:



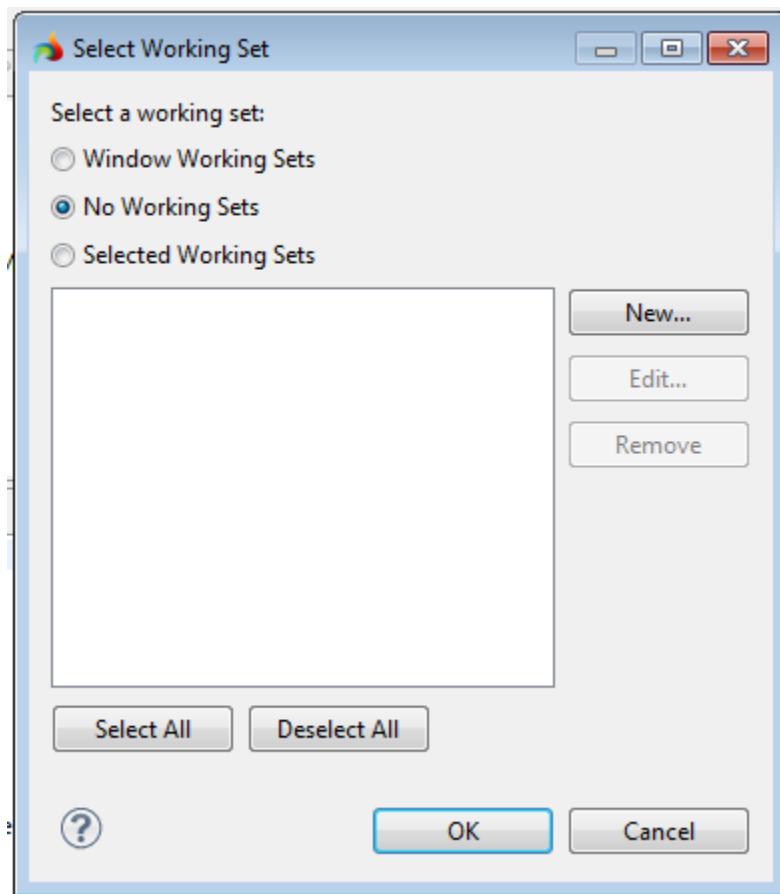
- The **Working Sets...** option offers another property to group your breakpoints by. A working set is a group of breakpoints which you create by adding breakpoints to it without regard to their source files, projects, etc. It is possible to create several working sets and to choose the default working set. After you have created a working set, breakpoints can be grouped by working sets.



- 1) To create a working set choose the **Working Sets...** option.

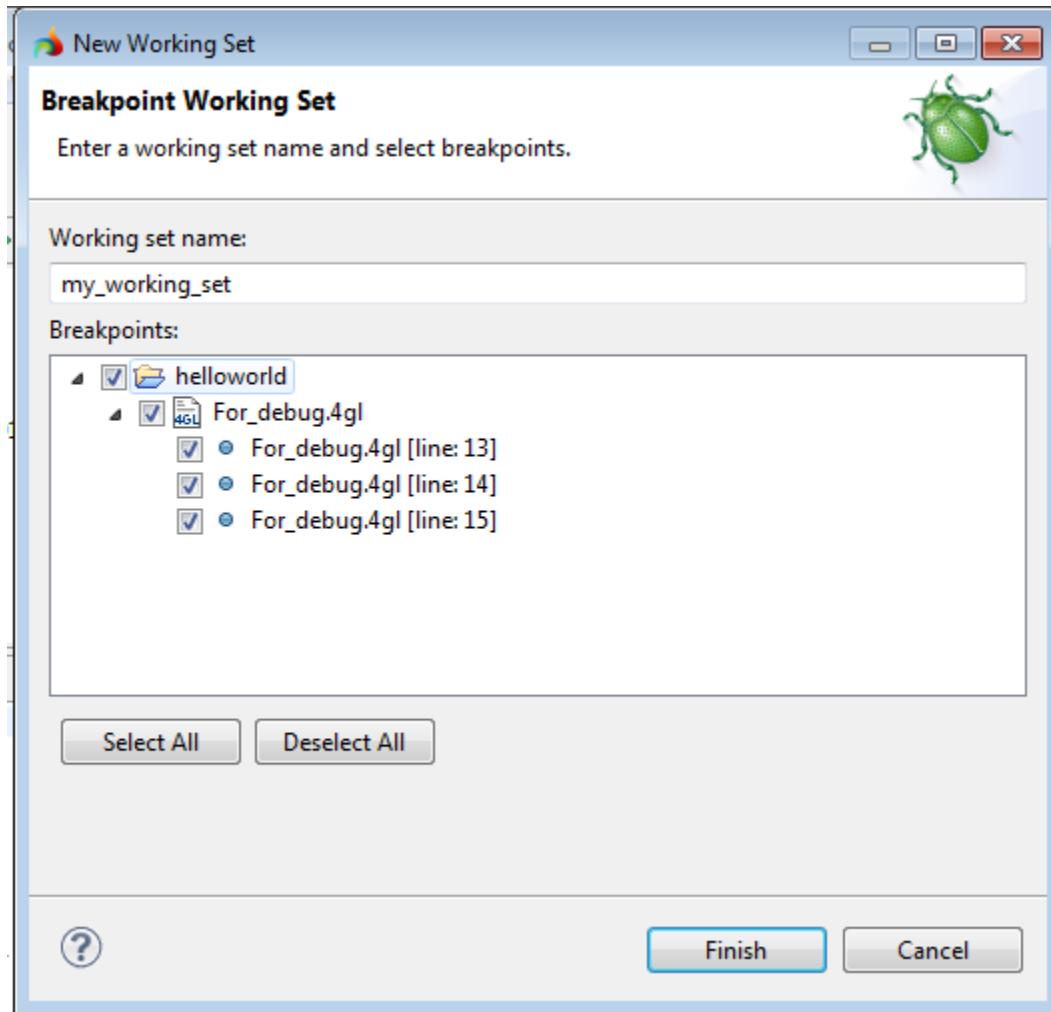


- 2) In the dialog box, select the **New** button:





- 3) Enter a name for the new working set. Here you can also choose which breakpoints you want to include in this working set. Press the **Select All** button to include all the existing breakpoints to the working set and press **Finish**:

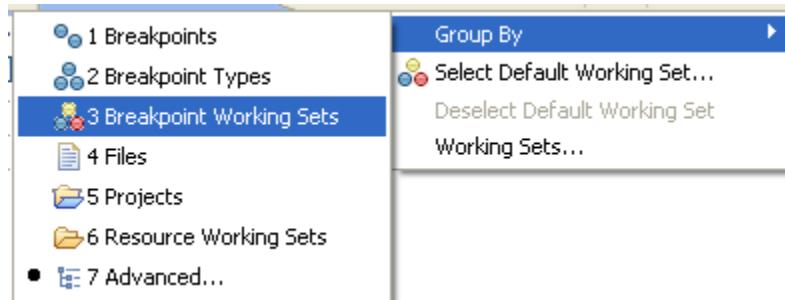


	Note: The same breakpoint can be added to several working sets.
--	--

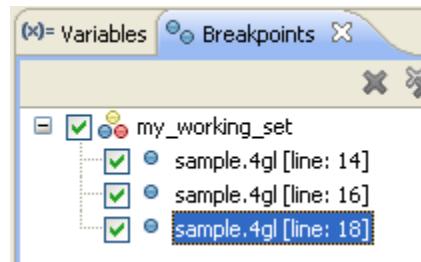
- 4) Press **Finish** in the dialog box that will appear.



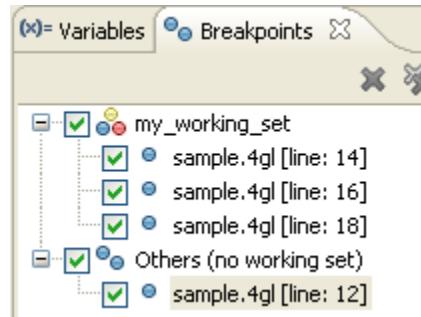
- 5) The breakpoints in the 'Breakpoints' view will remain grouped by project and source files. To group them by working sets, select **Group By > Breakpoint Working Sets**



- 6) Now the breakpoints will be grouped by the newly created working set:

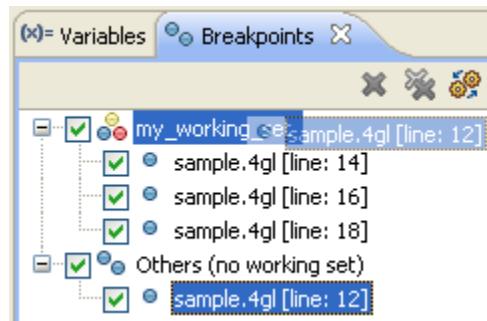


- 7) Try adding another breakpoint at the line 'LET a=1' (line 12). It will be displayed in the 'Breakpoints' view outside the working set:





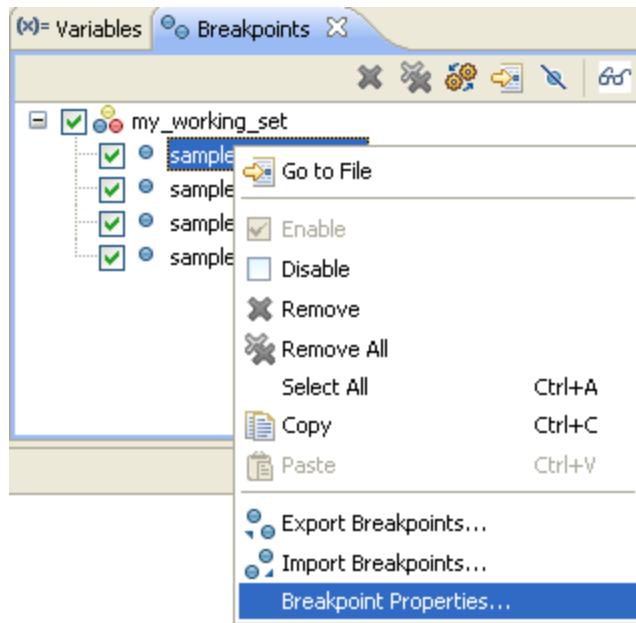
- 8) You may want to include a new breakpoint into your working set. Choose the **Working Sets...** option, then select the working set, press the **Edit** button and make any necessary changes. It is also possible to add a breakpoint to a working set by dragging and dropping it to the working set, as shown below:



- There is another button which helps to organize breakpoints when working with several different projects. If the button on the toolbar of the 'Breakpoints' view is selected, only the breakpoints for the currently selected project are displayed.

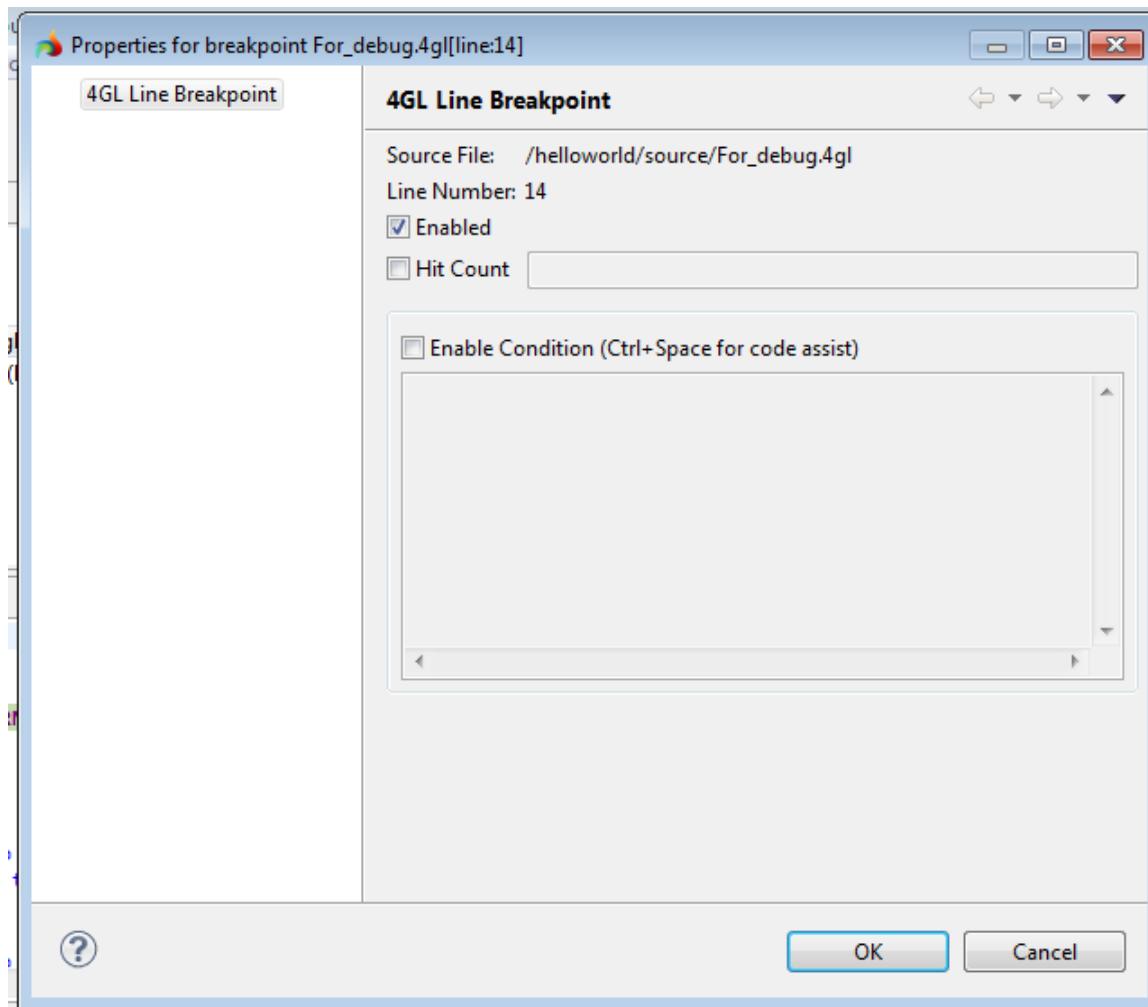
Breakpoint Properties

To view the properties of a breakpoint first select the required breakpoint, then choose the **Breakpoint properties...** option from the context menu of the breakpoint.





Any information on the selected breakpoint will be displayed in a dialog box:



- The **Enabled** option is used for enabling or disabling the selected breakpoint.
- The **Hit Count** option is used to determine when your program should suspend on the breakpoint. If a breakpoint has a hit count of N, where N is any integer, execution will suspend when the breakpoint is encountered for the Nth time. After being hit, the breakpoint is disabled until either it is re-enabled or the hit count is changed. It will also be re-enabled after the program has been terminated or re-launched.

- 1) To enable the hit count option for the first breakpoint, select the 'Hit Count' option and set 1 for hit count value as shown below:

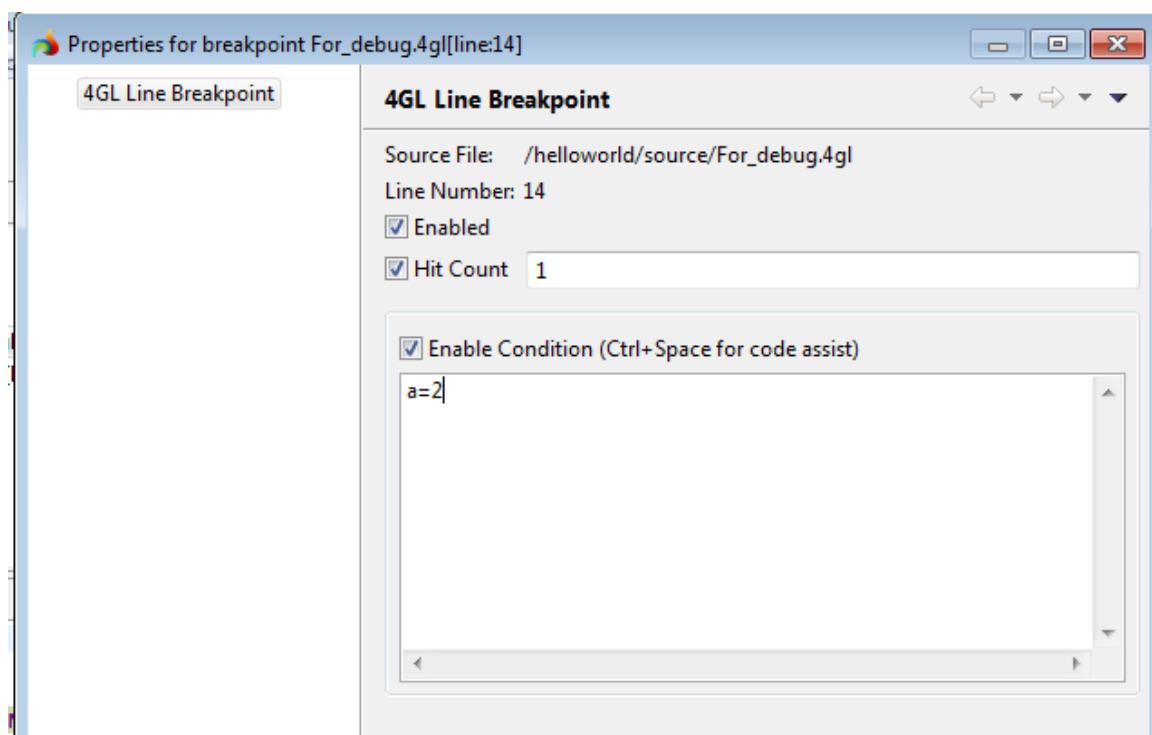
<input checked="" type="checkbox"/> Hit Count	1
---	---

- 2) Launch the program in debug mode and step into the first breakpoint. After the program hits the first breakpoint for the first time, the breakpoint becomes



disabled as the number of hits corresponds to the hit count set and the next line is executed.

- The **Enable Condition** option is used to enable the ability to provide a custom condition for the breakpoint. Each breakpoint can have a unique condition that determines at what point the breakpoint will be hit. A condition for a breakpoint can be any logical expression that evaluates to either true or false. If a condition is enabled, the breakpoint will not suspend the execution of the program unless the condition is met. If the breakpoint with an enabled condition is disabled, it will not suspend the execution even if the condition is met.
 - 1) To enable the condition for the first breakpoint, select 'Enable Condition'. Enter 'a=2' into the field and press **OK**:

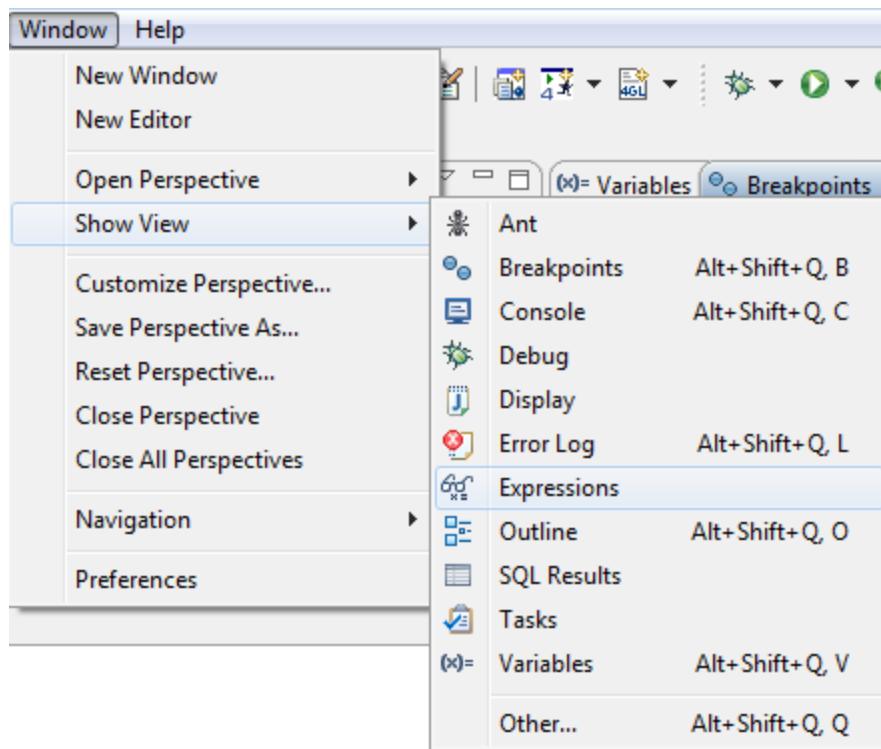


- 2) Relaunch the program in debug mode. Note that the program will suspend at the second breakpoint (line 14), though the first breakpoint will not be disabled. By default, variable 'a' has a value of 1, so the condition for the first breakpoint is not met and the breakpoint is skipped.



Expressions View and Watch Expressions

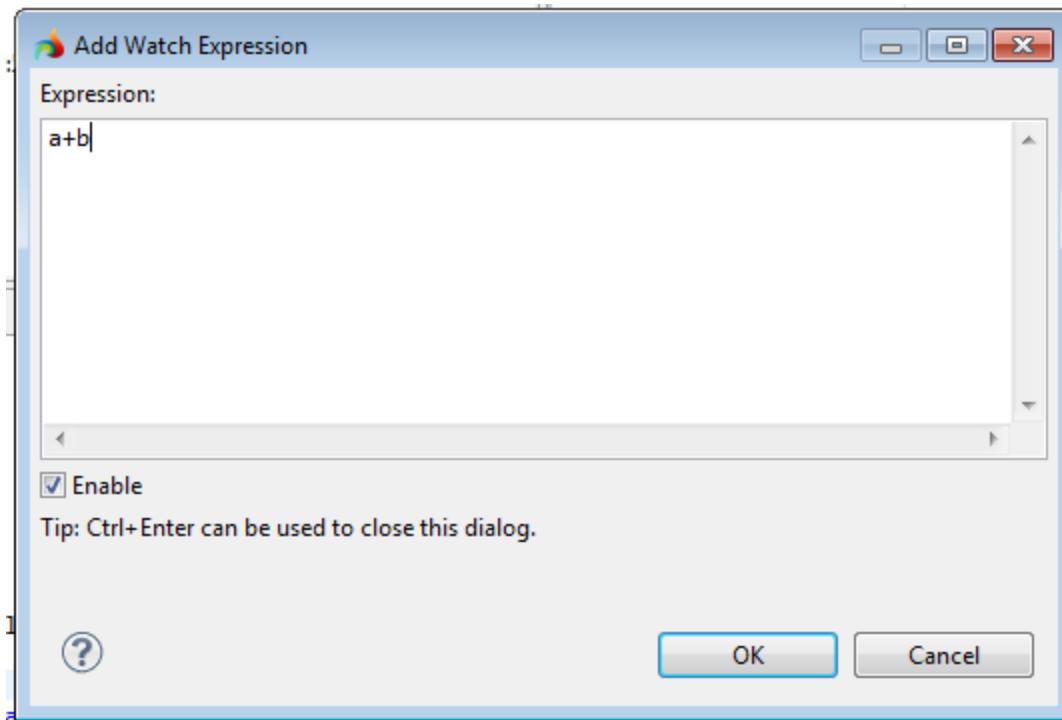
The 'Expressions' view is, by default, not included in the 'Debug' perspective. To open it, select from the menu **Window > Show View > Expressions**:



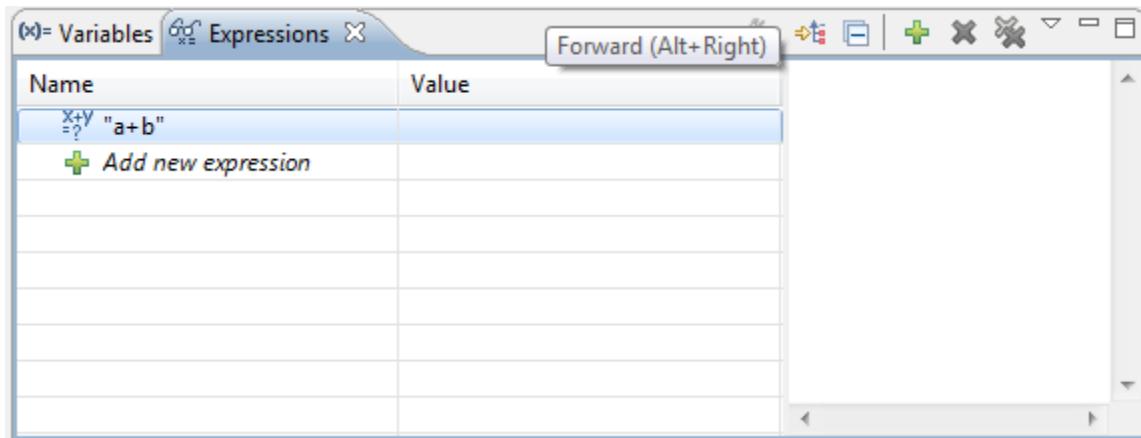
It will be opened in the same panel as the 'Breakpoints' view and the 'Variables' view.

An expression is a snippet of code which can be evaluated to produce a value. A watch expression is an expression that is repeatedly evaluated as the program executes. To create a watch expression, use the following steps:

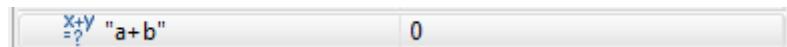
- 1) Open the 'Expressions' view and press the button on the toolbar of the view.
- 2) Right click in the Expressions view, and in the pop-up window, enter the following expression: 'a+b'. You will see how this expression will be evaluated throughout the program execution. Press **OK**.



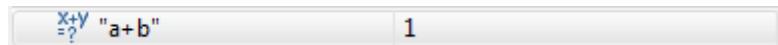
- 3) This expression is displayed in 'Expressions' view; if the program is not running, it has no value:



- 4) Run the program in debug mode.
- 5) Before the breakpoint at line 12 (LET a=1) the expression is equal to zero, because no values are yet assigned to the two variables used in the expression.

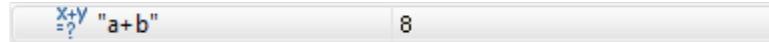


- 6) As we step into the first breakpoint, the value '1' is assigned to variable 'a'. As variable 'b' has no value yet, the whole expression will be evaluated to 1:



A screenshot of a software interface showing the 'Expressions' view. The input field contains the expression `x+y` and the output field shows the value `1`.

- 7) Step through the code until input is required for variable 'b'. Enter '7' when prompted to do so.
- 8) As the first function returns value for variable 'b', the expression is evaluated again:



A screenshot of the same software interface, showing the expression `x+y` and the value `8`. This indicates that the value of variable 'b' has been updated to 7, and the expression is now evaluated as 8.

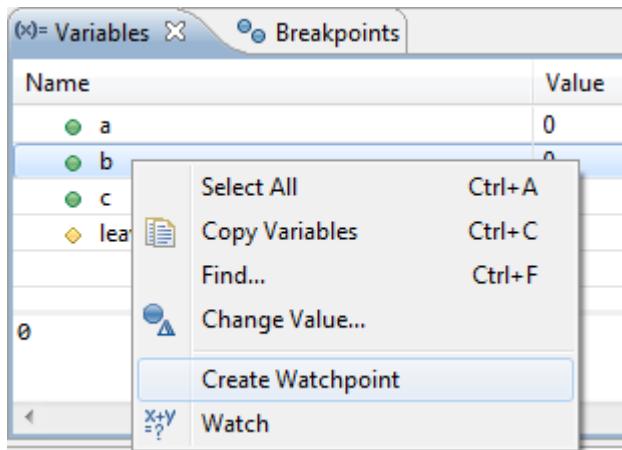
The other buttons of the 'Expressions' view have the same functions as described in the 'Variables' view.



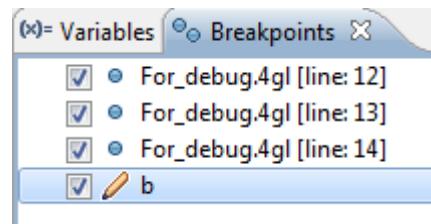
Watchpoints

Watchpoints are similar to breakpoints. A watchpoint is attached to a variable or an expression which is evaluated continually as the program is executed. Program execution is suspended when the value of the evaluated expression or variable changes.

- 1) To create a watchpoint on a variable, right-click variable 'b' in the 'Variables' view and select 'Create Watchpoint':



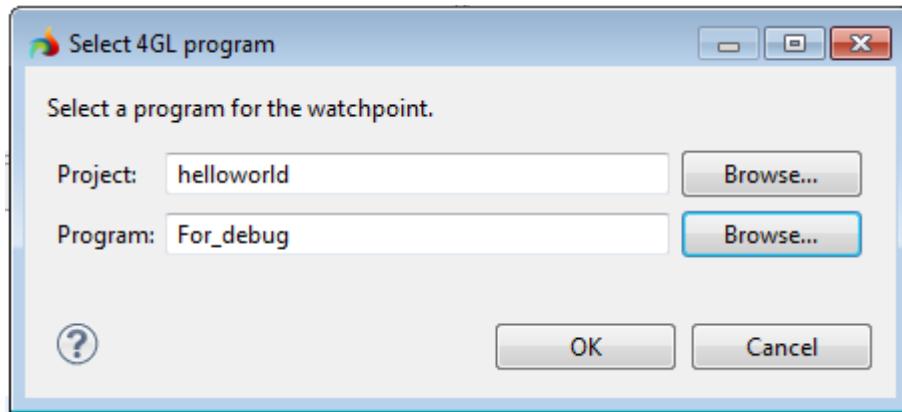
- 2) The watchpoint will be created in the 'Breakpoints' view:



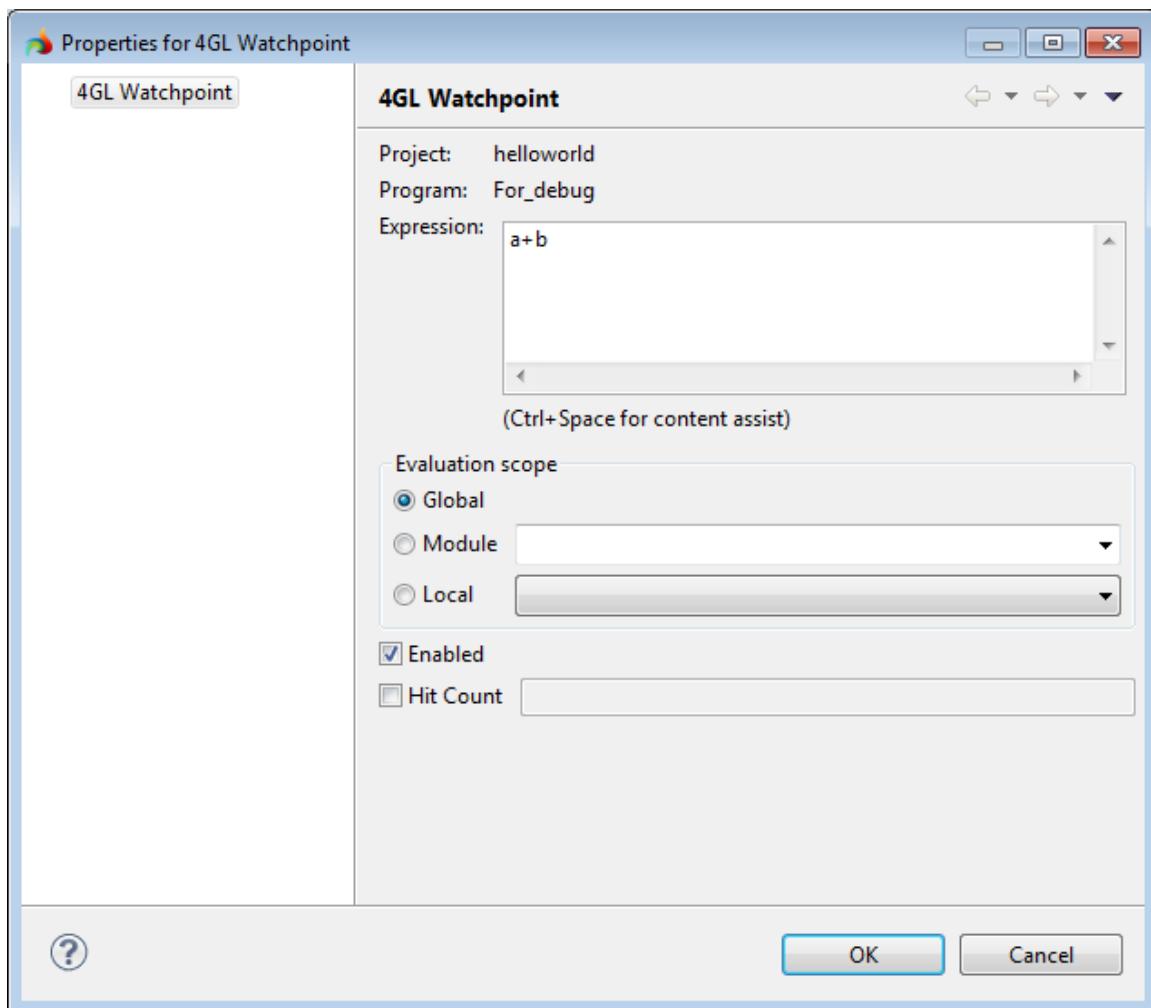
Note: It is also possible to create a watchpoint by pressing the button on the 'Breakpoints' view toolbar.



- 3) In the pop-up window, select the current project and program and press **OK**:



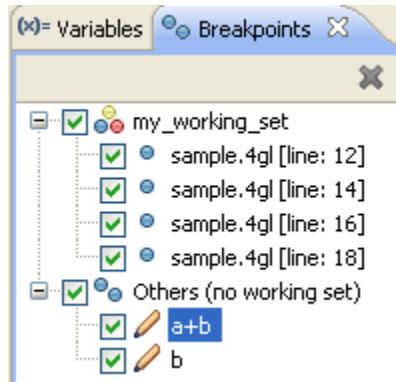
- 4) We have already created one watchpoint on a variable. The second watchpoint will be created on an expression. Enter: 'a+b' to the 'Expression' field and press **OK**.



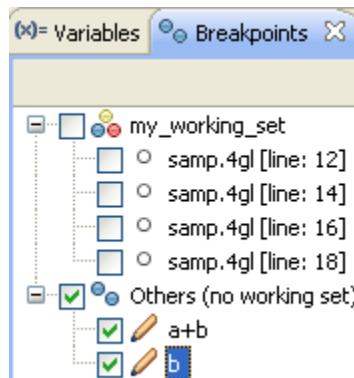


Note: Watchpoints on a variable can be also created this way; the name of the variable should be typed in 'Expression' field.

- 5) Now there are two watchpoints in the 'Breakpoints' view which are not included into the working set:



- 6) Disable all the breakpoints in the working set:



- 7) Relaunch the program in debug mode. The program will not be suspended at breakpoints because they are disabled, but it will be suspended after line 14 because value '1' is assigned to variable 'a' and the value of the expression 'a+b' is changed from 0 to 1. You can observe this change in 'Expressions' view.

Watchpoints are disabled in the same way as breakpoints. If you disable the first watchpoint and relaunch the program, the execution will be suspended after line 34 because the value for variable 'b' has been entered.



Configuration settings

Environment settings are specific configurations used by Lycia to compile and execute 4GL programs, and since everybody's 4GL program is different, Lycia allows these settings to be freely customised.

Lycia has several files (or file analogues) which contain the environment variables:

File name	Location	Purpose	OS
inet.env	This file can be opened from the Window -> Preferences -> 4GL -> Run/Debug-> GUI Servers preferences page. You need to select the server and press the Edit Environment button, the file will be opened in the editor area of the Studio.	It contains variables used when running a program in GUI mode (using LyciaDesktop, LyciaWeb, etc.). Any modifications will take effect only after the application is restarted.	Windows UNIX/Linux
env.properties	The LYCIA_DB_DRIVER stored in this variable can be changed using Window -> Preferences -> 4GL -> Database preferences page. It is located at \$LYCIA_DIR/LyciaStudio/etc	It contains environment variables used during compilation using LyciaStudio and for running a program in character mode. It is not advisable to change this file manually. Any modifications will take effect only after the Studio is restarted.	Windows UNIX/Linux
environ.bat	It is located at \$LYCIA_DIR/bin	It contains the environment variables used when the Lycia Command Line Environment in Windows is launched.	Windows
-	Run the following command within a terminal/shell session or window from the Querix directory: . environ ["dot" "space" "environ"]	It contains the environment variables used for working in the Lycia Command Line Environment in Unix/Linux.	UNIX/Linux



On Windows the environment settings for the command line environment can be also set manually using the system methods in Lycia Command Line Environment. For example, the following line sets the LYCIA_DB_DRIVER variable to <db2> value:

```
set LYCIA_DB_DRIVER=db2
```

This method specified temporary values for the variables, as after the Lycia Command Line Environment is relaunched, the variables are loaded from the environ.bat file. Thus, if you want to change the values of the variables on continuing basis, edit the environ.bat file.

On UNIX/Linux you should use the "export" command instead:

```
export LYCIA_DB_DRIVER=db2
```

To see the environment settings currently active use the "set" command without arguments both for Windows and for UNIX/Linux.

Environment variables

Lycia can have the following environmental variables:

Variable	V4 equivalent	Description
LYCIA_DB_DRIVER	HYDRA_DB_DRIVER	The database driver that will be used in compilation. It can acquire the following values: <oracle>, <informix>, <informixqx>, <sserver>, <db2>, <odbc>
LYCIA_DIR	HYDRA_DIR	Location of the Lycia installation. This is set by the installer.
LYCIA_PATH		Comma separated list of paths to search for p-code modules. It is optional and should be used when a program uses p-code libraries
MSGPATH	MSGPATH	The message path (e.g. C:\Program Files\Querix\Lycia\msg)
LYCIA_DRIVER_PATH	HYDRA_DRIVER_PATH	The driverpath (e.g. C:\Program Files\Querix\Lycia\lib)
LYCIA_GUI	HYDRA_GUI	Sets the Application Server to be in either GUI or text mode.



LYCIA_REPORT	HYDRA_REPORT	Configures the Report Writer output
PATH		The variable that duplicates the Windows or UNIX/Linux system path lists to ensure the correct work.
LICENSE_KEEPER_CONFIG		Location of the license_keeper.cfg file which contains the license keeper configurations. The name of the file should be included into the path
FGLPROFILE	FGLPROFILE	The location of the fglprofile.std file. The name of the file should be included into the path

Environment Settings - fglprofile

The fglprofile is an environment configuration file. It is a text file that provides information on the default values for a variety of settings. The fglprofile file sets global defaults for every Lycia application; if you want to tailor your environment for individual applications then use the appropriate form definitions, GUI script options or relevant functions. Any fglprofile setting will then be overridden for that particular application.

The following is a list of the fglprofile settings explaining their functionality.

Environment variables

```
system.environment.<name> = value
```

This fglprofile setting allows you to specify environment variables in the fglprofile for use globally. Here, <name> can be any valid environment variable.

Toolbar

The toolbar settings are used to make alterations to the default scripted toolbar toolbar. As soon as the dynamic toolbar is specified, this becomes irrelevant.

```
gui.toolbar.visible
```

This dictates whether the toolbar is present on the screen or not. This is True by default.

```
gui.toolbar.showtexts
```

Set this to False to remove the accompanying labels for toolbar buttons. This is True by default.



```
gui.toolbar.startindex  
gui.toolbar.endindex
```

These will specify the start and end points of reference for toolbar buttons.

Tab Controls

The following fglprofile settings will affect behavior when using tabbed windows:

```
gui.foldertab.input.sendnextfield = TRUE
```

Use this setting to make the focus shift to the first field in a selected tab. This behaviour will work more reliably if the field order logic is disabled, i.e., moving from one field to another will perform only the before and after logic for the current and destination fields respectively.

```
gui.key.foldertab.<number>.selection = <key name>
```

Use this setting to select a keypress to be sent to the server when the given folder tab is selected. The number value is the tab number, as specified when defining the tab, and the key name value is the actual keypress.

Default Keys

The fglprofile can change the key used as the default Interrupt key or as the default Accept key. The following syntax allows you to change these keys:

```
gui.key.<action> = <key name>
```

Here action is either "accept" or "interrupt" keyword. The key name is the name of a key which you want to use by default as the Interrupt or Accept key. It can be any key, for example, we can set the "delete" key as the Interrupt key:

```
gui.key.interrupt = delete
```

However, if you want to specify END or HOME as the interrupt key, you need to put the "key" keyword before the key name, e.g.:

```
gui.key.interrupt = key_home
```

Grid Settings

Where a data grid exists, with a sizeable amount of screen records, the transmission time between the grid and client can be detrimentally high because the server will send all records to the grid by default. The following grid settings allow you to specify the transmission rate meaning only the required data will be sent, as opposed to all.

```
gui.input.grid.data_mode = <paged/full>
```



This fglprofile option dictates whether data sent to a grid is sent in its entirety or whether it is sent a predetermined number of records at a time.

```
gui.input.grid.page_size = <integer>
```

Used in conjunction with the data mode option, this sets the number of records sent at a time when in paged mode.

```
gui.input.grid.lookahead = <integer>
```

When the previous two options are in use, this option will determine when the next page of data is sent to the grid. It does this by checking how far the current record is from the end of the page. The integer entered here sets the number of records from the end the current record needs to be before a new page is sent.

Miscellaneous

The following fglprofile settings do not fit into a specific category.

```
dialog.fieldorder = true
```

The field order logic refers to the before/after field logic that applies when navigating from one control to another. In simplistic terms this logic dictates an application event (e.g., a message prompt) that occurs before and after a user traverses from one field to another.

Therefore, where five fields exist, e.g. f1-f5, and where field f1 is active, clicking f5 will execute the after field logic for f1 and then the before and after logic for f2, f3, f4, and then the before logic for f5. If this setting is set to False, the application will only execute the after field logic for f1 and the before logic for f5.

The default behaviour for this setting is True, which will apply all field order logic for all fields.

```
gui.input.no_clip_space = false
```

When entering data into a field and sending to the client, the above will remove the behaviour that removes whitespace.

```
gui.combo.min_width = 3
```

This setting will specify the minimum width, in terms of characters, that a field can be before it is considered for conversion to a Combo Box. This is because a Combo Box field has to have an arrow appear in the field as well as the values and with very small words or numbers; the appearance of the field can be obscured. The above example will change the minimum width for a field to 3 before it becomes a Combo Box field. The default value is 2.

```
gui.combo.max_values = 50
```



There is a limit on the number of items allowed in any one field. As well as specifying the values directly in the INCLUDE statement; it is also possible to specify a range. This range of values could have been any size before GUI development, with values entered being checked to see whether they were within the range specified. With Combo Boxes however, all the values in the range are sent to the application when the Combo Box is drawn. If this range were too high, it would take a long time to draw the Combo Box and would be very inefficient. The default setting is 100.

```
gui.lazy_cache = false
```

With this fglprofile value set as TRUE, the client resource when loading LyciaDesktop will check the timestamp of the main script file as opposed to all the resource files. If this main script file is deemed to be the latest copy, all other resource files are assumed unchanged and the application loads in accordance with these files. Where the script file is not deemed to be the latest version, or this fglprofile setting is FALSE, the application will continue to check the timestamp of all resource files before loading the latest.

```
gui.key.<n>.translate = source_key destination_key
```

This setting performs an arbitrary translation of keypresses in the client. This does not affect character mode.

SSL Settings

The fglprofile file can be used to specify certain SSL settings for Lycia encryption. The following settings can be set:

```
gui.ssl.certificate = cert.pem
gui.ssl.privatekey = privatekeyenc.pem
gui.ssl.password =
```

These settings specify the SSL certificate, the private key to use and the password for the private key respectively.

```
gui.ssl.timeout = 300
```

The above determines the timeout on SSL connections in seconds. This is the limit on time that the connection will stay open without receiving a response from the server before closing the connection.

```
gui.ssl.enabled = false
gui.ssl.enforced = true
```

The two settings above are used to decide whether SSL is used in the first place and then, if it is, whether it is enforced. If SSL is used, it is strongly recommended that it be enforced.



System

These fglprofile settings will alter system memory setting:

```
system.memory.stack = 256
```

The default size of the 4GL program stack.

```
system.memory.stringpool = 32767
```

The program string pool size in bytes.

GUI

These fglprofile settings are used to make alterations to the Graphical User Interface:

```
gui.layout.break.max_spaces
```

This is the maximum number of spaces between two strings before they are declared separate strings.

```
gui.layout.break.start  
gui.layout.break.end
```

The above settings will specify characters that declare the beginning and the end of a string, respectively.

```
system.report.newline = native
```

The above fglprofile setting will tell the program which character is used when pressing the enter key for a new line in reports. This character differs depending on the system you are using, hence the need for a setting to tell the program which character means a new line. This setting has the system that the application is running on assigned to it, which will then tell determine the corresponding return character when interpreting documents. The choices are native, unix, windows, mac.

```
system.report.output.printer
```

This setting is Windows only – it specifies the printer to use when printing and has to be a valid Windows network device name.

```
system.report.output.screen = notepad
```

Again, this option is a Windows only option and works in character mode only. It declares the program to output data to.

```
system.report.output.pipe.fork = true
```



Sets whether the process is forked when reporting to a pipe.

Shadow options for compiler switches

```
gui.input maxlen = true
```

This dictates whether the maximum input length is the field length (FALSE) or the variable length (TRUE).

```
compat.ifx.input.validation = true
compat.ifx.input.numeric_entry = true
compat.ifx.input.display = true
compat.ifx.input.display_to = true
compat.ifx.form = true
compat.ifx.form.formline = true
compat.ifx.form.multiline = true
compat.ifx.input.auto_delete = true
compat.ifx.input.display.nobind = true
compat.ifx.report.enforce_columns = true
compat.ifx.report.enforce_right_margin = true
```

Most of these are confirmed with Informix compatibility, and therefore it is recommended that they be kept on.



Localization Settings

Localization allows you to create programs in Lycia that use different character sets and follow specific language rules.

All the data in the compiled programs are stored in Unicode. The source files can be saved with different encoding but during the compilation the pre-processor converts the source files into Unicode, thus achieving the unified encoding for the whole program. This allows you to have source files encoded using different character sets in the same program. The GUI clients always receive and send the data in Unicode. The locale differences manifest only when the connection to a database is established.

Each component included to the 4GL process has to be set up properly, so that the character conversions could be performed properly from the very beginning to the end of the chain. The 4GL process starts on the end-user workstation with front-end windows and ends at the database storage files.

The main points of the 4GL process character conversions are as follows:

- Each source file is compiled with the code set that is specified for it, at runtime all encoding is converted to Unicode.
- Runtime system always runs in Unicode, thus the encoding of the source files and the system locale are of no importance at runtime while there is no connection to a database.
- Upon connecting to a database, the data from the Unicode is converted to the database client locale.
- Database server locale settings can differ from database client locale.

It is important to remember that database servers usually don't detect that the locale of the database client is not set up properly. That is why no error message is displayed, but the character strings are passed incorrectly. This happens due to the fact that in different code sets one and the same character is represented by different codes, and one and the same code represents different symbols in different code sets.

For example, in CP1252 code set the Latin letter é is represented as 0xE9/233 and as 0x82/130 in CP437. The 233 and 130 are valid codes in both code sets. So, if the database locale is CP1252, 233 will correspond to é and 130 - to a curved quote. If the client application locale is CP437, the é will be encoded as 130 and stored as a curved quote in the database. When the symbol is retrieved from the database, it will be displayed again as é. Therefore, for the front-end user, it will be impossible to see that the data are stored incorrectly in the database.

At Compilation

All the source files are converted into Unicode before the compilation. The pre-processor needs to define the incoming file encoding to convert it successfully into Unicode. It is done in accordance with the following rules:



- The pre-processor defines the encoding of the incoming file in accordance with the `-e` flag passed to the compiler.
- If the `-e` flag was not passed to the compiler, the pre-processor converts its contents into Unicode treating the locale specified in the `CLIENT_LOCALE` environment variable as the incoming file encoding.
- If neither the flag nor the `CLIENT_LOCALE` variable is set, the compiler uses the default system locale.

File Encoding

If a file is saved in a non-default encoding, the encoding should be passed to the compiler during the compilation. This flag is valid for `qfql`, `qmsg`, and `qform` commands and its value should correspond to the encoding name. For more details see [qfql](#), [qmsg](#) and [qform](#) sections of this manual. E.g.:

```
qfql -e UTF-8 ...
```

If the file is compiled in LyciaStudio, this flag is passed to the compiler automatically and for each individual file it contains the encoding of this file set in file properties. If the encoding was not set explicitly, it is inherited from the container which can be the folder, program or project. The default encoding of all projects is CP1252. To change the encoding of a file in LyciaStudio:

1. Right-click the file and select **Properties** from the context menu.
2. In the Resource tab, the Text File Encoding section is used to specify the file encoding.
By default the encoding is inherited from the project.
3. Select **Other** and choose the encoding from the list or enter it manually.

If you want to change the encoding of a set of files, you can change the encoding of a folder or a project and all the newly created and already existing files will inherit this encoding.

CLIENT_LOCALE

This environment variable is typically set for the Informix database client. The name of the locale should follow certain rules and may be different from the encoding name which should be passed using the `-e` flag. The rules for the locale name are as follows:

```
CLIENT_LOCALE = Language_territory.code_set
```

For example, the French locale for Canada will look as follows:

```
CLIENT_LOCALE = fr_ca.8859-1
```

In this case, if you wanted to pass this locale using the `-e` flag instead of the `CLIENT_LOCALE` variable, the flag value would be `ISO-8859-1`. For more information about the `CLIENT_LOCALE` environment variable see Informix documentation.



At Runtime

While no data exchange with the database is performed, the application runs in the Unicode. At runtime, the data sent by the application to the database client are converted once more from the Unicode to the corresponding locale encoding. The conversion happens in accordance with the following rules:

- If the CLIENT_LOCALE environment variable is set, its value is treated as the target locale into which the data are converted.
- If the CLIENT_LOCALE variable is not set, the -e flag value used during the compilation of the program module containing MAIN block will be used.
- If neither of the above is set, the data are converted into the default system locale encoding.

After the program is converted, it communicates with the database driver using the database client encoding. The database driver defines whether the database client encoding differs from the database locale and if it does, performs the conversion, if possible.

The database locale is defined by the environment variable DB_LOCALE. The database driver sends the following values to the database from the client locale:

- CLIENT_LOCALE value. If it is not set - the value of the default system locale is sent instead.
- DB_LOCALE set on the client side. If it is not set on the client side, it is not sent, no values will be sent and the DB_LOCALE set on the server side will be used instead.
- Customized formats: GL_DATE, GL_DATETIME, DBDATE, DBTIME, DBMONEY, DBFORMAT. If they are not set, no values will be sent.

The above listed values are used to verify the database locale and check whether the client and database locale coincide. If the client locale differs from the database locale, the locales must be compatible, otherwise a conversion error will occur. Querix tools do not perform the conversion and cannot influence the database locale, so you must make sure that the conversion between the client locale and database locale is possible. For more information refer to the Informix database documentation.

Identifiers

The rule for naming 4GL identifiers normally defines that an identifier must start from a letter or an underscore and contain only Roman letters, numbers and special printable symbols. However, in non default locales and in source files saved in a non default encoding it is possible to use the 4GL identifiers containing non-Roman characters. An identifier may begin with any letter of the encoding the source file is saved in or an underscore, but it still should not begin with a number or a special symbol.



SQL identifiers can also contain non-Roman symbols depending on the file character set. But you must ensure that your database locale is compatible with the used encoding, otherwise a runtime error will occur.

Numeric and Currency Settings

Though the runtime always runs in Unicode, the numeric and currency settings. The runtime system needs DBMONEY or DBFORMAT to convert decimal values to and from strings. These variables store the settings that specify decimal and hundreds separators as well as currency symbols used for MONEY data types.

If these variables are set on the client side, they will be used for displaying the data and for recording the data to the database. If they are not set on the client side, the data will be displayed and recorded to the database in accordance with the CLIENT_LOCALE value, and if it is not set - according to the system locale settings of the client machine.

Date and Time Settings

The runtime system uses the DBDATE and DBTIME environment variables to perform to and from string conversions of date and time values. If they are not set, GL_DATE and GL_DATETIME variables are taken into consideration, if they are set on the client side. If these variables are not set, the date format of the CLIENT_LOCALE variable will be used. If none of the above is set - the system locale settings are used.

The system uses English words to convert the dates entered as shortened months and days names when ddd/mmm markers are used (this format of date input and display can be specified by the FORMAT field attribute or the USING operator). I.e., the day (ddd) and month (mmm) abbreviations will always be displayed in English, irrespectively of the current locale settings.

Multi-Byte Values

Some of the character data types require their length to be predefined during the declaration, e.g. CHAR(n) or VARCHAR(n). You must remember that in multi-byte locales 'n' stands for the number of bytes rather than for the number of characters like it does in the default English U.S. locale. Thus, when declaring variables in multi-byte locales you must remember that each character will most probably take up more than 1 byte.

For example, Cyrillic letters take up 2 bytes each and a variable declared as CHAR(10) will be able to contain only 5 Cyrillic letters, whereas in the default English U.S. locale it would be able to contain 10 letters.

Other built-in functions and SQL operators also calculate the length of variables by bytes rather than by letters in multi-byte environment. So the length() function will return 10 for a variable declared as CHAR(10), even though in a Cyrillic locale it can accommodate only 5 letters and in Chinese locale characters can take up to 4 bytes each. If a program attempts to write more bytes than can fit into such a variable, the excess bytes will be discarded.

However, the byte or character semantics may differ depending on the database, e.g. if variables are declared LIKE database columns. Different database engines use different length semantics



and definitions for CHAR, VARCHAR, NCHAR and NVARCHAR variables. You should take it into account when developing an application.

Some engines use character length semantics, some - byte length, the others can use both. For example, SQL Server uses byte length for the size and supports non-UCS-2 character sets in CHAR, VARCHAR and TEXT columns, and USC-2 characters and char length semantics are applied for NCHAR, NVARCHAR, NTEXT columns.

The list of database engines and their character data type length semantics with comments is given below:

Database Engine	Comments	Length Semantics
Genero DB	You can set CHARACTER_SET=UTF-8 configuration to store data in UTF-8	BLS
IBM DB2	The character set is specified by the CODSET of CREATE DATABASE	BLS
Informix	The character set is specified in DB_LOCALE environment variable	BLS
Microsoft SQL server	Bytes semantics is used for CHAR/VARCHAR columns. The character set used for data storage is specified by database collation. NCHAR/VCHAR sizes are stored in characters; UCS-2 is used to store the data.	BLS/CLS
MySQL	Configuration parameter specifies the server character set used to store data	CLS
Oracle	The usage of Byte or Character Length semantics can be specified for the whole database or on column level. Database character set is applied for CHAR/VARCHAR columns; NCHAR/NVARCAHR sizes are specified in local character set.	BLS/CLS
PostgreSQL	The database character set is specified by WITH ENCODING of CREATE DATABASE	CLS
Sybase Adaptive Server Enterprise (ASA)	CHAR/VARCHAR column sizes are specified in bytes by default, but this can be changed by adding the CHAR[ACTER] as the part of the length specification; the data are stored using the db character set	BLS/CLS
	NCHAR/NVARCAHR columns sizes are specified in characters; UTF-8 character set is used to store the data	
Sybase Adaptive Server Enterprise	CHAR/VARCHAR columns sizes are specified in bytes;	BLS/CLS



(ASE) db character set is used to store the data.

NCHAR/NVARCHAR columns sizes are specified in characters; db character set is used to store the data.

UNICHAR/UNIVARCHAR columns sizes are specified in characters; UTF-16 code set is used to store data

SQL statements and functions are processed using the byte length semantic. For example, the LENGTH() function in Informix returns an integer identifying the value length in bytes, and in Oracle one needs to use LENGTHB() function to get the bytes value, because the LENGTH() function returns the length in characters.

It is advised to use byte length semantic based character data types for data bases. This can be important in cases, when the application uses DEFINE LIKE statement which is based on the database schema.

Complex Substring Expressions

When byte length semantics is used, all the character positions correspond to the byte positions. In a multi-byte environment, byte and character positions need special attention; otherwise, invalid data will be returned and processed. For example, you should specify only the first byte of a multi-bit character in a substring operator [x, y], otherwise errors may occur.



Using C API in Lycia

The C API in Lycia allows a developer to call C functions in their 4GL code. C code cannot be embedded in Lycia 4GL, to use it you must put it into a dynamic C library and then link this library to your program. Calling C functions from Lycia programs differs a bit from calling them using earlier compiler versions.

To be able to use a C function, the following requirements must be met:

1. The C source should contain a global variable usrcfuncs, declared using the cfunc_t struct defined in fgifunc.h. This is used to declare which C functions should be callable from the 4GL.
2. All C source must be compiled into a dynamic library and linked against lib4glc in the Lycia distribution.
3. The ready C library should be linked to the 4GL program which uses the C functions.

Creating C sources

Variable usrcfuncs in fgifunc.h should be declared as follows:

```
typedef struct {

    char *cf_name;          /* Name of function */

    int (*cf_ptr)();         /* Pointer to the function */

    short cf_nargs;          /* Ignore. RDS compatibility only */

} cfunc_t;
```

This file is located at %LYCIA_DIR\include directory. This directory is referenced by -I flag when a C source is compiled.

Now suppose we want the following C function to be callable from our 4GL program:

```
/* my_c_src.c */

#include "fglssys.h"

int my_c_func(int argc) {
    int n;
    popint(&n);
```



```
    pushint (n/2);  
  
    pushint (n*2);  
  
    return(2);  
  
}
```

All we have to do is make sure that our C module declares a global variable usrcfunc as follows:

```
#include "fgifunc.h"  
  
int my_c_func(int argc);  
  
cfunc_t usrcfuncs[] = {  
  
    { "my_c_func", my_c_func, 1 },  
  
    { 0, 0, 0 },  
  
};
```

Creating a dynamic C library

You can either use Lycia Studio to compile and link c sources, or command line.

If you create a dynamic library using Lycia Studio, go to **File -> New -> New Dynamic C library** main menu option. Then right-click on the library in the 4GL Project view and select **New -> New C source file** to create C sources. Then you can select the build option from the interface and the library will be built and linked against lib4glc automatically.

Using the command line

If you use the command line, C sources must be first by means of qxcc command line tool. This tool just invokes the C compiler installed on your system (in case of Windows - Visual Studio). Its syntax is as follows:

```
qxcc [ -32 | -64 ] [ -debug ] [ -pic ] [ -c ] [ -I<path> ] [ -L<path> ]  
[ -lname ] [ -o output_name ] input_files
```

Flag -32 or -64 indicated whether the source is building for 32 or 64 bit system. Flag -o references out_file.obj which is the name of the file which will be the result of the compilation, it may include the path. The input files are the names of the files you want to compile including the path if necessary. The -I flag points at the directory where C headers are located which is typically %LYCIA_DIR\include and where fgifunc.h file should be located. Flag -L<path> points to the directory where libraries are stored and must be followed by the library name.

Here is an example:



```
qxcc -o "C:\Users\me\workspace\C-lib\output\c2.obj" -I "C:\Program Files\Querix\Lycia/include" "C:\Users\me\workspace\C-lib\source\c2.c"
```

After everything is compiled, you will need to use qxld command to link the dynamic library. It has the following syntax:

```
qxld -dynamic -o library_name sources.obj -L$LYCIA_DIR/lib -l4glc
```

Here -dynamic flag instructs the linker to create a dynamic C library, -o flag contains the name of the library which it will have when built. The -L flag links the library to l4glc. Sources.obj are the compiled C source files which need to be included into the library. Here is an example:

```
qxld -dynamic -o mylib mysource1.obj mysource2.obj -L"%LYCIA_DIR%\lib" -l4glc
```

esqlc

The compiler for C files with embedded SQL code.

```
esqlc [-t] [-s] [-l] [-c] [-e] [-E] [-D] [-i] [-o] input_file
```

The following options apply to esqlc:

- **input_file**

One or more source C files to be compiled.

- **-t**

If set, enables the names transliteration.

- **-s suffix**

Sets the variable suffix to the 'suffix' argument used with the flag. For example the 'suffix' value can be _esql_var.

- **-l locale**

Sets the name for the output locale. The 'locale' value must be in the format 'language_territory'. For example, for a German locale it should be de_DE and for the Canadian locale using French language - fr_CA.

- **-c**

If set, the file is processed with a customized collation enabled. For example, if the argument following the flag is 'x=y', it signifies that there should be no difference between x and y and variables varx and vary should be treated equally.



- **-e encoding**

Sets the output encoding name. For example, the 'encoding' argument can have value cp850 to set Code Page 850 as the file encoding.

- **-E encoding**

Sets the input file encoding name. For example, the 'encoding' argument can have value iso88591 to set ISO-88591 as the file encoding.

- **-D arg**

Specifies preprocessor definitions. E.g. -D IsLycia=1

- **-i file**

Specifies the name of the input file.

- **-o file**

Specifies the name of the output file.

Calling a C function from 4GL

We can call my_c_func in the 4GL source like any other function:

```
# my_4gl_src.4gl

MAIN

DEFINE n1, n2 INTEGER

CALL my_c_func(10) RETURNING n1, n2

DISPLAY n1

DISPLAY n2

END MAIN
```

The 4GL source can be compiled using fglc, as per usual. The C source must be compiled into a dynamic library and linked against lib4glc in the Lycia distribution. Assuming our C source is compiled into a library with the base name my_c_lib, we can dynamically link it to the 4GL module as follows:

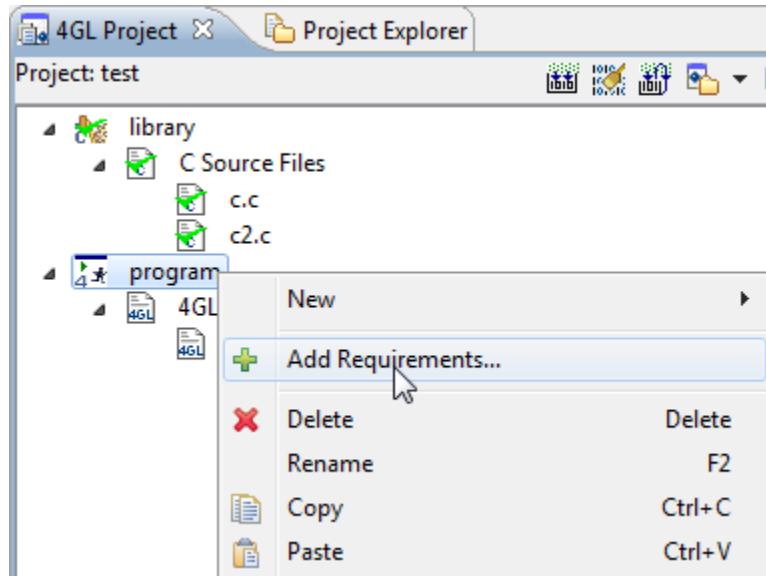


```
qlink my_4gl_src.4o -lib my_c_lib -o my_prog.4o
```

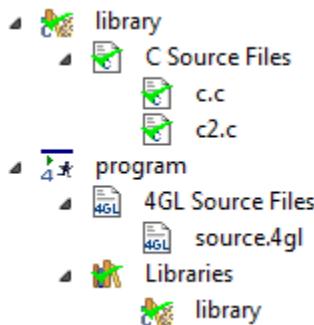
More information about [qlink](#) can be found in the corresponding section of this manual.

The same can be achieved, if you create a 4GL program in Lycia Studio. To link a dynamic C library located in the project to a 4GL program do the following:

1. Right-click the 4GL program in the 4GL Project view and select **Add Requirements...** option.



2. Select the library from the list and click **Finish**.
3. The library will be added to the program requirements and will be linked to the program, when the program is built.



If the dynamic library is not built, the command to build the 4GL program will cause the library to build also. This only works if the library is an internal one, i.e. it is located in the same project as the 4GL program. For more information about how to link internal and external libraries in LyciaStudio see [Using libraries](#) section.



Calling a 4GL function from C

It is also possible to call a 4GL function from within the C code. You just need to use the API defined in fglapi.h.

To load a PCode module into memory:

```
int fgl_start(const char *modname);
```

To call a function in the currently loaded PCode module:

```
int fgl_call(const char *funcname, int nargs);
```

To unload the currently loaded PCode module:

```
int fgl_end();
```

Here is an example of calling a 4GL function from a C program. Suppose we want the following 4GL function to be callable from our C program:

```
# my_4gl_src.4gl

FUNCTION my_4gl_func(n)
  DEFINE n INTEGER
  RETURN n / 2, n * 2
END FUNCTION
```

We can call this function from our C program as follows:

```
/* my_c_src.c */
#include "fglapi.h"
#include "fglsys.h"
#include <stdio.h>
int main() {
  int n1, n2;
  fgl_start("my_4gl_module.4o");
  pushint(10);
  fgl_call(my_4gl_func, 1)
```



```
popint(&n1);

popint(&n2);

fgl_end();

printf("%d\n", n1);

printf("%d\n", n2);

return(0);

}
```

The 4GL source can be compiled using fglc, or Lycia Studio interface, as per usual. When we compile the C source, the resultant executable will need to be linked against lib4glc in the Lycia distribution.



The Application Server

In order for clients to access a GUI compiled 4GL program, the server or local machine must have a GUI Server installed – this is the Application Server. This server uses specific TCP/IP port numbers so it can respond to connection requests from any of the Querix thin-clients that require access to 4GL applications.

Using the GUI Server

During the installation of Lycia, you are prompted to configure the connection details for the GUI Server, once configured the GUI Server runs as a background service called qxinetd under Windows or as a daemon under Unix.

Defining the GUI Server connection has several entry requirements:

- Service name – The alias for the connection socket
- Port number – TCP/IP port number that clients connect to
- Querix directory – Location of the Querix install directory
- Program directory – Location of your 4GL programs. Default directory: LYCIA_DIR\Progs
- Log directory – Location of the session transcripts (logs). Default directory: LYCIA_DIR\Logs
- Environment file – This file contains the environment settings specific to each GUI Server.
- Default database – This is the database driver that will be used for all connections to this port.

Your GUI Server connections can be managed from the LyciaStudio by clicking the “Properties” tab in the Project toolbar menu. Each GUI Server that has been configured on the system will be displayed, denoted by the server’s name.

Each GUI Server will have its own environment configuration; double clicking the environment tab allows you to modify the settings in the form of environment variables. The data is saved in the file \$LYCIA_DIR/etc/inet.env, so if during installation you have opted not to install LyciaStudio, these settings can still be edited.

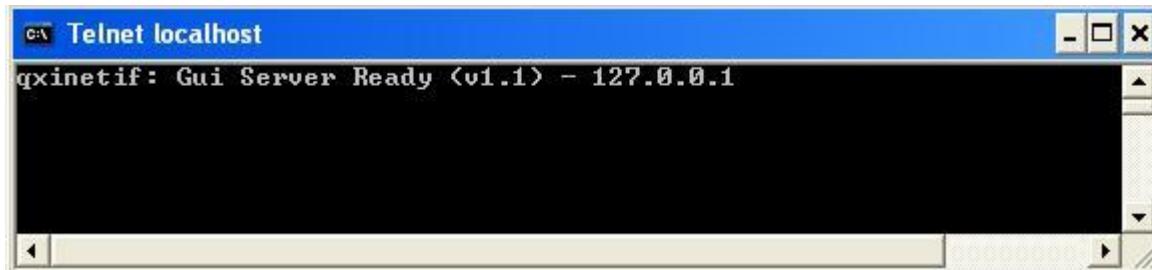


Testing the GUI Server Connection

Before any thin-clients try to connect to the GUI Server, it is worth while running a connection test. The application server will respond to telnet requests, so by entering the following at the command line (assuming you have a telnet client installed):

```
telnet localhost 1889
```

If the connection is successful, you will see the response:



This is the same for Unix/Linux users – perform the same actions as above in a command line to check that you can connect to the GUI application server service.

It is also possible to verify whether the GUI application service is running and whether you are able to connect to the Querix listener service.

UNIX/Linux

The easy way to do this is to use the following code at the command line:

```
/etc/init.d/qxinetd2 status
```

If the GUI server is running, the message 'The GUI Server is running' will be displayed.

Windows

To test whether the GUI application service is running in a Windows environment, simply open the Control Panel, then Administrative Tools, and finally Services. You should see in the list of services 'Querix Hydra GUI Server'. If the GUI Server is running, this will say 'Started' under the Status.

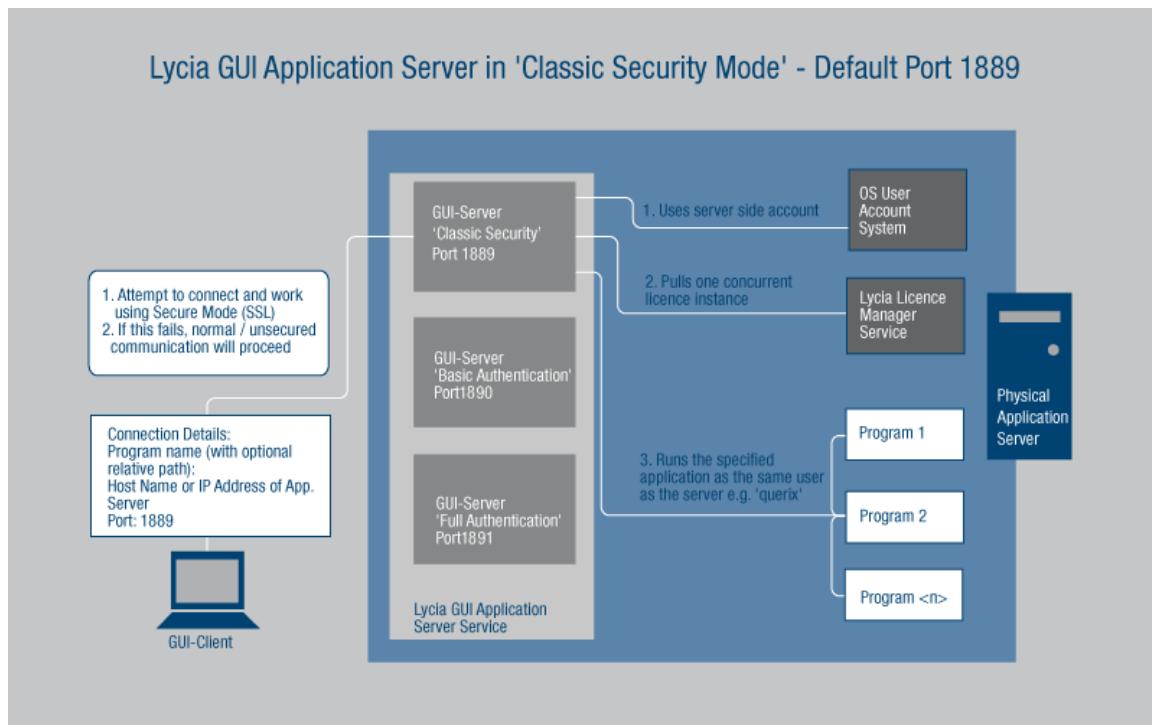
For more detailed information about the GUI server verification and about the troubleshooting, if you discover that your application service is malfunctioning see the Lycia Getting Started Guide.



GUI Server Security

There is the ability for a client to authenticate to a server, which allows us to execute the server process as the user that has actually connected to the server, rather than the traditional model where the server processes were executed within the same user context as the service itself. To allow us to achieve this, the new listener introduces two types of authentication: none (classic) and full.

- 'None' or the classic mode



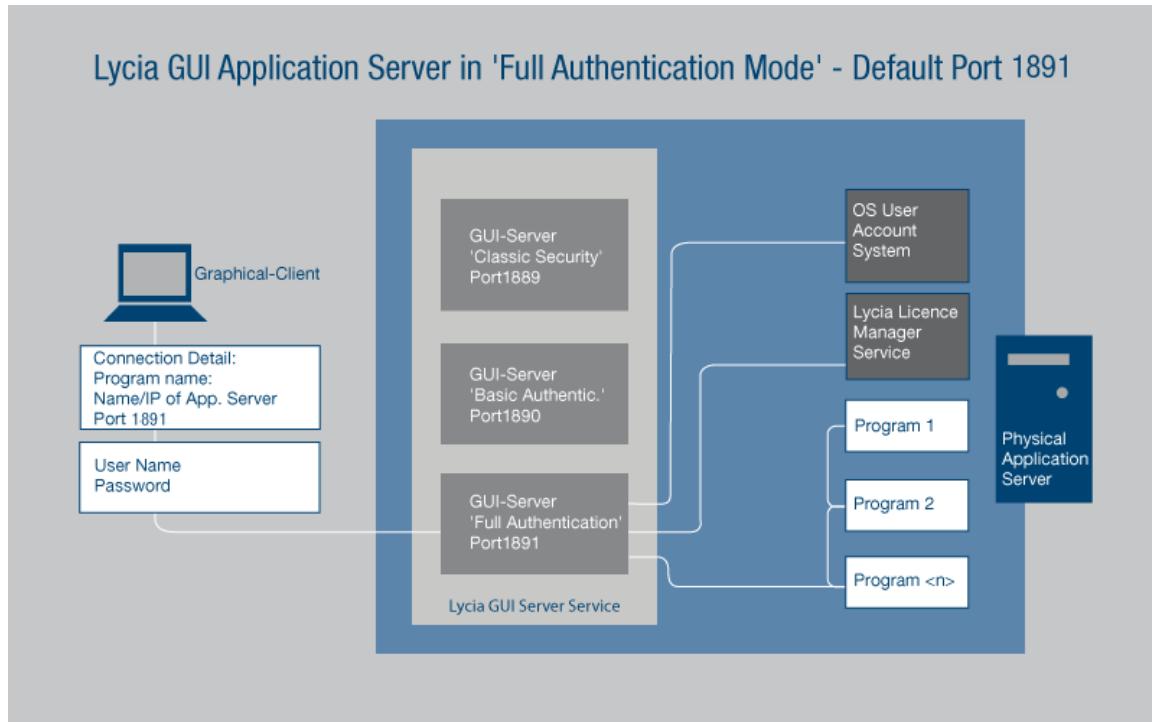
'Classic' Mode

In this mode, the listener operates in the same manner as the GUI application server (Legacy versions) with the server processes being launched as the system user that the service itself is running as. One difference between this mode and the traditional server is that rather than the client explicitly specifying if an SSL connection is to be used, the client will first attempt to connect to the server using SSL, dropping back to a plain text connection if the SSL connection fails.

Note: If the SSL connection is successful, a small 'lock' icon is displayed in the status bar of the client application window.



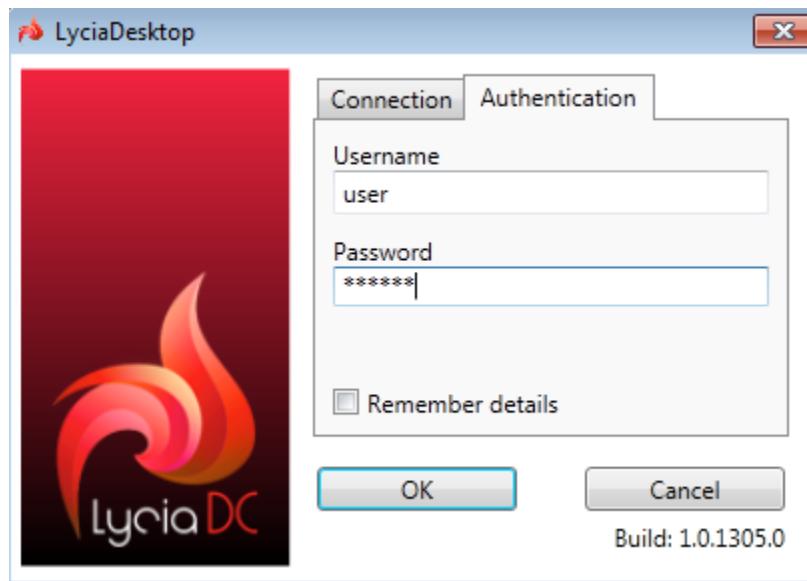
b. 'Full' authentication mode



'Full' Mode

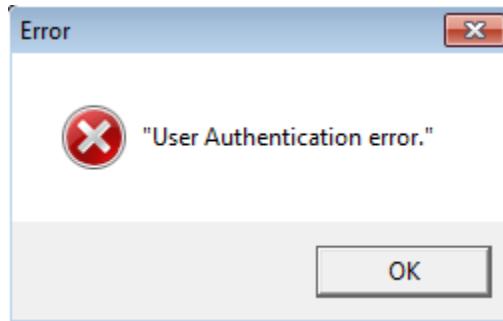
When a listener is operating in the full authentication mode, the client must specify both username and password.

Running the connection dialog box in LyciaDesktop and clicking on the 'Authentication Tab' allows you to specify the username and password.



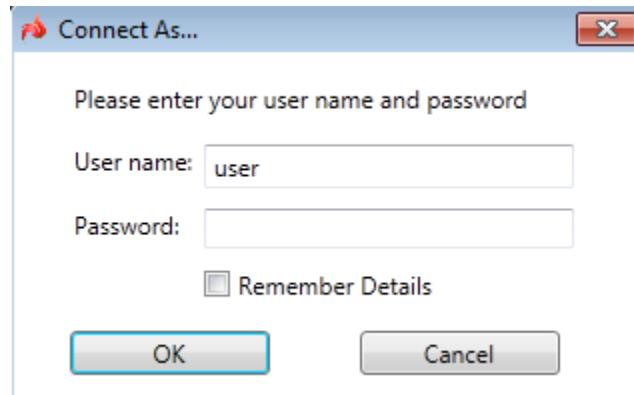
LyciaDesktop Authentication

This information is then sent to the server and validated. If no user with the specified password exists, an error message is displayed.



Authentication Error Dialog

If no username and password were specified in these fields (both empty), a dialog box appears and asks the user to enter the user name and password:



User name and Password Dialog

The server will then check that both the specified username and password are valid on the system before either returning a user authentication error (if the username and password do not match) or launching the process as the specified user (if a valid username and password are entered).

Note: This mode requires the server to run as the system account (either *root* in UNIX or the Local System account under Windows). The full authentication mode will only operate in full SSL mode meaning that if SSL is configured to be turned off or the SSL keys/certificates are missing or invalid, the full authentication mode will not function.

Lycia Encryption

Lycia uses the 'SSL – Asymmetric Key Cryptography' Encryption Model. After Lycia is installed, a demonstration purpose SSL certificate (file) and a private key (file) are installed in the application server directory. The GUI application server can also be configured to use other (third party) keys and certificates.

The following SSL related environment settings can be specified in the corresponding GUI application server environment:

QXGUSSL	Boolean	Turn automatic SSL on/off (will attempt to connect via SSL)
QXSSLENFORCE	Boolean	Force SSL – only works with SSL or not at all
QXSSLCERTNAME	String	Filename of the SSL certificate with path
QXSSLPRIVATEKEYNAME	String	Filename of the SSL private key with path



QXSSL_TIMEOUT	INT	Maximum Time in milliseconds the server tries to establish a SSL connection with the client.
---------------	-----	--

Example:

```
# SSL Configuration  
  
QXGUSSL=1  
  
#QXSSLENFORCE=1  
  
QXSSLCERTNAME=$QUERIXDIR/progs_ssl/cert.pem  
  
QXSSLPRIVATEKEYNAME=$LYCIA_DIR/progs_ssl/privatekeyenc.pem  
  
QXSSL_TIMEOUT=600
```

	<p>Note: Using HP-UX 11.11 on the PA/RISC architecture you may encounter difficulty in getting SSL handshakes to work correctly. This is due to the fact that the default installation of the HP-UX 11.11 platform does not have a dev/random or dev/urandom meaning that OpenSSL is unable to generate keys. To fix this problem, you will need to install the strong number generator patch from HP found at the following location:</p> <p>http://h20293.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=KRNG11I</p>
--	--



Authentication

Windows

On Microsoft Windows, the application server uses the standard Windows API, allowing users that are local to the server as well as domain users authenticate to the Lycia application. Once the user has been successfully authenticated, the application server will load the authenticated user's security context, registry hive and environment and launch the process as the authenticated user.

Unix/Linux

Because of the many different methods that are used for authentication under UNIX operating systems, Querix currently offers 3 authentication modules for UNIX operating systems; a crypt based authentication module (Crypt), an MD5/shadow based module (Shadow) and a PAM module. During the installation process you will be asked by the installer to select which module you wish to use on the server that you are installing to. The following descriptions may help you to decide which method is suitable:

Crypt

This is the simplest authentication method that is available for UNIX based operating systems. On a system where the crypt method of authentication is used, all user information including the password is stored in the /etc/passwd file. The passwords are encoded with the following one-way hash:

- user sets a password
- Password is encoded with randomly generated value called a salt
- The salt is then stored along with the encoded password in the /etc/passwd file

This method means that any particular password could be stored in 4096 possible ways. When a user logs into the system, the salt is retrieved from the /etc/passwd file. This salt is then used to encode the supplied password and this encoded password is compared with the stored encoded password in the /etc/passwd file; if there is a match then the user is authenticated.

Due to the fact that the /etc/passwd file must remain world readable, it means that anyone who has access to the system can have access to everyone's salt / password combination and using the list of salt values could perform a very simple dictionary attack (whereby a large list of standard passwords are encoded using the salt values gained from the passwd file, searching for matches with the encoded passwords in the passwd file) to gain access to other accounts including system / root accounts.

Shadow

To get around the problem of a user being able to gain access to the salt / password combination from /etc/passwd file the password shadowing method was introduced. This gets around the problem relocating the passwords to another file (usually /etc/shadow) that is only readable by the root user, replacing the salt / password combination with a character (this is usually 'x'). This



effectively removes the ability for a user to perform a dictionary based attack from the encoded passwords on the system.

	<p>To check the type of authentication currently in use:</p> <p>If you examine the contents of your /etc/passwd file, you will notice each line contains an entry for a different user, e.g.:</p> <pre>smith:fi3sED95ibqR6:100:100:John Smith:/home smith:/usr/bin/sh</pre> <p>The second parameter in the string is the encrypted password / salt, so we can determine that the above system is using the crypt method for authentication. If the system is using a shadowed method of authentication, then the entries will be of the form:</p> <pre>smith:x:100:100:John Smith:/home smith:/usr/bin/sh</pre> <p>Note that the encrypted password has not been stored in the /etc/passwd file, but replaced with the 'x' character.</p>
---	---

Pluggable Authentication Modules (PAM)

The third type of authentication method we offer for UNIX is a PAM based module. PAM allows a systems administrator to decide which of the underlying operating system authentication methods an application should use rather than this information being hard coded into the application itself. This is accomplished by means of a collection of libraries and APIs that the application can use to request authentication information. PAM also provides the ability to use many different methods of authentication (e.g. smart cards) by means of modules that 'plug into' the PAM library to perform the actual authentication and pass the results back to the PAM library.

The systems administrator can then configure separate applications to use different methods of authentication and PAM modules by means of the PAM configuration files. When a PAM application is launched, the PAM library will read the configuration for the application and act appropriately based on the information contained within the PAM configuration files. PAM also allows the stacking of authentication modules so that if one method of authentication fails for a user a secondary method can be attempted before failing.

There are two ways that PAM can normally be configured: using a single file located at /etc/pam.conf or using a collection of configuration files located within the /etc/pam.d directory. It is worth noting that if both /etc/pam.conf and a /etc/pam.d/sytle configuration are present on the system, PAM will use the /etc/pam.d/ rather than the /etc/pam.conf for its configuration.

The /etc/pam.conf file is made up of a list of rules, each placed on a single line. The format for each line is of the following format:

```
service type control module-path module-arguments
```



The syntax of files contained within the /etc/pam.d/ directory is identical apart from the absence of the 'service' field; in this case the service is the name of the file in the /etc/pam.d/ directory.

The service is the familiar name of the application (e.g. 'qxinetd2'). The service name *other* is generally reserved for giving default rules for services that aren't defined.

The type field defines the management group that the service belongs to and can be set to account (non-authentication based account management), auth (user authentication & group authentication), password (application can update user password) or session (this is used for operations that need to be either before or after a user has authenticated).

The control field configures what the PAM library should do if the user fails to authenticate and can be set to required (this will return failure after all methods have been completed on fail), requisite (like required, but if this module fails control is passed directly back to the application without processing any additional modules), sufficient (success of this module is enough to satisfy the authentication requirements of the stack of modules [note this is ignored if a *prior* module had the required flag]) or optional (the success or failure of this module is only important if it is the only module in the stack associated with the service & type).

The module-path is either the full path of the PAM-module that is to be used by the application, or a relative path from the default module location (normally /lib/security).

Finally, the module-arguments are a space delimited list of tokens that are passed to the PAM module as arguments.

Getting PAM to work with your applications

When the Lycia application server is running in PAM authenticating mode, it will identify itself with the service name of 'qxinetd2'. With this in mind, if you wanted to configure PAM so that the application server could use PAM, you could add the following lines to the /etc/pam.conf file:

```
qxinetd2    auth sufficient    libpam_krb5.so.1
qxinetd2    auth required      libpam_unix.1 try_first_pass
```

This states that the authentication details provided by the application server should first try the kerebos PAM module and if this succeeds then the user has been authenticated. If this fails, the system should use the same password and attempt to use the UNIX PAM module for authentication. If the UNIX module fails, then the PAM system should return authentication failure to the qxinetd2 service (the 4GL application server). If the system is configured to use the /etc/pam.d/ directory configuration rather than the /etc/pam.conf file, then the systems administrator should create a /etc/pam.d/qxinetd2 file containing the following:

```
auth sufficient    libpam_krb5.so.1
auth required      libpam_unix.1 try_first_pass
```

This will attempt to perform the steps in the /etc/pam.conf example stated above.



 A circular icon with a white 'i' inside, set against a light purple background.	<p>If you require any additional information regarding PAM configuration on your system you should consult your system documentation as detailed PAM setup is outside the scope of this document.</p>
---	---

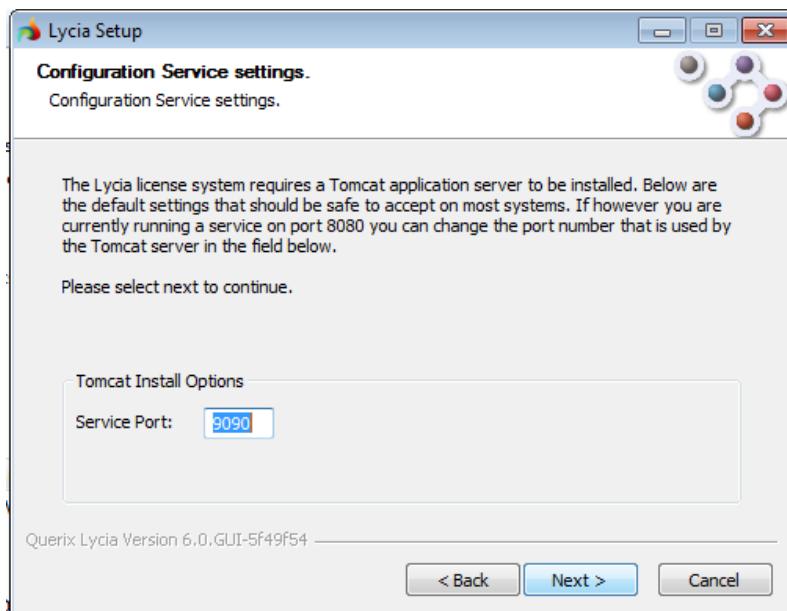


Web Application Server

Lycia installation package contains source files for Lycia Web Application Server installation. The Web Application Server is installed and set up, and run automatically and makes it possible to use Lycia Administration Tool, LyciaWeb and Web Services.

Web Application Server Installation

Lycia Web Application Server is included into the Lycia Development Suite installation package. The default Port is specified during the installation. You can change the port number, if necessary, or leave the default port 9090:



Web Application Server is based on Apache Tomcat Server, and these services can be run simultaneously without interfering with each other's operation, provided that they use different ports.

However, if you have Tomcat Server installed to your computer, Lycia Web Application Server will probably not launch automatically. Therefore, for the first time, you will have to launch it manually from the Services menu.

Changing the Default Settings

The web server configuration tool can be found in **Start-> All Programs -> Querix -> Web Server -> Querix Web Server Manager**. Launching it requires administrator permissions. This tool is a native Apache Tomcat tool for web server configuration, so for any documentation on it or regarding any issues found contact the vendor.



Using Web Application Server

As have been mentioned above, Web Application Server is used by different Lycia web applications, including LyciaBI Server, LyciaWeb and web services.

All the products that need Web Application Server can be accessed by two common ways: one can input their name, following the server address, to the address line of the browser, or one can login to the Web Server and go to the Applications list.

http://localhost:9090/manager/html				
		Start Stop Reload Undeploy		
/LyciaAjax	Lycia Ajax Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 120 minutes
/LyciaBI	LyciaBI	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/LyciaBIBirtReportEngine	Eclipse BIRT Report Viewer	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/LyciaBIJasperReportEngine	LyciaBIJasperReportEngine	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

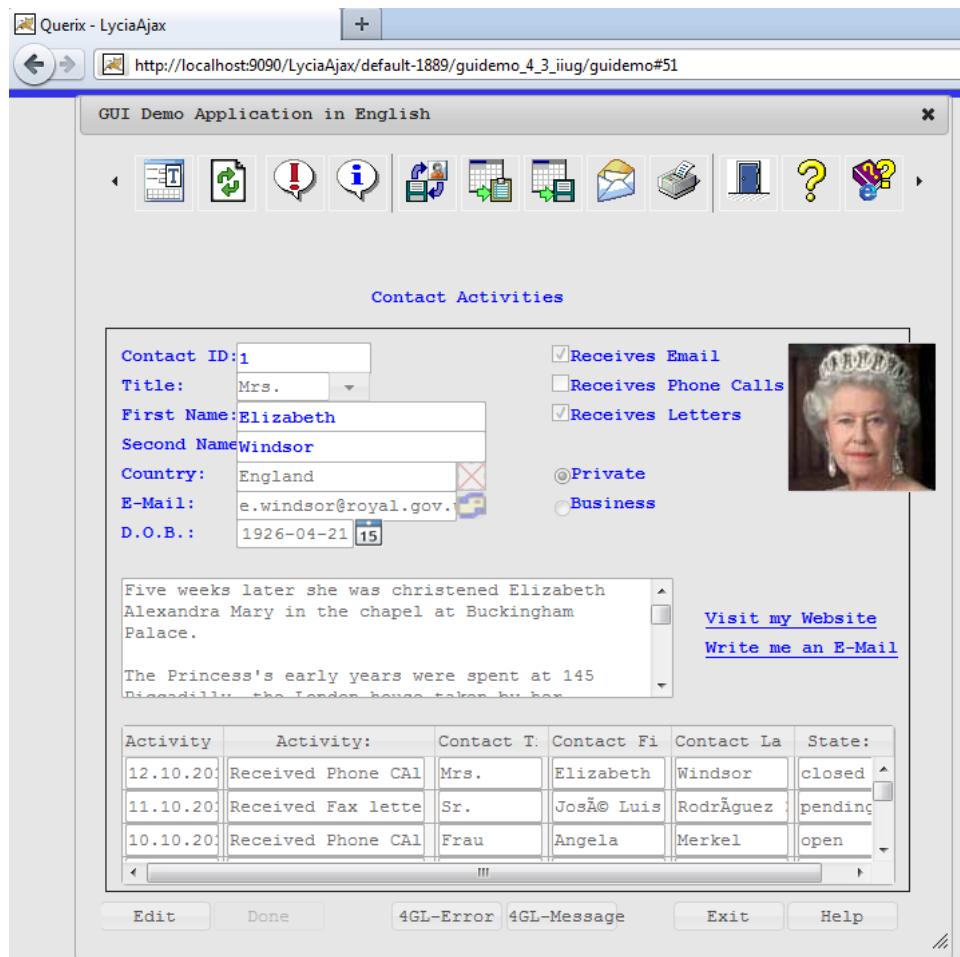
Here you can chose the application you want to use and click the corresponding link. The application page will open in your web browser.

LyciaWeb

One of the most essential functions of Web Application Server is that it transforms GUI protocol to HTTP and vice versa, thus allowing to pass 4GL applications data to web environment. LyciaWeb is a web client which allows the user to launch 4GL applications in web browser, in case they have access to Web Application Server.

LyciaWeb performance is provided by LyciaWeb application, stored on Web Server. This application provides the requests pass from web environment (a web browser) to application server on the computer and helps to retrieve information from application server and pass it to web environment. This allows the user run 4GL applications even if they do not have Lycia products installed on their computer. The only requirement here is that the user has to have access to web application server and know the host number of the computer on which the application server with the program is stored.

LyciaWeb can be run by both ways described above, as well as from Lycia Studio. Lycia Studio toolbar contains a button "Run with LyciaWeb", which starts your system default web browser and runs the application in its tab.



For more detailed information on LyciaWeb, see 'LyciaWeb Guide' document.

Web Services

Querix Web Application Server is a tool that makes it possible to develop web services and add them to Lycia 4GL applications. Web application data are stored on the server in WSDL documents (one for each web service) which contain the descriptions of all functions the web service contains. These files are referenced by Lycia to create .xml documents which are then used by 4GL applications.

Querix Web Application Server already contains the deployed axis2 service which is required for deploying and running web services.

You can find the detailed information about web services in the 'Lycia Web Service Getting Started' document.



Database connectivity

Many 4GL programs require access to a database server, and so it is necessary to provide 'Lycia' with the connection details for that server. There are several different methods for this; which one you use is dependent on your platform.

Since Querix-4GL is a complete implementation of Informix-4GL, all database connections are made to emulate Informix connection conventions, which will ensure that your 4GL source code requires no changes to access multiple RDBMSs.

Database connections are discussed in more detail in the various dedicated Database Migration guides, which can be downloaded from our website, www.querix.com. These Migration guides cover such databases as Informix, Oracle, MySQL and other frequently used in individual guides.

Lycia II makes it possible for a single application to connect different databases throughout its execution. These databases need not be on the same database server or even of the same type.

For an application to connect to a database the following requirements should be met:

1. Establish connection between the database and its respective database client. E.g. between Informix Client SDK and Informix database server.
2. Specify the connection details in the database configuration file database.cfg.
3. (Optional) Set the default database for compilation and launching which will be used if the addressed database will not be found in database.cfg.

Changing the default database

 A circular icon containing a white lowercase letter 'i' inside a purple circle, with a light purple oval shadow behind it.	Note: In Lycia II the default database setting only takes effect, if the database name does not appear in the database.cfg file. If the database is found in the database configuration file and has a set driver property, the data from the database.cfg is used at compilation and at runtime and the default database is ignored.
--	--

You can choose to which database to connect in LyciaStudio by setting the database using the **Window -> Preferences -> 4GL -> Database** preferences page. This preferences page specifies which database LyciaStudio should connect to while compiling and when running applications in character mode. This option changes the contents of the LYCIA_DB_DRIVER variable in the env.properties file responsible for the environment variables used when working with the Studio and running programs in character mode when they are launched from the Studio. It is not advisable to edit this file manually.



You can also set the LYCIA_DB_DRIVER in the inet.env file (for applications run in GUI mode), and in the environ.bat file (for command line activities) to specify which database you want to connect to. This variable is optional and will override the default database. The value of the LYCIA_DB_DRIVER variable in all three files (or only in the files responsible for the functionality you use) must be set each time you change the database or connect to a new database. For more information about the environment variables and the files they are stored in see the "[Configuration settings](#)" chapter.

Below is a list of valid databases supported by the LYCIA_DB_DRIVER environment variable:

oracle – Native connection to Oracle databases

informix – Native connection to Informix databases

sserver – ODBC connection to Microsoft SQL Server databases (Windows only)

db2 – ODBC connection to IBM DB2 databases

odbc – General ODBC interface

For example:

```
LYCIA_DB_DRIVER=informix
```

The database driver may also be set by appending the -d flag to an application on the command line, this overrides the variables in the environment settings.

However, only setting the driver is not enough - you need to set up the database clients provided by the corresponding database vendors to be able to connect to a database.

Setting up Database Clients

The first step in connecting to a database is setting the connection between the database server and the database client.

Informix Client

To be able to connect to Informix database you need Informix Client SDK to be installed on your machine. Note that SDK 32 needs to be installed, if you use Lycia 32 bit and SDK 64 bit needs to be used for Lycia 64 bit.

On Windows

For native Informix database connections, the standard Informix connection manager **SetNet32** is used. For the information regarding how this manager should be used to set up the connection, refer to the documents provided by IBM Informix.



The INFORMIXDIR (pointing to the Informix install directory) and INFORMIXSERVER (pointing to your database server) environment variables must be set in the inet.env file, if you want to run applications using the application server.

You must also set the default database driver to "informix" as described in the "[Changing the default database](#)" section above.

On Unix/Linux

You must set the environment variable INFORMIXSERVER to point to your database server. For example, run the command on UNIX/Linux:

```
INFORMIXSERVER=my_database_server  
export INFORMIXSERVER
```

The 'set' command is used to set the environment variable on Windows using Lycia Command Line Environment. It is important to ensure that the INFORMIXDIR environment variable is pointing to the Informix install directory (usually set as /opt/Informix) and that %INFORMIXDIR/bin in PATH, these should be set during the Informix install. If this variable is not set, you must also set it. The same variables must be also set in the env.properties file and in the inet.env file.

The database connection configuration is set in the file sqlhosts located in %INFORMIX/dir/etc. If this file does not exist, please consult your Informix documentation. This file will contain one line for each of the servers that you want to connect to. Each line should comprise four, space-separated fields, which are:

Instance name	Name of the Informix instance
Protocol	Connection Protocol for this instance
Server	Host name or IP of the database server.
Service	Service name of the instance

For example:

```
informix_on_line olsocotp myserver turbo
```

You must also set the default database driver to "informix" as described in the "[Changing the default database](#)" section above.



Oracle Client

On Windows

In general, an Informix database name is mapped to an Oracle user. Please see the Querix Migration Guide for more details.

Oracle database client provides the file called tnsnames.ora configuration file that defines databases addresses for establishing connections to them where all oracle service names should be recorded.

You may need to set the ORACLE_HOME and ORACLE_SID environment variables in the inet.env file for running programs using the application server. Further adjustments should be done in the database.cfg configuration file.

You must also set the default database driver to "oracle" as described in the "[Changing the default database](#)" section above.

On UNIX/Linux

For a database connection on a Unix/Linux platform, two environment variables need to be set. These are ORACLE_HOME and ORACLE_SID. They must be set in the command line and also in the inet.env and in the env.properties files.

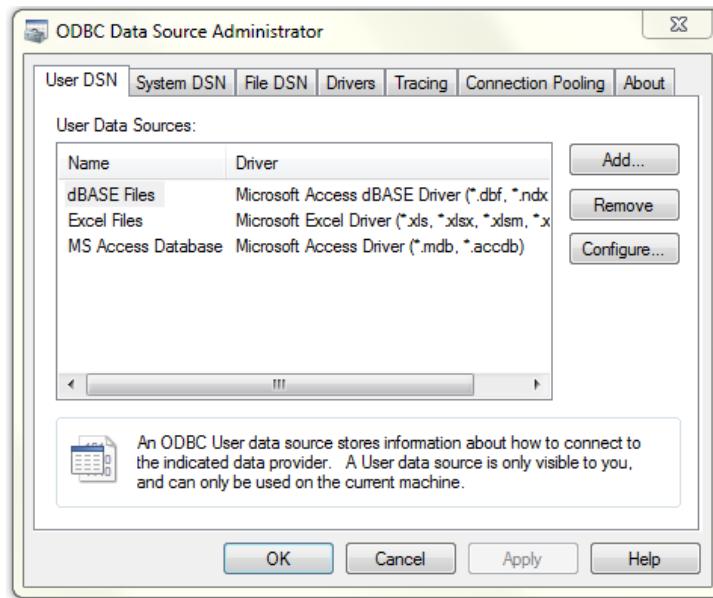
Oracle database client provides the file called tnsnames.ora configuration file that defines databases addresses for establishing connections to them where all oracle service names should be recorded. Further adjustments should be done in the database.cfg configuration file.

You must also set the default database driver to "oracle" as described in the "[Changing the default database](#)" section above.

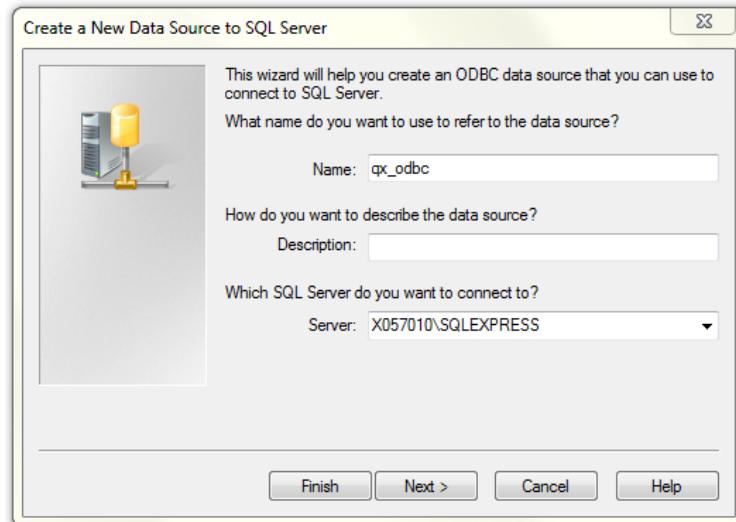
ODBC Client

The ODBC Administrator tool on Windows platforms can be used to establish connections to other database management systems that use SQL. In Windows 7 it is located in Control Panel\All Control Panel Items\Administrative Tools\ Data Sources (ODBC).

Individual connection profiles can be set up using the tabs in the ODBC Data Source Administrator window:



Press the **Add...** button and then enter the name of the data source and select the server, i.e. SQL Server. In the new dialog enter the DSN name and select the server to connect to:



Note: The name of the data source entered in this dialogue (e.g. qx_odbc) must be the same as the name of the database you want to connect to.

After that, follow the steps of the dialogue for new data source creation.



You may want to set the ODBC_DSN variable. If this variable not set, the 4GL considers that the DATABASE statement specifies the DSN name. If you set this variable, the 4GL treats the value of the ODBC_DSN variable as the DSN name and the DATABASE in such case points to a catalogue in the database to which the 4GL should switch when connected. For the graphical clients set the ODBC_DSN variable in the inet.env file. To open inet.env file in LyciaStudio, go to **Window-> Preferences -> 4GL -> Run/Debug -> GUI Servers**, select the GUI server from the list and press the **Edit Environment** button. The file will be opened in the editor area of the Studio.

You must also set the default database driver to "odbc" as described in the "[Changing the default database](#)" section above. If you connect to an SQL Server using the ODBC connection, you should set the database driver to "sserver", otherwise the functionality can be restricted and in some cases the connection might fail.

For detailed information concerning the connection to the databases, please see the appropriate documentation for your database type and the corresponding Querix migrate guide, which are available for download from our web-site.

Connecting to Multiple Databases

In Lycia II a 4GL program can connect only to several databases at a time, these databases can be located on different database servers and belong to different database vendors. This makes the task of migrating the data from one database to another extremely easy.

From the perspective of 4GL code, the CONNECT TO, SET CONNECTION, and DISCONNECT statements handle simultaneous connection to several databases. For the syntax and usage of these statements see the "[Querix 4GL Reference](#)" guide.

From the perspective of environment settings and database connection settings, they are all handled in one file called database.cfg located in %LYCIA_DIR%/etc/ directory. This file should contain information about all the databases you plan to connect to.

	Note: You should first set up each database client you want to use in accordance with the database vendor requirements. Before using the settings described in this section you must set up the database connection as described above in Informix , Oracle or ODBC section.
---	---

Database Configuration File

The database.cfg file is created empty during Lycia installation. The file structure should be as follows:



```
dbname {  
  
    driver = "dbtype"  
  
    source = "databse"  
  
    username = "user"  
  
    password = "pwd"  
  
}
```

The options listed above should have the following values:

- database_alias - the alias that will be used in 4GL code as the database name, for example, in the DATABASE statement or in the CONNECT TO statement
- driver - its values should be the database type. This parameter is optional, if it is not set the value of the LYCIA_DB_DRIVER variable will be taken as the driver. It can be one of the following keywords:
 - informix
 - oracle
 - odbc
 - sserver
 - db2
- source - its values should be the name of the database as it is specified on the database server. This parameter is optional, if it is not set for the given database connection, the database alias is treated as the source.
- username - should contain the login of the user. This parameter is optional. If it is not set
- password - should contain the access password for the given user login. Note, that the password is stored as plain text. This parameter is optional. For the security reasons use the USING clause of the CONNECT TO statement instead.

The database.cfg file can contain a number of such database specifications, their number is not limited. Here is an example of this file containing two database specifications:

```
or_db {  
  
    driver = "oracle"  
  
    source = "stores"
```



```
username = "john"

password = "123456"

}

ifx_db {

driver = "informix"

source = "stores@ixserv"

username = "jane"

password = "567890"

}
```

Source

The source value differs depending on the database type. In some cases, like for connecting to Informix, it may include the server name, if you use several Informix servers. The table below lists the values of this parameter with regard to the database type:

Database Type	'source' value	Description
IBM Informix	dbname[@dbserver]	Informix database name with optional server specification
Oracle	tnsname	Oracle TNS Service name typically stored in file tnsnames.ora
SQL Server	datasource	ODBC data source name
MySQL	dbname[@host[:port]] or dbname[@localhost~socket]	Database name with optional host name and TCP port or with optional localhost and Unix socket file specification
PostgreSQL	dbname[@host[:port]]	Database name with optional host name and TCP port
IBM DB2	dsname	DB2 Catalogued Database

For the databases of other types refer to their documentation provided by their respective vendors.



Precedence of Parameters

The database connection can be set in the database.cfg file and then overridden by the CONNECT TO statement completely or partially. Since all the clauses of the CONNECT TO statement and the parameters of the database connection in the configuration file are optional, there may be a set of cases when some parameters are missing and some are set twice.

Generally speaking CONNECT TO has higher precedence than the database.cfg and database.cfg has higher precedence than the environment variables (i.e. LYCIA_DB_DRIVER) and flags (-d). The -d flag of [grun](#) and the environment variables have the lowest precedence in order to enable the usage of different database drivers throughout the program.

The following table shows the result of some parameters being set twice or not being set at all.

Parameter	Set in CONNECT TO	Set in database.cfg	Used Value
driver	yes	yes	CONNECT TO
	yes	no	CONNECT TO
	no	yes	database.cfg
	no	no	-d flag for program launching is used. If it is not passed - LYCIA_DB_DRIVER value will be used.
source	yes	yes	CONNECT TO
	yes	no	CONNECT TO
	no	yes	database.cfg
	no	no	The dbname value is treated as source.
username	yes	yes	CONNECT TO
	yes	no	CONNECT TO
	no	yes	database.cfg
	no	no	DB-specific settings are used, if they are not set then program tries to connect with no username
password	yes	yes	CONNECT TO
	yes	no	CONNECT TO



no	yes	database.cfg
no	no	Legacy settings are used, if they are not set then program tries to connect with no password

Here is an example with not all optional parameters present:

```
stores {  
  
    driver = "oracle"  
  
    username = "john"  
  
    password = "123456"  
  
}  
  
ifx_db {  
  
    source = "stores@ixserv"  
  
}
```

If we use the CONNECT TO without optional clauses, we will get:

- In the case of stores connection setting the program will look for the tnsname 'stores' in the tnsnames.ora file and use the specified login and password for connection.
- In the case of inf_db connection setting, it will look for the database named 'stores' on the ixserver Informix database server, if the program is launched with -d informix flag. If it is launched without this flag, the runtime error will occur. If the value of the flag is not 'informix', but, for example, 'oracle', the program will look for the tnsname 'stores@ixserv' in the tnsnames.ora file and would most probably also throw an error.

Legacy Database Settings

If the database connection used is not set in dbconfig.cfg file and the DATABASE statement instead of the CONNECT TO statement is used, the legacy database connection settings are used for compatibility.

- For Informix - the environment variables LOGNAME and INFORMIXPASS are used.
- For Oracle - the changes in the register made by the Connect Tool on Windows and the content of the dbtrans file on Linux/UNIX are used.
- For ODBC - the value of the ODBC_DSN variable referencing a valid ODBC DSN name is used.



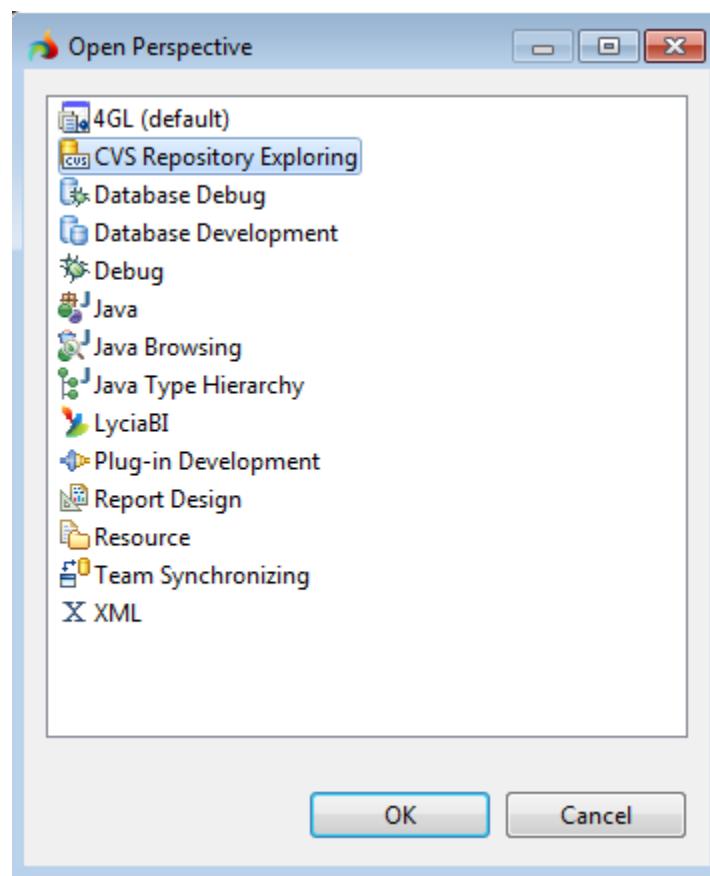
Working with repositories

LyciaStudio provides you with the opportunity to access to Concurrent Versions System (CVS) repositories by means of the built-in CVS client.

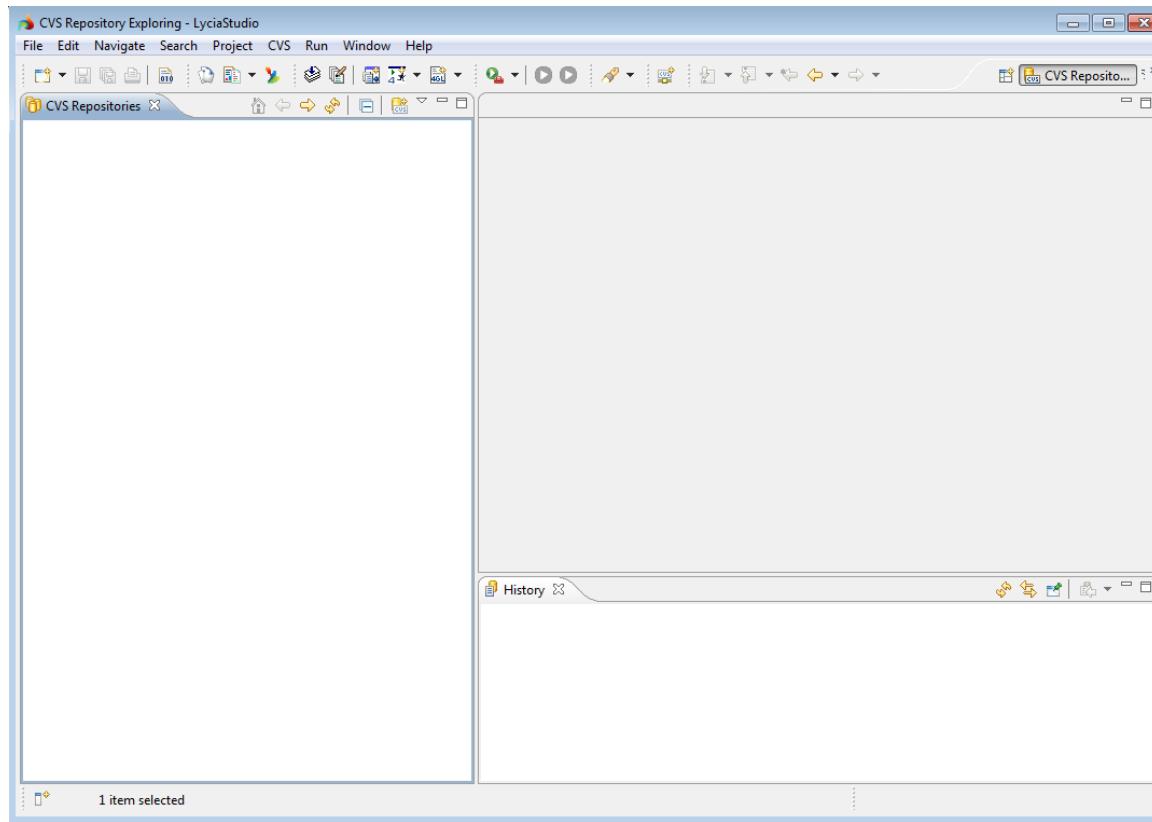
CVS Repository

LyciaStudio contains a built-in CVS client which allows you to access CVS repositories. This guide will touch upon the CVS tools of the Studio and ways of interaction between CVS and LyciaStudio. You can find more information on CVS as such at <http://ximbiot.com/cvs/>.

LyciaStudio has a built-in perspective for working with CVS. To open this perspective, go **Window -> Open Perspective -> Other -> CVS Repository Exploring**.

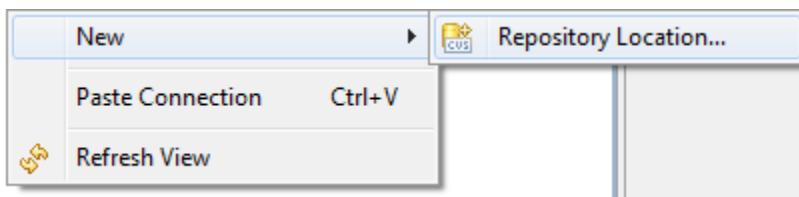


The CVS perspective will be opened, though it will be empty as no CVS locations are specified yet:

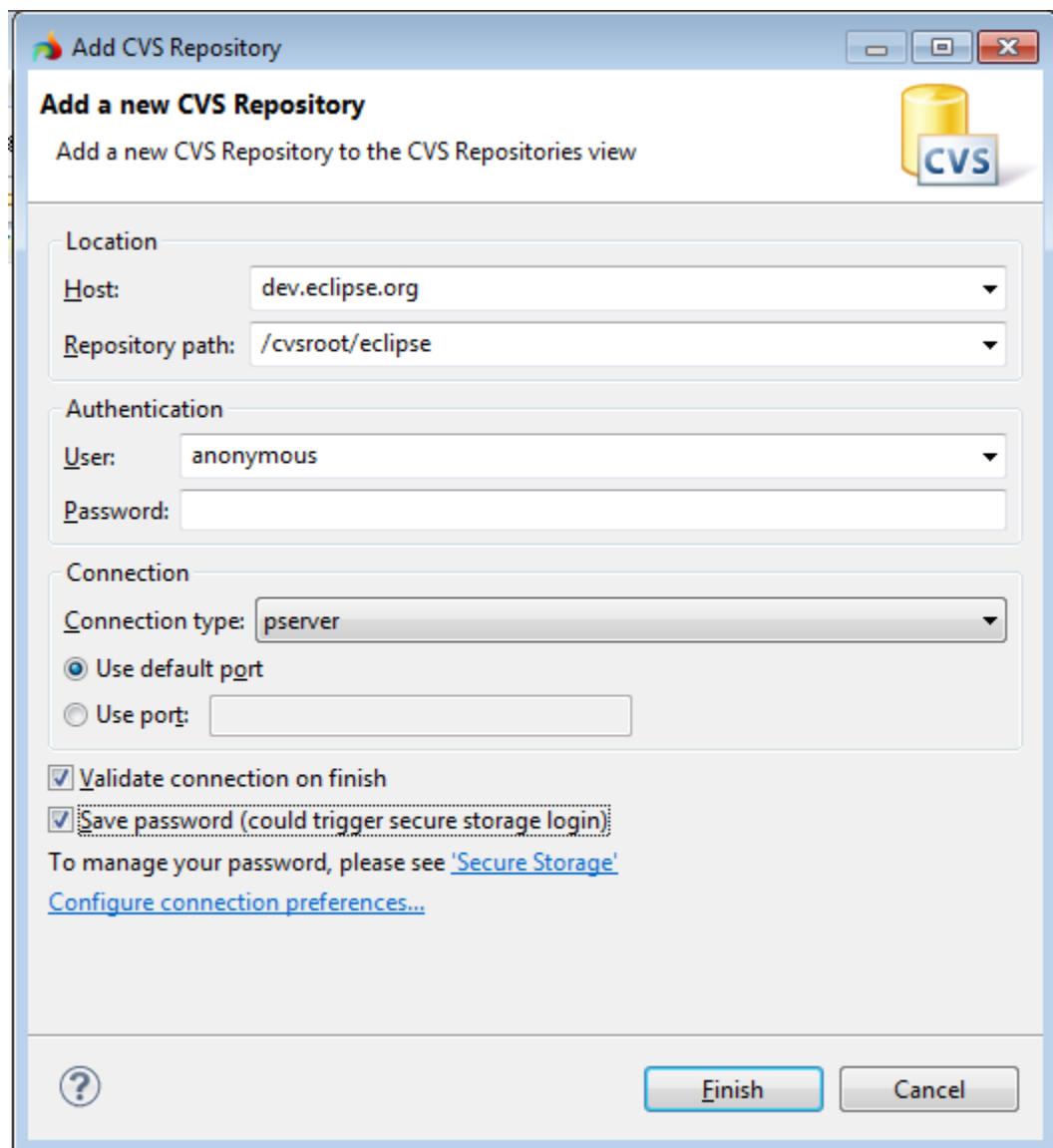


To create a repository location, follow these steps:

1. Press the “Add CVS Repository” button on the CVS Repositories toolbar or right-click the CVS Repositories view and select **New -> Repository Location...**



2. You will be presented with the Add CVS Repository dialogue.
3. Enter your CVS data into the corresponding fields. To test the CVS connection you can enter the data of the Eclipse CVS repository:



4. Select the "Validate connection to finish" to check whether LyciaStudio can connect to the specified CVS repository before closing the dialogue.
5. Press the **Finish** button. LyciaStudio will try to connect to the server:



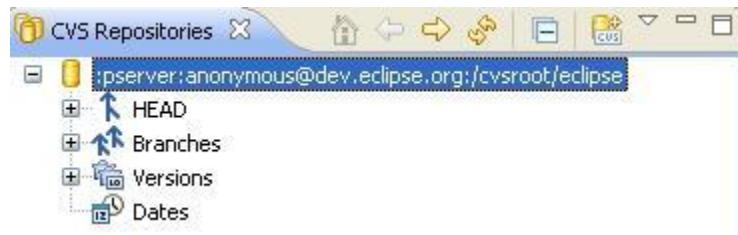
6. If the connection is successful, you will see the CVS location in the CVS Repositories view:





CVS Repositories view

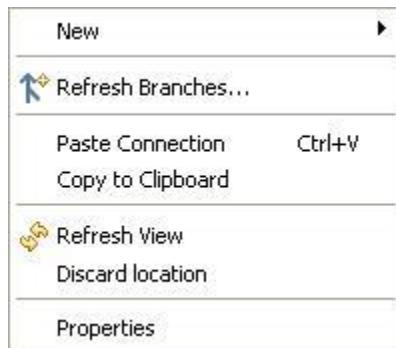
The CVS Repositories view shows the CVS repository locations which you have added to the Studio. Click on the plus button next to the repository location to expand the repository location tree which includes the main trunk (HEAD), project versions and branches in that repository. You can further expand the project versions and branches to reveal the folders and files contained within them:



This view offers some tools for file manipulation.

Context menu of the repository location

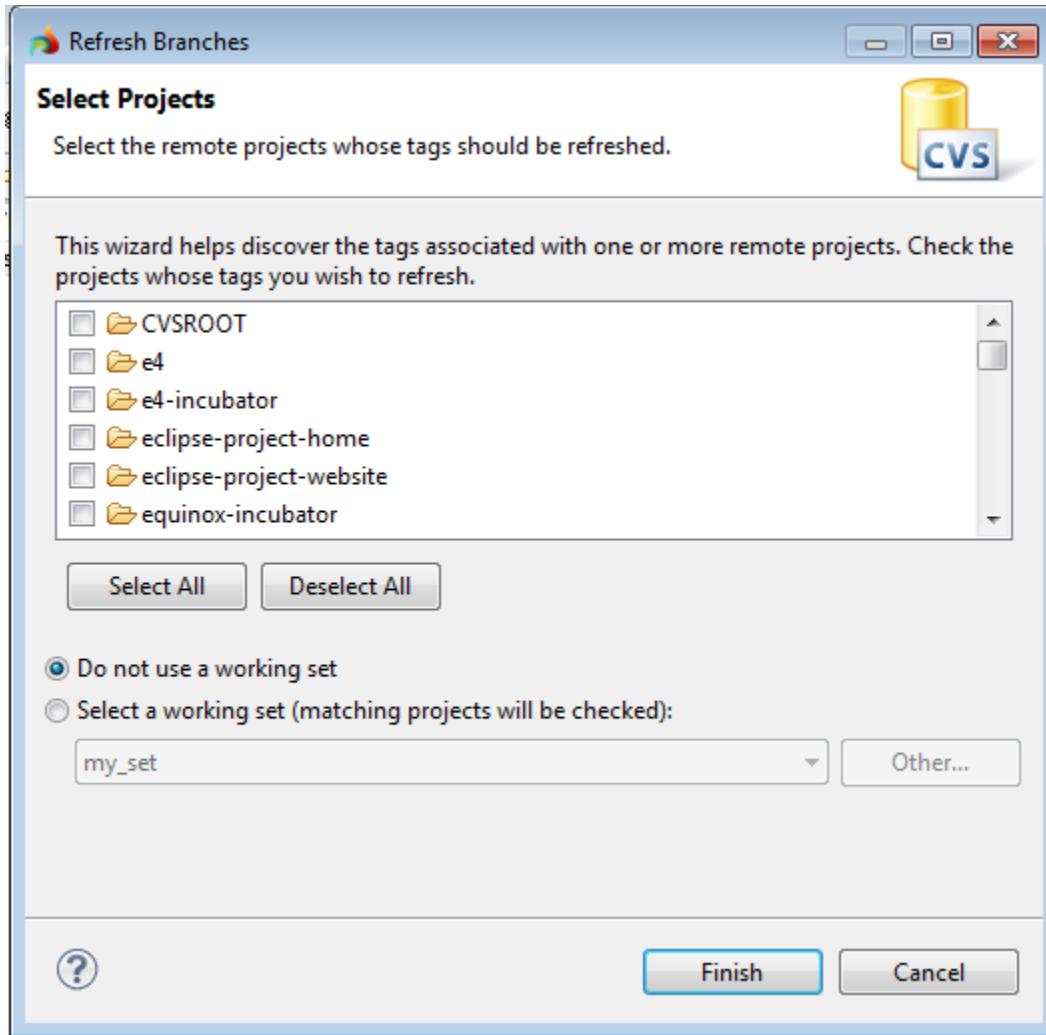
Right-click on the repository location, to open its context menu:



- The **New** option allows you to create more repository locations
- The **Refresh Branches** option allows you to refresh the list of known branches and versions that are displayed in the repositories view for selected projects.



You can select the projects you want to refresh in the Refresh Branches dialogue:

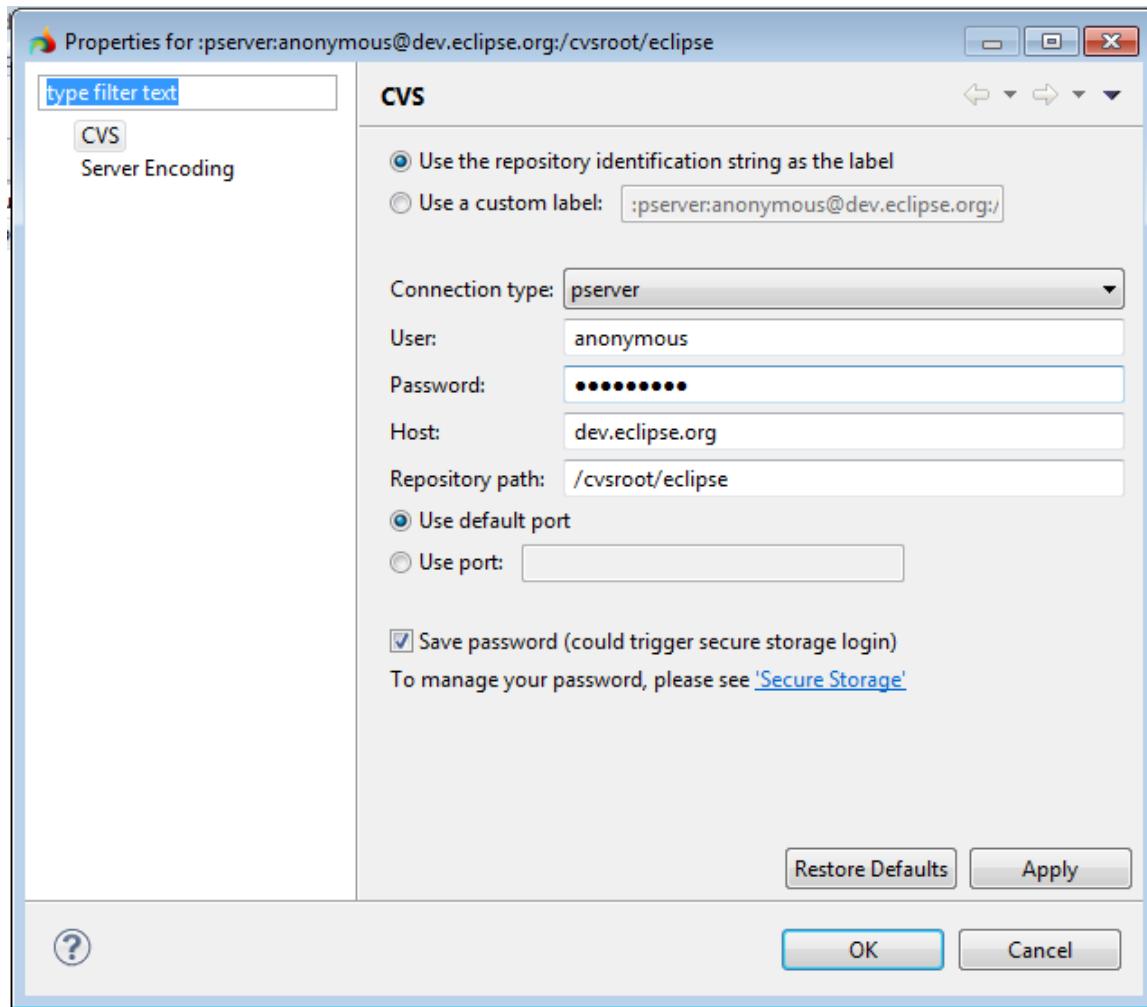


If the operation fails for a particular project, use **Configure Branches and Versions** in the context menu of a project to select one or more appropriate refresh files.

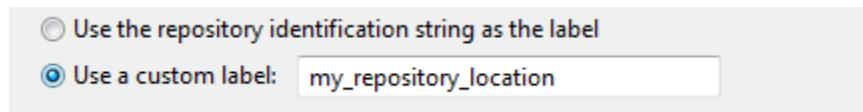
- The **Paste Connection** option creates a new repository location using on the content of the clipboard. The clipboard must contain a valid repository location string.
- The **Copy to Clipboard** option copies the identification string of the selected repository location to the clipboard.
- The **Refresh View** option is used to refresh the repository folder tree.
- The **Discard Location** option removes the selected repository location. All projects mapped to this repository must first be unmapped.



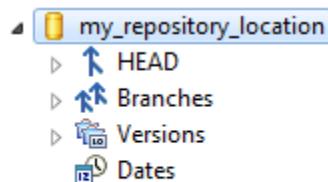
- The **Properties** option opens a dialogue where you can modify any of the repository location properties:



- Here you can also assign a display name to the repository location by selecting the "Use a custom label" option and typing the new name for the repository location:

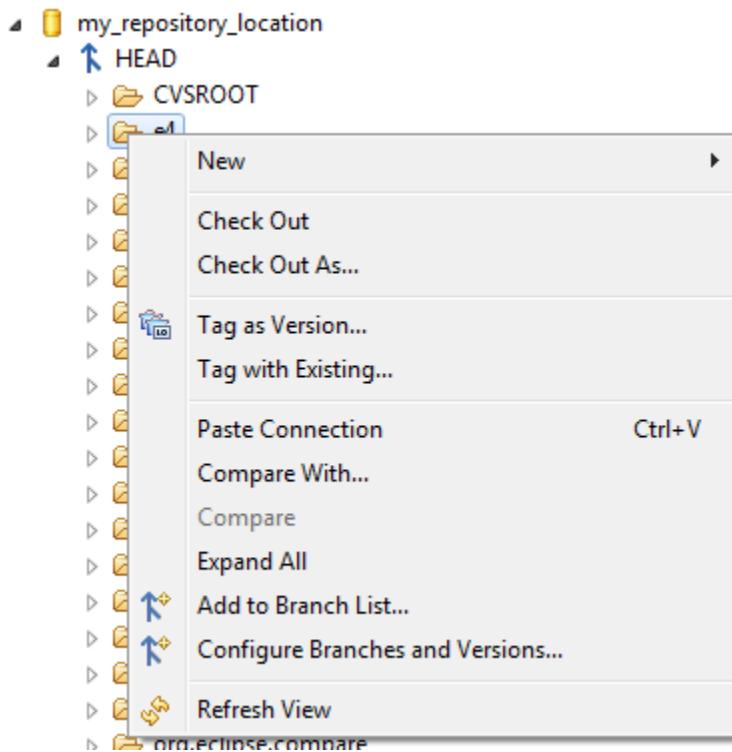


- This name will be displayed in the CVS Repositories view, when you press the **OK** button:



Context menu of a project

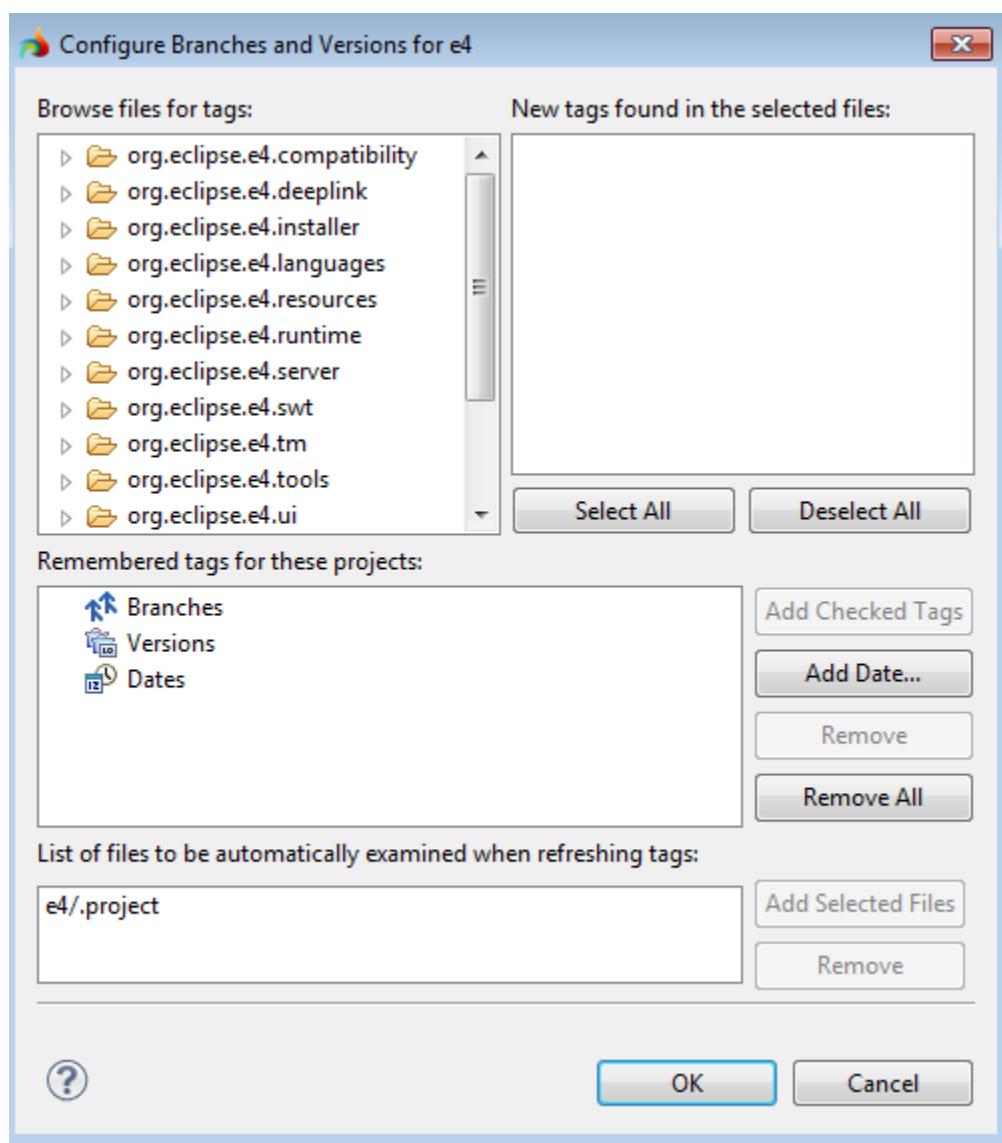
Expand the HEAD branch and right-click on any project folder which it contains. You will see the context menu of a project:



- The **Check Out** option checks the selected CVS modules out into projects in the Studio with the same names as the remote modules.
- The **Check Out As...** option opens a Check Out Dialogue in order to allow the configuration of how the selected modules are checked out into the Studio.
- The **Tag as Version...** option versions the selected resource based on the current branch contents. This option is available only if you have the write access to the repository.
- The **Tag with Existing...** option versions the selected resource based on the current branch contents, moving the tag from previously tagged resources if required. This option is available only if you have the write access to the repository.

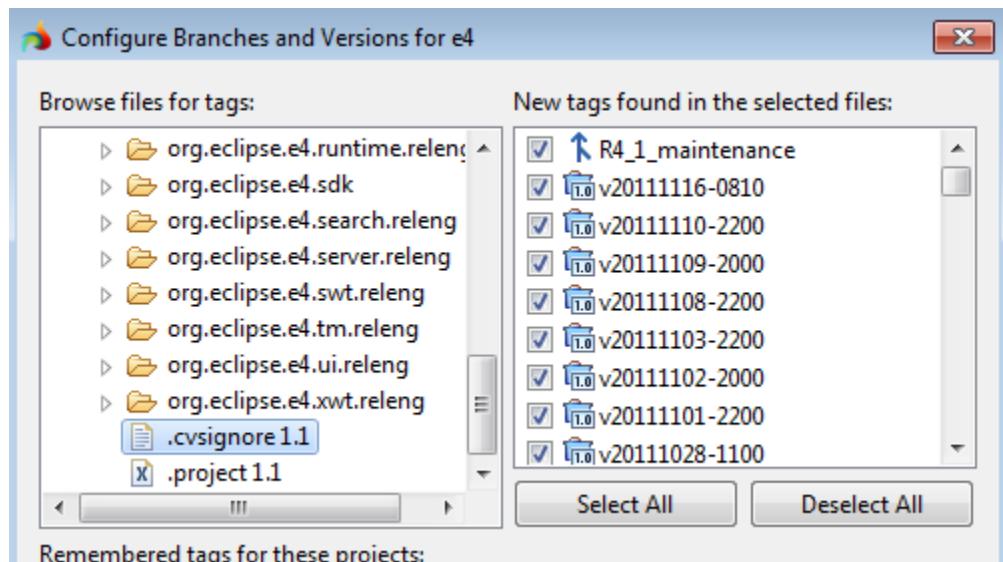


- The **Compare With...** option compares the selected folder with a branch or version of the same folder:
- The **Add to Branch List** option adds the selected project to the list of projects that are displayed under the specified branch in the repositories view. This command only modifies the repositories view and does not affect the repository in any way. If you want to add the project to a branch, you can perform a **Tag with Existing** after performing this operation.
- The **Configure Branches and Versions** option opens a dialogue which helps to find the branch and version tags that exist in the repository for the selected folder so they can be added to the repositories view to allow the resources that have these tags can be browsed:



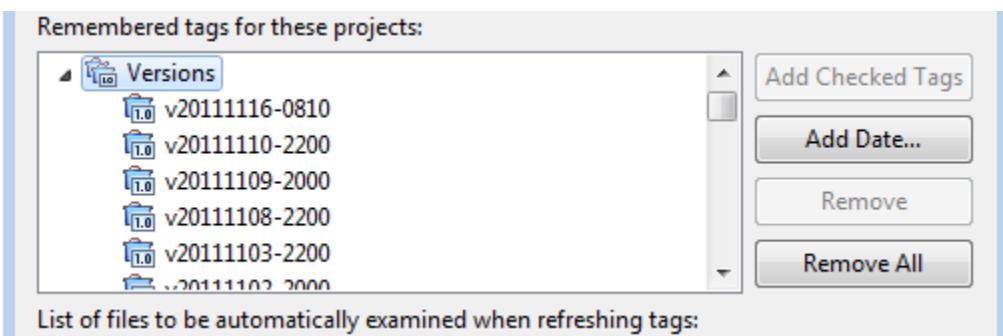


- Expand the folders and select a file with tags, the available tags will be displayed in the right pane:



Remembered tags for these projects:

- If you press the **Add Checked Tags** button, the selected tags will be added to the versions section:

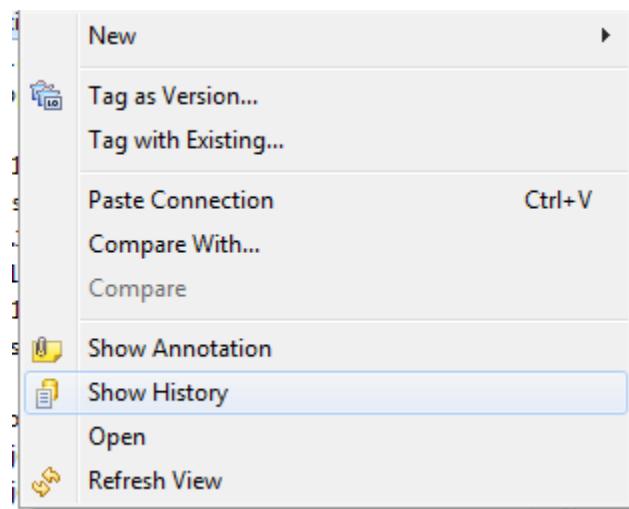


History View

The History view provides a list of all the revisions of a resource in the repository as well as all the revisions of a resource in the local history. Here you can:

- Compare revisions
- Load a revision
- Revert a workspace file to a revision
- Show annotations
- Open a revision in an editor

To open a History view right-click on a file in the CVS Repositories view and select the **Show History** option



The history for the selected resource will be opened in the History view:

A screenshot of the History view window. The title bar says "History" and the file path is "Application.xmi". The window contains a table with the following data:

Revision	Tags	Revisi...	Author	Comment
1.3		12/6/09 10:4...	yvyang	update to ne...
1.2		10/19/09 10:...	yvyang	update to ne...
1.1		7/30/09 12:3...	yvyang	Initial version...



Double-click on a revision, to open it in the editor:

The screenshot shows the Lycia IDE interface. At the top, there is a code editor window titled "Application.xmi 1.3" displaying XML code. Below the code editor is a toolbar with various icons. Underneath the toolbar is a "History" view window titled "Application.xmi". The history view contains a table with three revisions:

Revision	Tags	Revisi...	Author	Comment
1.3		12/6/09 10:4...	yvyang	update to new versi...
1.2		10/19/09 10:...	yvyang	update to new UI w...
1.1		7/30/09 12:3...	yvyang	Initial version for XWT

You can also right-click on a revision and select the **Show Annotation** option to see the contents of file with annotations identifying the author of each line of the code:

The screenshot shows the Lycia IDE interface with the code editor window. A tooltip is displayed over the XML code, showing the author "yvyang" and the date "1.3 Dec 6, 2009 10:40 PM". The tooltip also includes the text "Press 'F2' for focus". The background of the code editor shows the XML code for the application.

The History view has its own toolbar with the following options:

- - these buttons switch the display mode of the History view between remote revisions, local revisions or both
- - this button turns on the compare mode in which the revisions are opened in the Compare editor when double-clicked
- - this button groups revisions by date



- - this button allows you to select a resource for which a history has been viewed in the past. This button is disabled if you have not viewed the history of other resources.

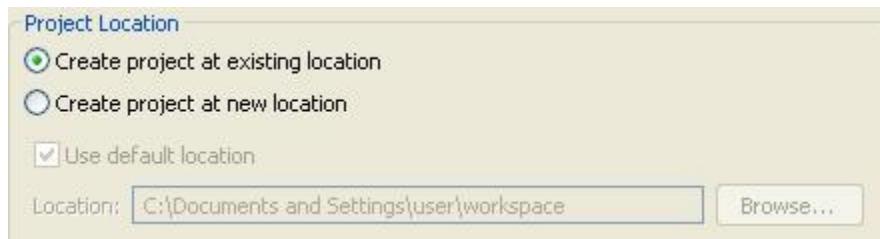
Sharing projects in Lycia

A project located in your workspace cannot be shared with other developers, because Lycia locks the workspace when it is in use. Each user must have their own workspace on their local system or on a mapped or mounted drive where the projects are stored.

However, a project does not need to be located in the workspace and as Lycia has tools for working with various repositories, it is possible to share projects even if the workspace cannot be shared.

'Hydra Studio' legacy projects and Lycia projects can be imported into the LyciaStudio by means of the Import option in such a way that a project will be opened in the Studio though physically it will be stored in the directory from which it has been imported:

- To import a 'Hydra Studio' legacy project without copying it into the workspace directory, use the **Import... -> 4GL > 'HydraStudio' Project** wizard and select the "Create project at existing location" option.



- To import a Lycia project without copying it into the workspace directory, use the **Import... -> General -> Existing projects into workspace** wizard and select the "Copy projects into workspace" check box.





	This can be done only if you import a project that has been exported to the file system not to an archive file
--	--

See the “Importing a Lycia project” and “Importing a ‘Hydra Studio’ V4 project” sections of the “Using LyciaStudio” chapter of this guide for details.

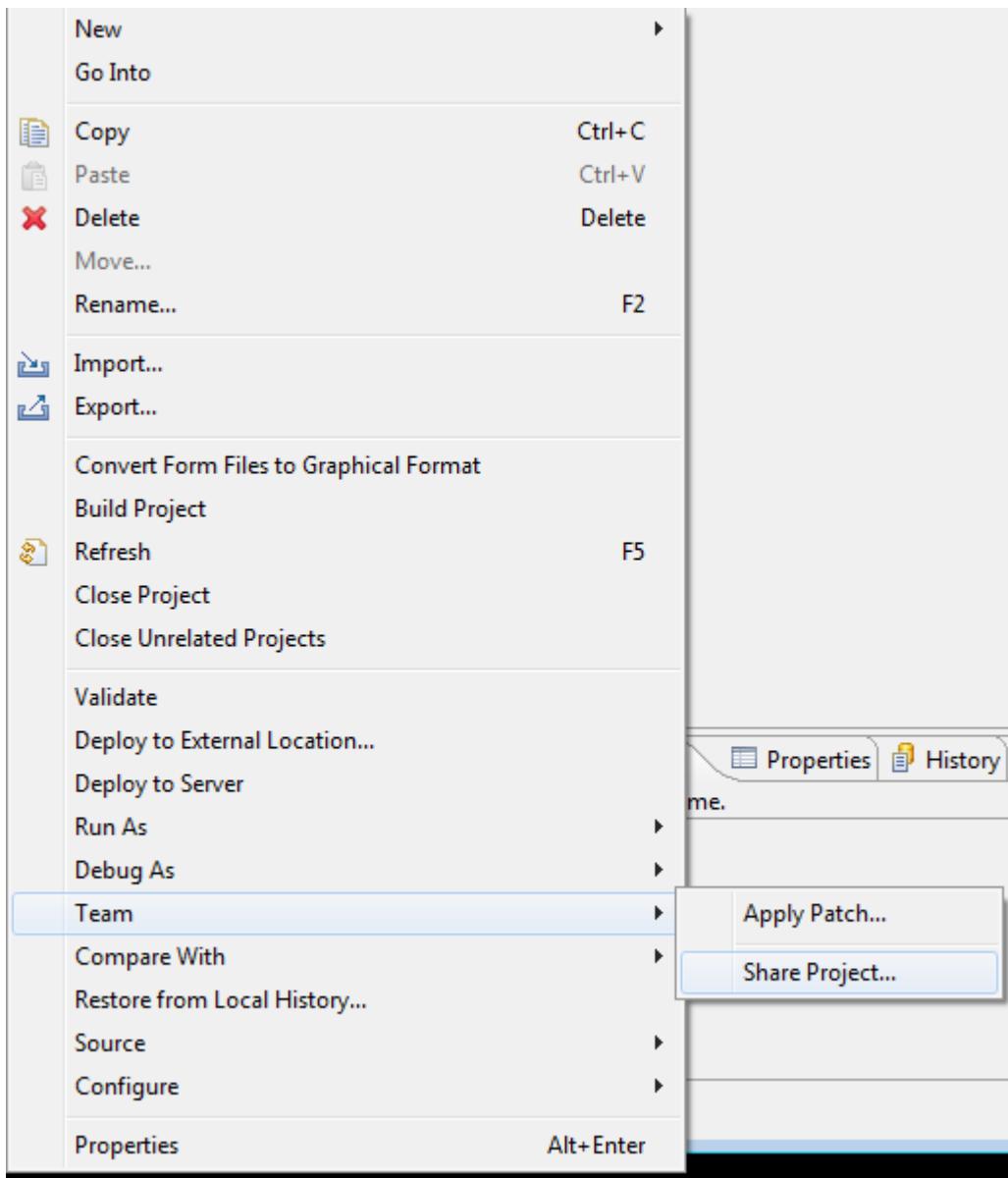
Thus a project located elsewhere can be opened in your workspace.



Sharing with the help of CVS

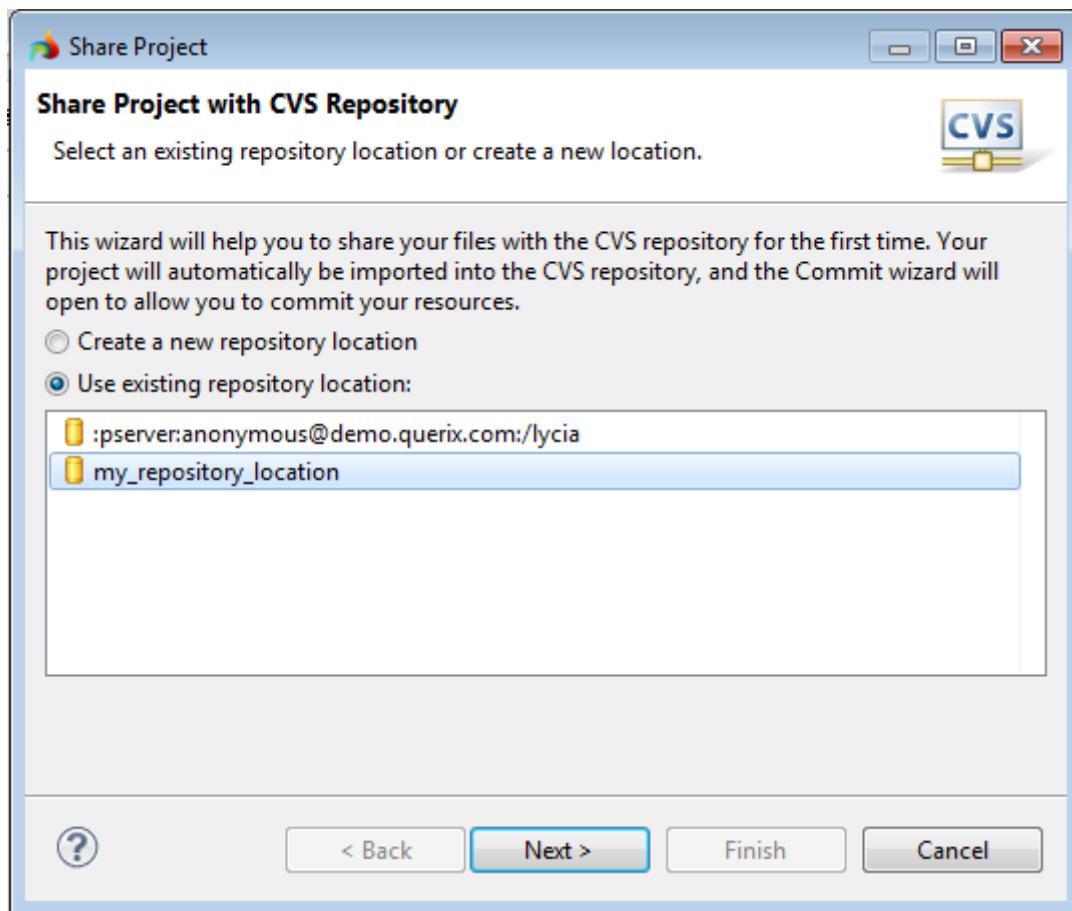
To share a project with the help of Lycia do the following:

1. Right-click on a project or a number of selected projects in the Project Explorer view
2. Select the **Team ->Share Project** option from the context menu:





3. You will be presented with the Share project dialogue where you can either select an existing CVS repository location or create a new one.

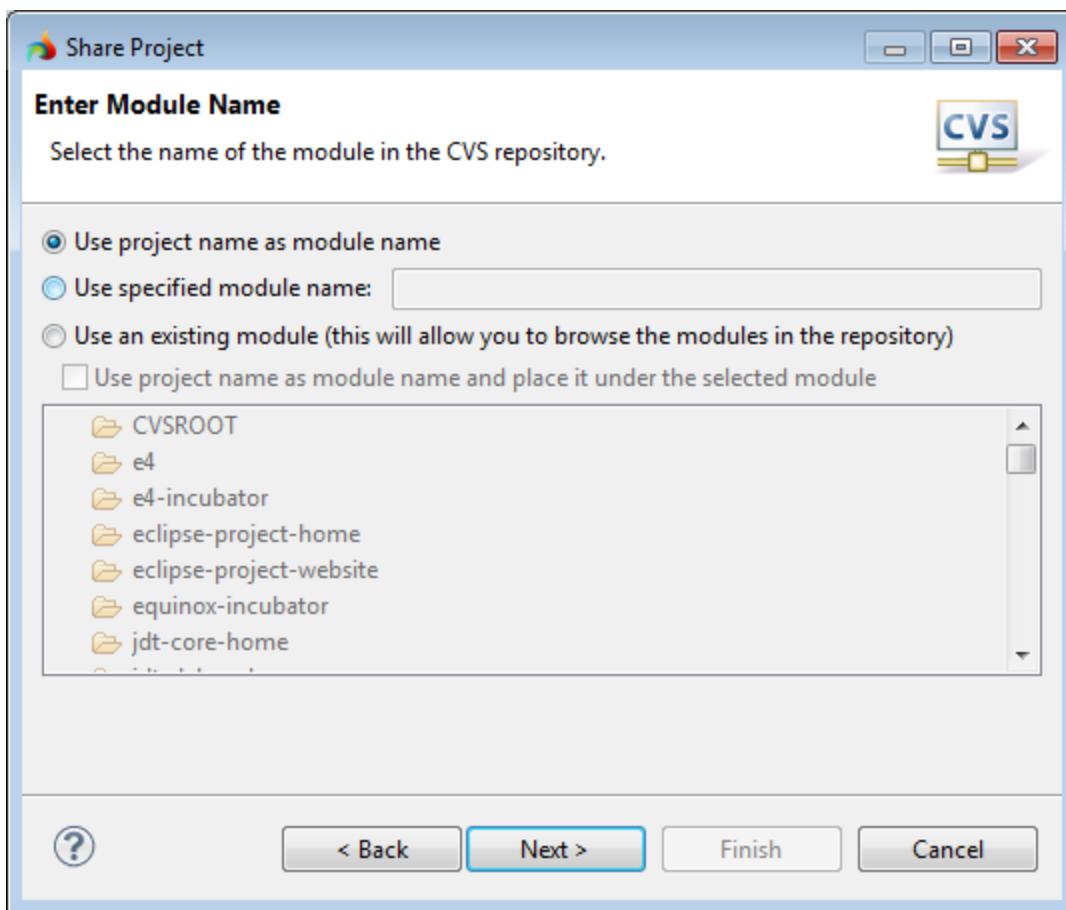


If you have clients for other repositories installed, this dialogue will be preceded with another dialogue allowing you to select a repository type.

- a. Select an existing location and press the **Next** button.
- b. If you do not have an existing location, select the "Create a new repository location" and press the **Next** button. Create a new repository location and press the **Next** button once more. For details about how to create a repository location see the "CVS Repository" section of this chapter.



4. The next page of the Share Project dialogue allows you to select the name for the module repository:



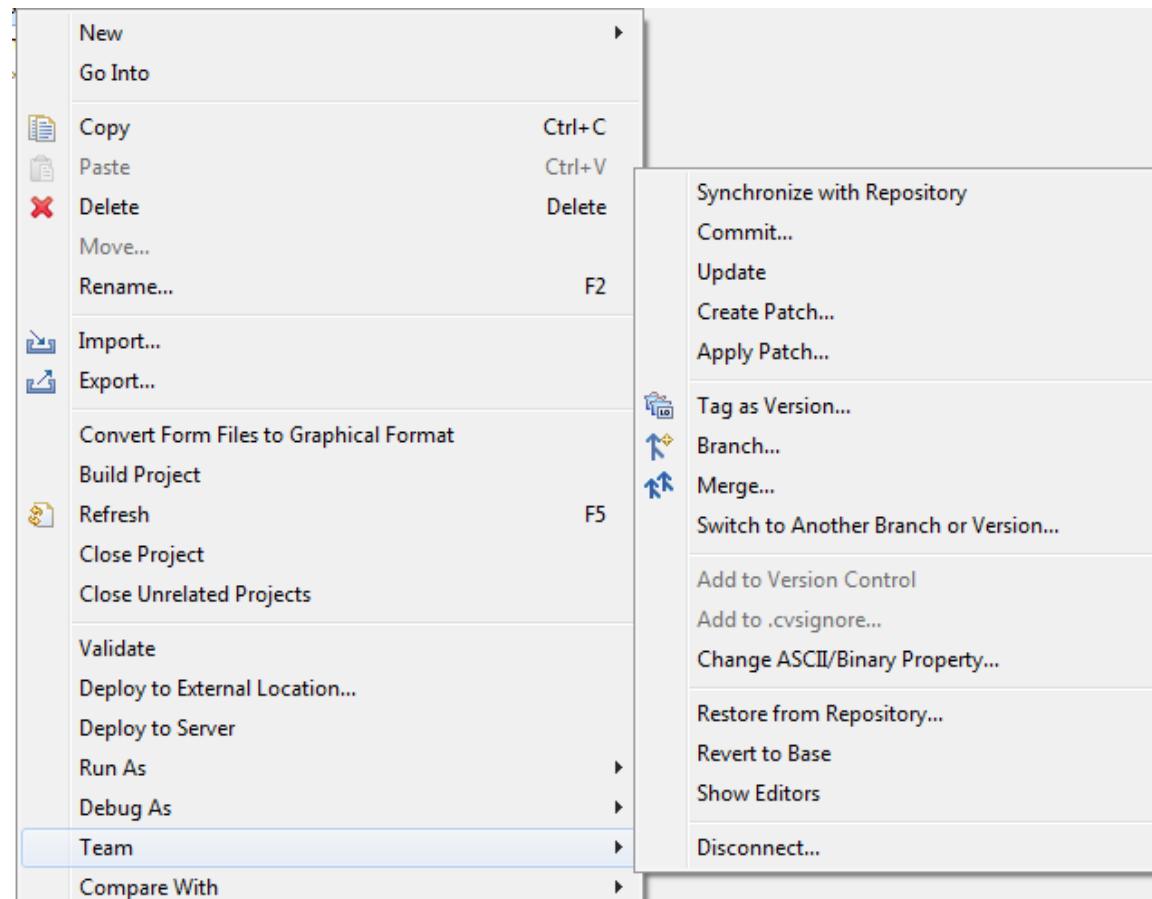
You can use the name of the project as the module name, you can specify a module name or you can use one of the modules which are already in the repository. After you have selected the module name, press the **Next** button.

5. If you have the write access to the repository, the final page of the Sharing Project dialogue allows you to decide which resources to commit and which to ignore.
6. The selected resources will be added under the selected module name. After that you can manage your shared project using the CVS Perspective as described below.



Managing shared projects

Right-click on the project in the Project Explorer view, select **Team** option. You will see that this option differs for a checked out or shared project, if compared to a local project:



- The **Synchronize with Repository** option allows you to synchronize the changes in the selected source files with the repository (see Synchronize view below)
- The **Commit...** option is used to check in the changes made
- The **Update** option is used to update your local version of the project to the latest version available in the repository
- The **Create Patch** option opens a dialog which creates a patch containing all the changes made and specifies the location for the patch using the CVS Diff command
- The **Apply Patch** option is used to apply a patch previously created using the CVS Diff command
- The **Tag as Version** option is used to version a project whereas committing a project does not assign an understandable version tag to it.
- The **Branch** option allows you to create a branch and release resources to that branch when you are not yet ready to put your changes in the main development flow. It is also used for creating incremental patches.

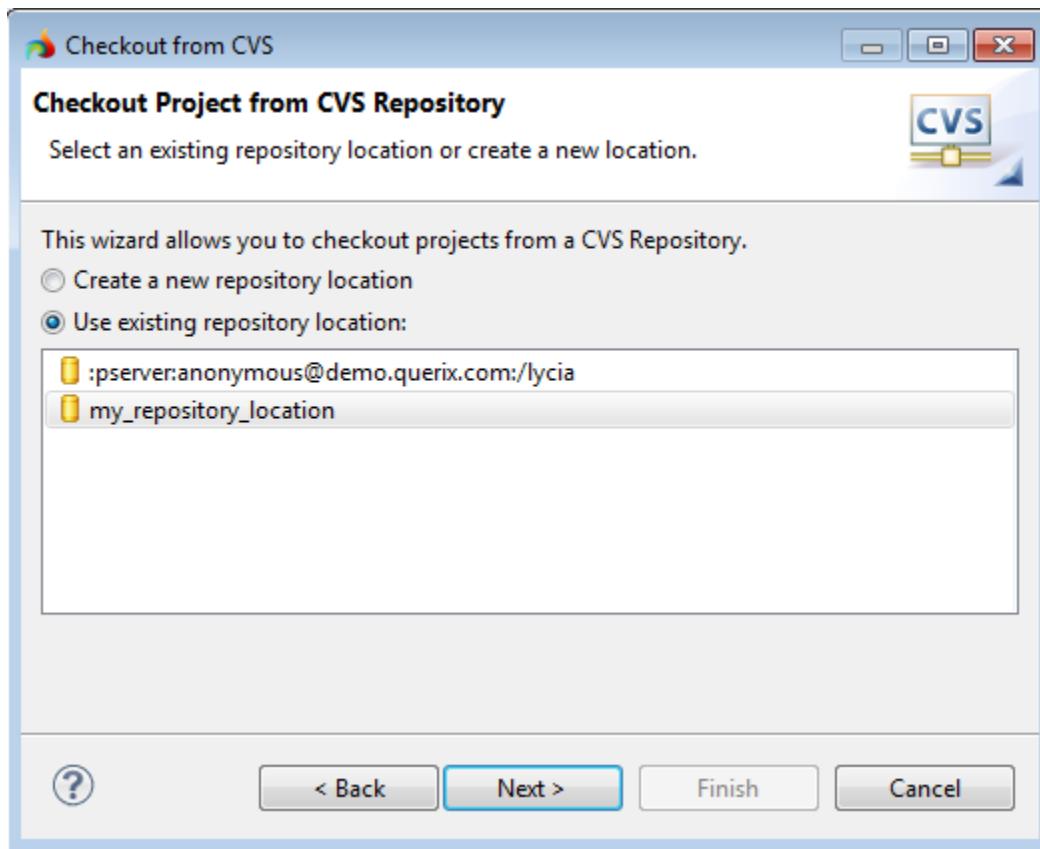


- The **Merge** option is used to merge changes between two states of a project in your workspace
- The **Add to Version Control** option

Checking out

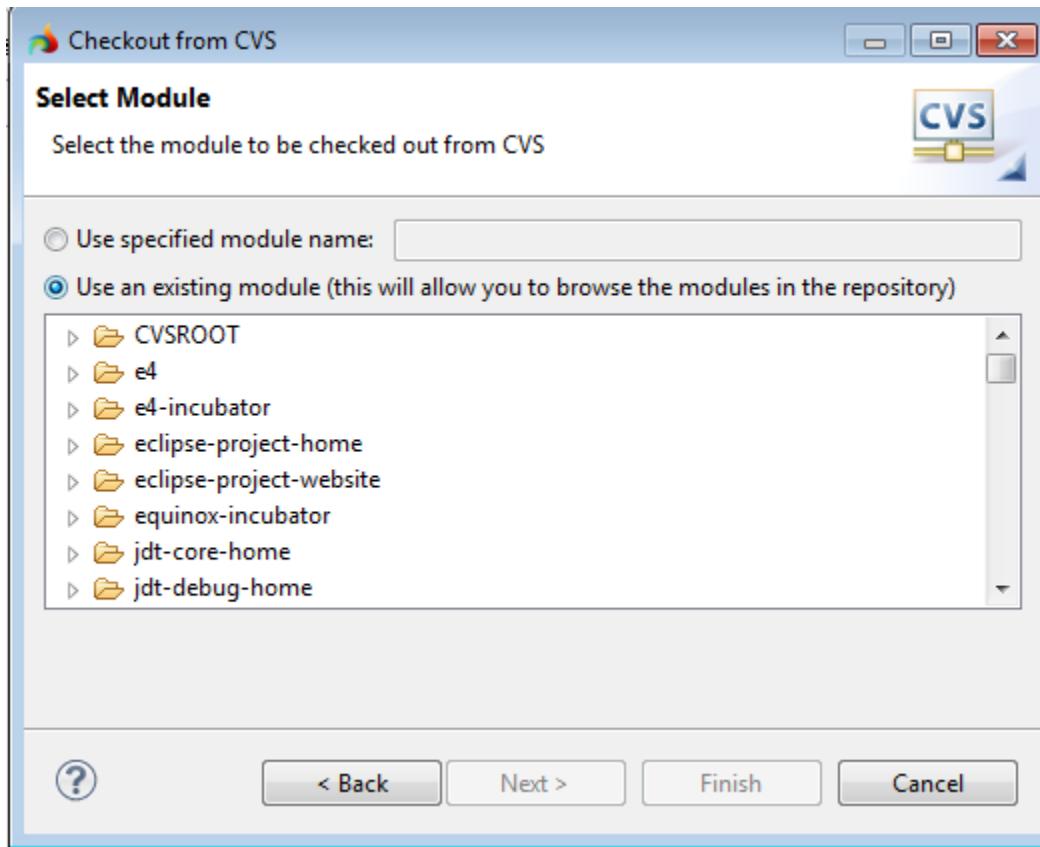
You can check out a project using an import option:

1. Select **File -> Import -> CVS -> Project form CVS** import option
2. Select a repository location or create a new repository location:





3. Choose the "Use an existing module" option:

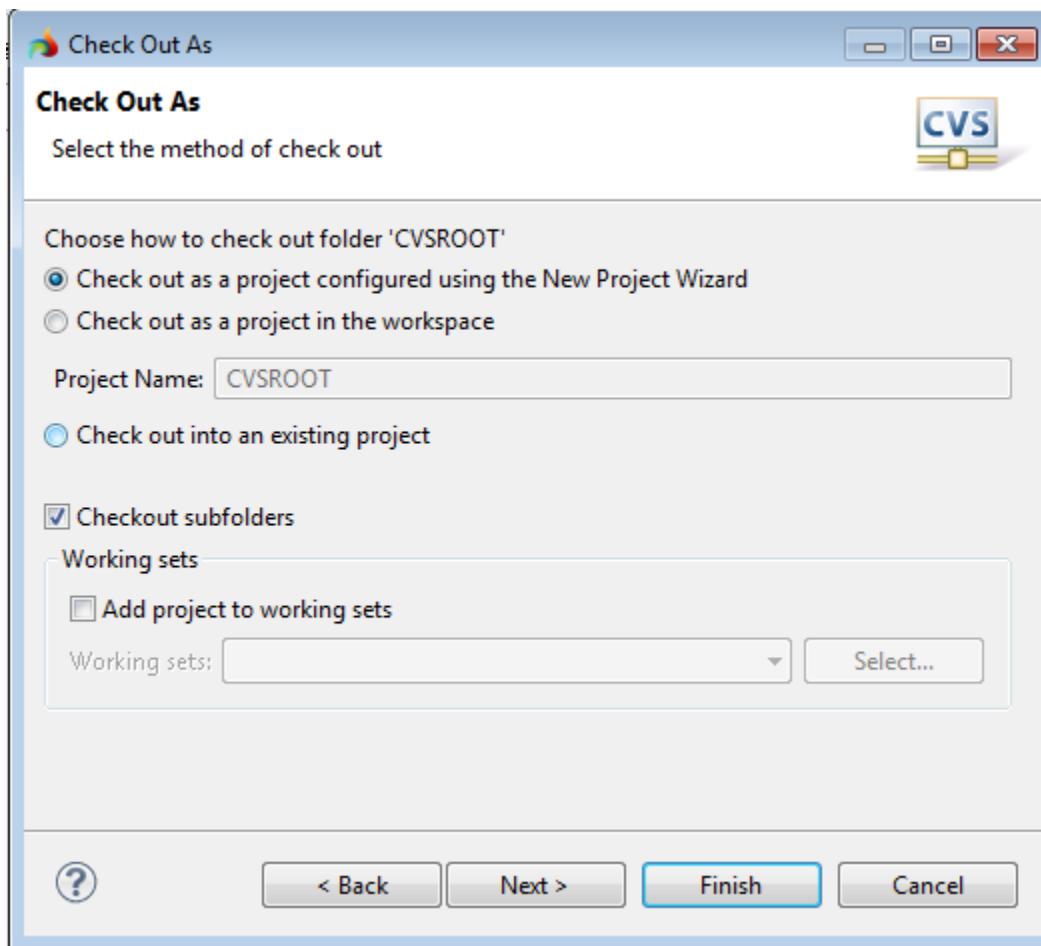


You can also specify the existing module name manually

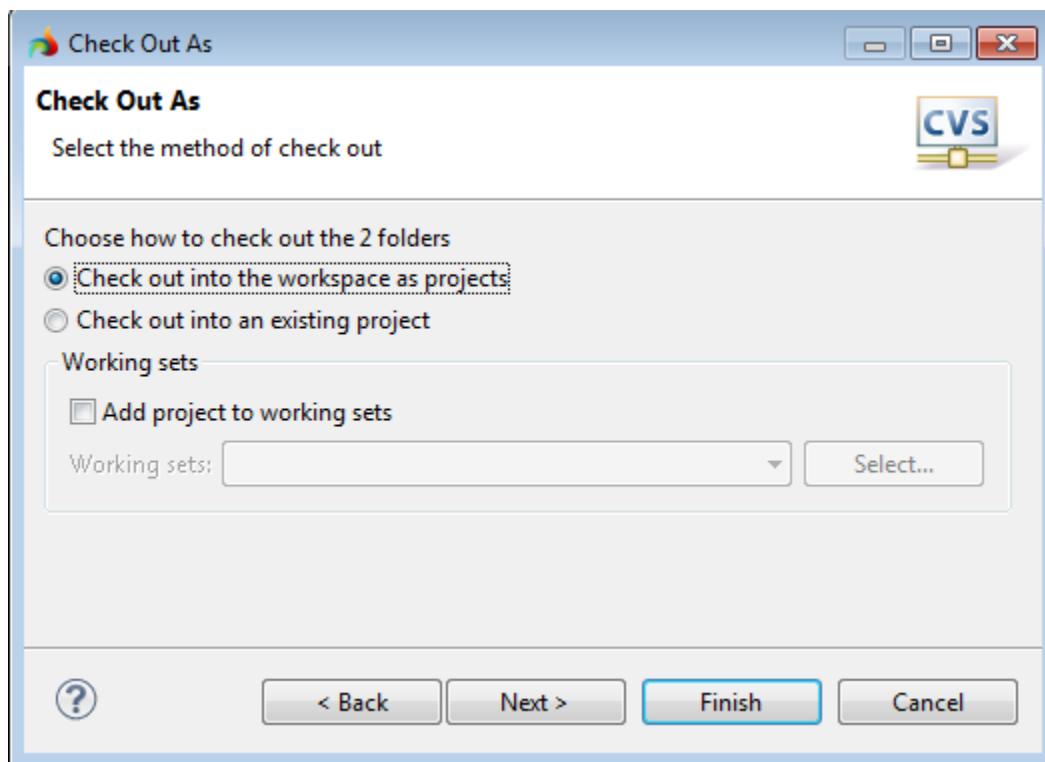
4. Select a module you want to import from the list of CVS modules. Please note that a module with such icon design: cannot be imported with the help of the Check Out As wizard. If you select it, the **Next** button will be disabled and the **Finish** button will import the content of the module as separate projects by default. You can expand such module and select one or more module folders inside it.



5. Press **Next** to open a Check Out As wizard:



Note that if you have selected more than one module, the Check Out As wizard will look differently:



Here you can either check the modules out as projects or check them out into an existing project.

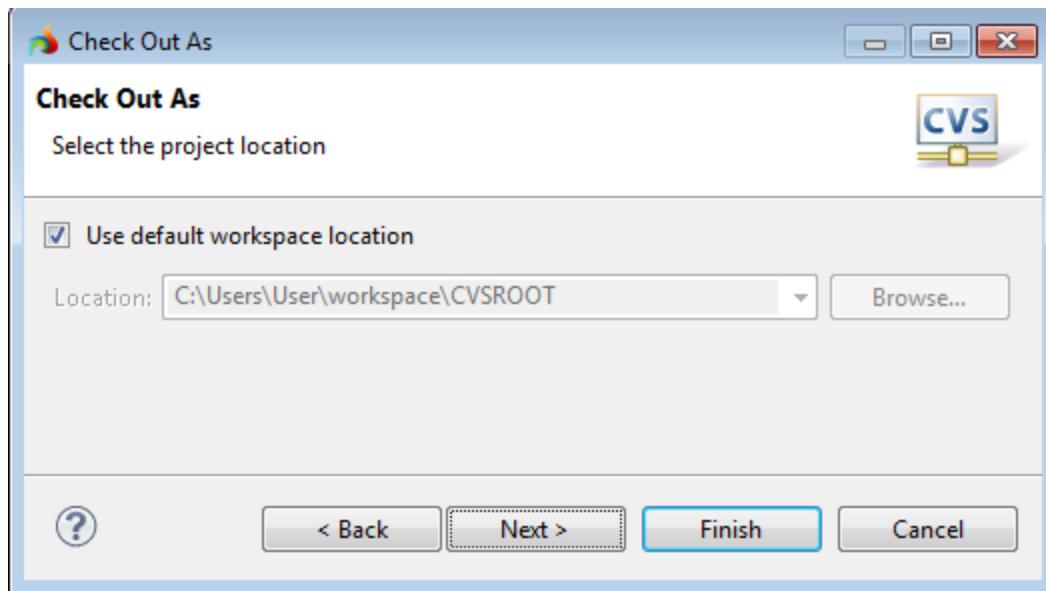
The Check Out As wizard

The Check Out As wizard allows you to configure the project before checking out. You can also include the checked out project into one of your working sets.

- The "Check out as a project configured using the New Project Wizard" option is available only if the .project file does not exist in the selected directory and you want to check out a module as an independent project. However, it is advisable to check out such modules into an existing project.
 - Select this option and press **Next**.
 - The New Project wizard will be opened. It will help you to create a new project containing the checked out resources. The process of project creation is described in the "Using LyciaStudio" chapter of this guide.
- The "Check out as a project in the workspace" option checks out the modules which contain the .project file.
 - If you select one module, you can type the name of the project in the field below.
 - If you select more than one project, they will be checked out with the module names used as project names.
 - Type the project name and press the **Next** button.

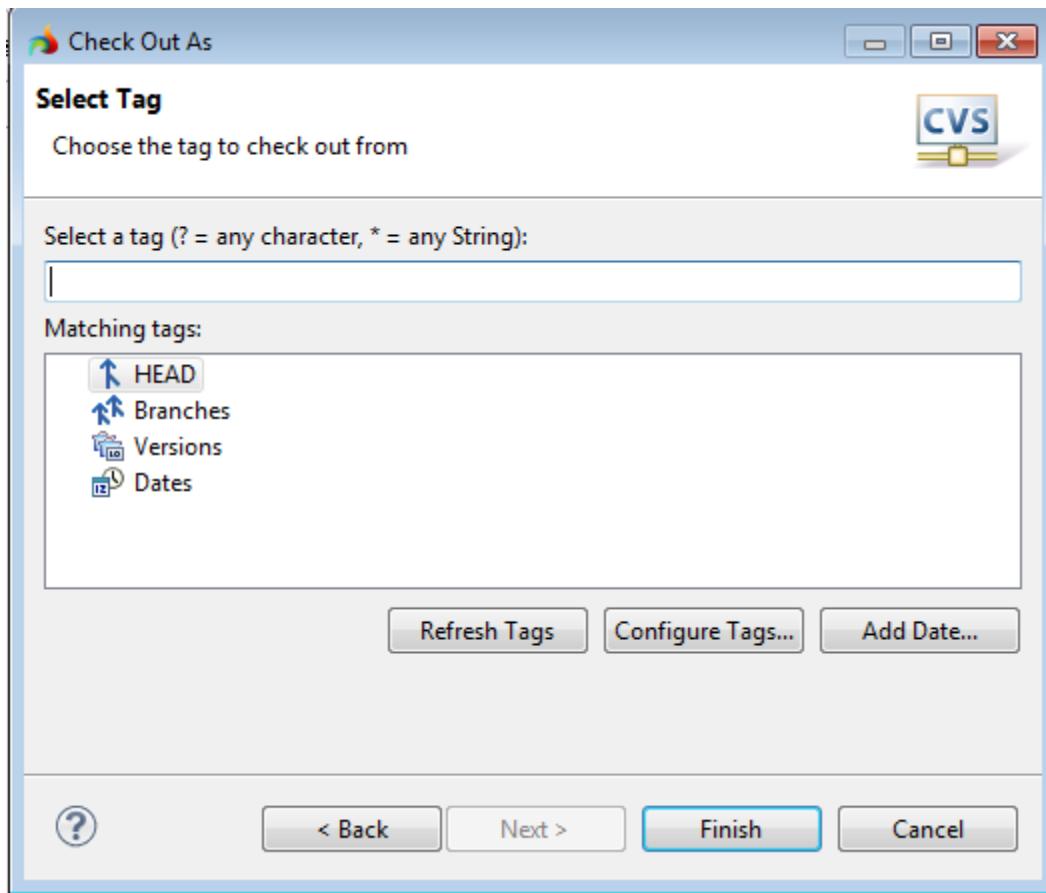


- Another page of the dialog will be opened, where you can specify the location to which you want to check out:

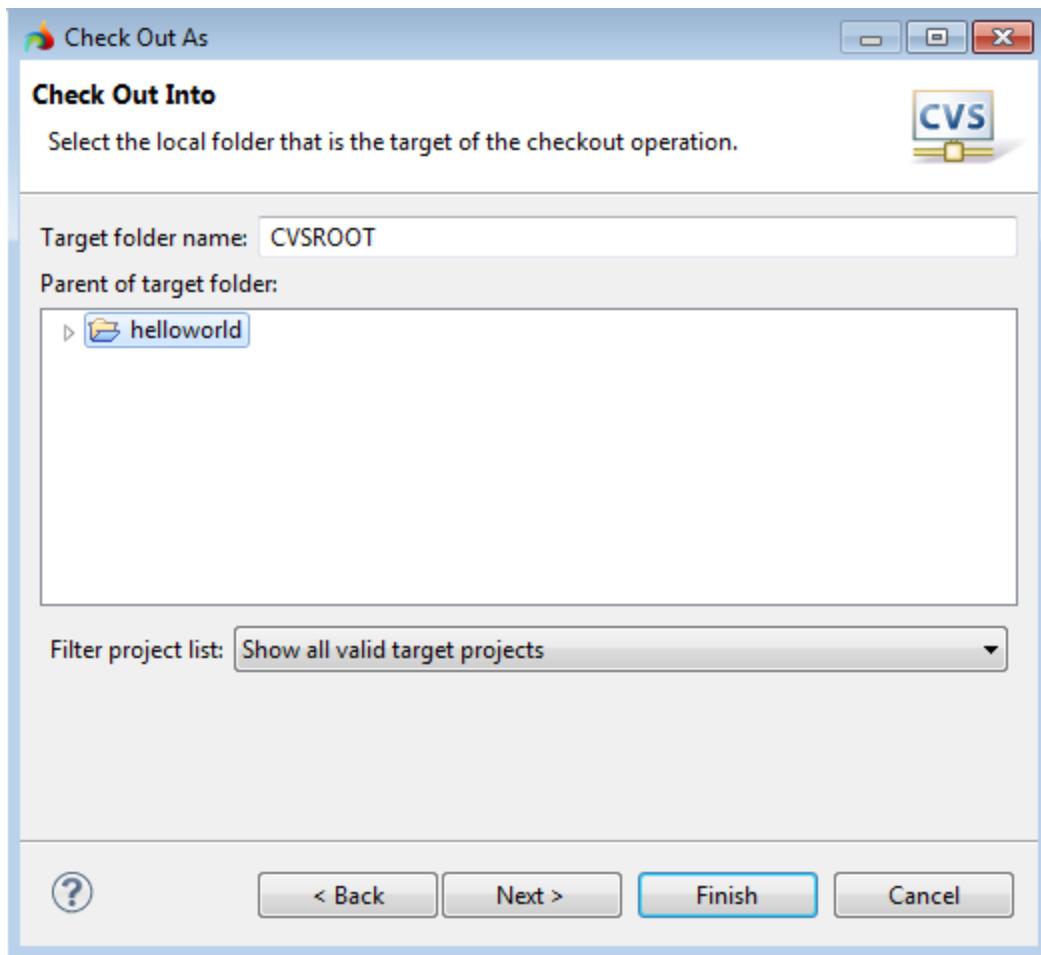




- Select the tags in the next page and press **Finish**. The modules will be checked out into the selected location.



- The "Check out into an existing project" option is used to check out resources into a project which is already present in the workspace. Select this option and press **Next**
 - Select the project into which you want to check out the resources from the list and press **Next**:



- Select the tags in the next page and press **Finish**. The modules will be checked out into the selected location.

You can also use the **Check Out As** option of the context menu of a resource which opens the Check Out As wizard:

1. Select a folder or file which you want to check out in the CVS Repositories view.
2. Right-click on it and select **Check Out As** option from the context menu.
3. The Check Out As wizard will be opened, where you can configure the import process as described above.

A project checked out with another CVS tool

If a project has been checked out with the help of another CVS tool, you do not have to check it out again with the help of the Studio. All you need to do to work on such a project is to import it with the help of the **File -> Import -> General -> Existing Project into Workspace** import option. (For details about importing projects, please see the "Using LyciaStudio" chapter of this guide).

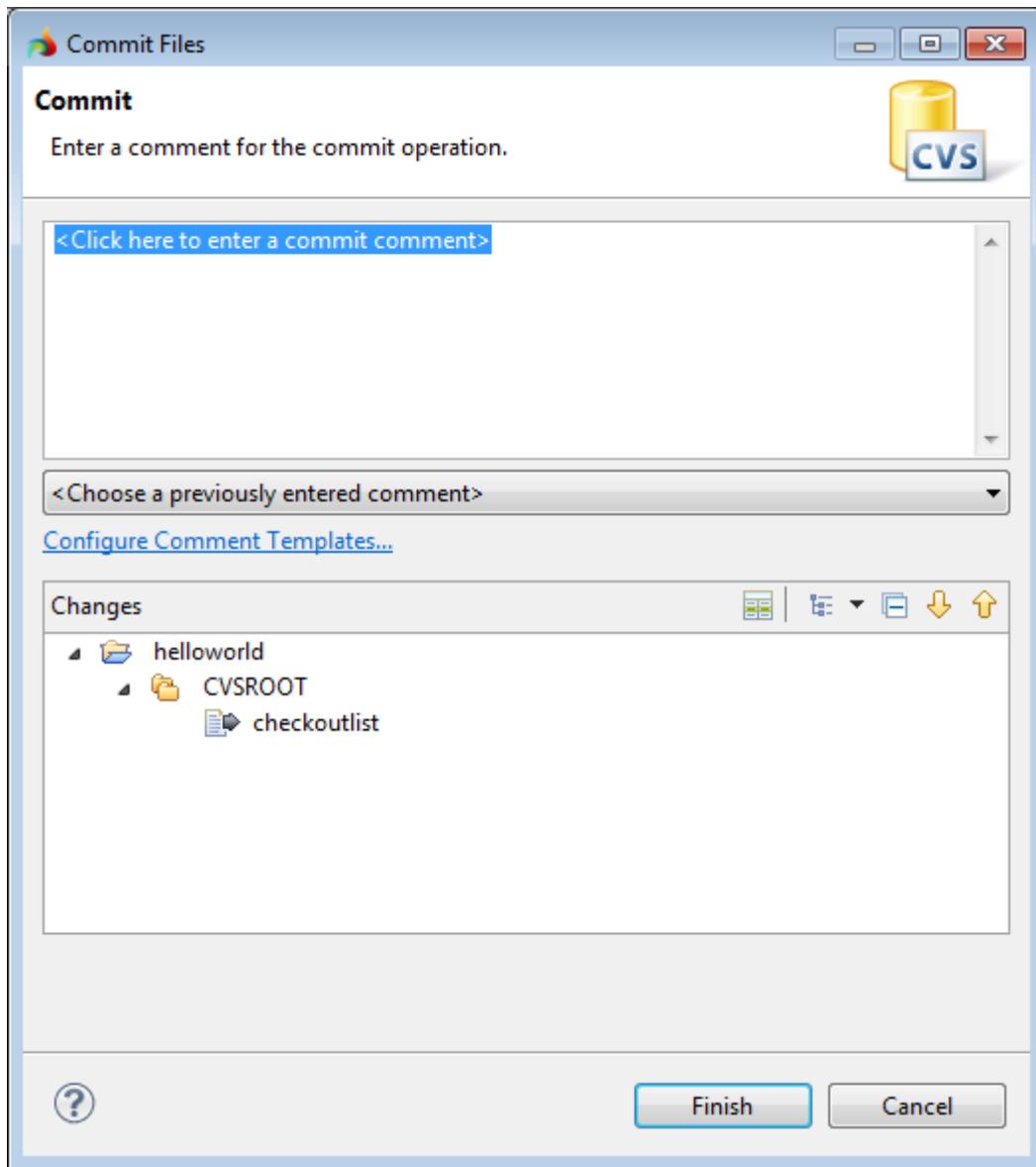


After you have imported a checked out project, you need to share it as described in the "Sharing with the help of CVS" section above. The last page of the share dialog will display the sharing information stored in the CVS folder of the project.

Committing

After you have made some changes in your local version of the project, you may need to commit these changes.

1. Right-click on the project in the Project Explorer view and select the **Team -> Commit...** option.
2. You will be presented with the Commit dialog:





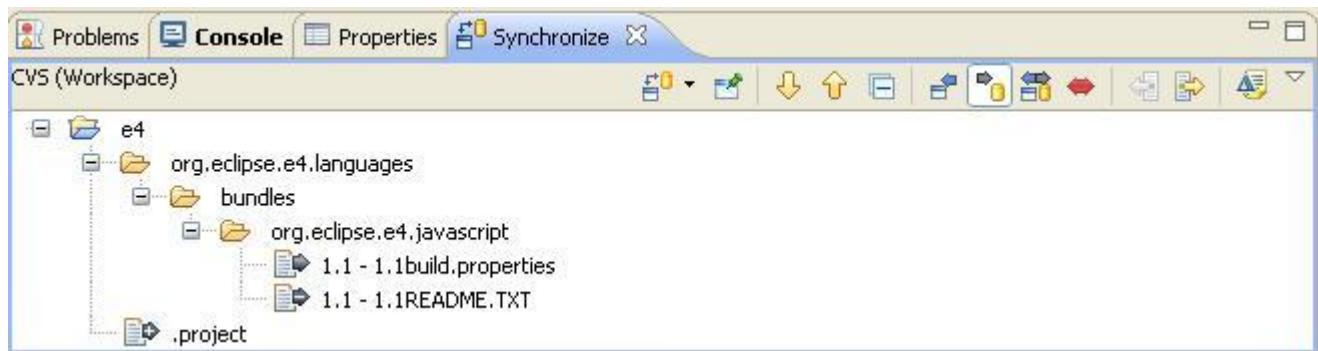
Here you can write a comment or choose a comment which has been entered before.
In the Changes window you can see the list of the files which have changes that will be committed.

3. Press **Finish** to commit the changes.

You can also use the **CVS -> Commit...** menu option, which is available in the CVS Repository perspective.

If you want to view and manipulate the outgoing changes before committing, use synchronizing option:

1. In the Project Explorer view select the resources you want to commit
2. Right-click on these resources and Select the **Team -> Synchronize with Repository** option from the context menu
3. The Synchronize view will be opened:



4. To see the outgoing changes before committing, press the button.
5. There are several tools used for updating, they are available from the context menu of any resource in the Synchronize view:
 - a. **Commit** option processes the selected outgoing and auto-manageable conflicting changes. Conflicts that cannot be automatically managed are not committed
 - b. **Override and Commit** option operates on conflicts. It will commit the local copy of the resource into the repository and remove any of the incoming changes.

You can manage the conflicting changes and not replace them with the local copies in the following way:

1. Select the **Open in Compare Editor** option from the context menu of the source file in the Synchronize view.
2. The local file and its repository copy will be opened in the Compare editor
3. Use the Compare editor to merge changes as described in the "Comparing" section of the "Using LyciaStudio" chapter of this guide.



4. Once completed, select the **Mark as Merged** option from the context menu of the conflict to indicate that you are done. This will change the conflict into an outgoing change.

If you want to add a new file to the project in the repository, use the **Team -> Add to Version Control** option of the context menu of this new file in the Project explorer view. After that you will need to commit the project. You can add any file into the ignore list by using the **Team -> Add to .cvsignore** option, and such files will be excluded from version control.

Updating

Whilst you are working on a project in your Studio, other members of your team may be committing changes to the copy of the project stored in the repository. To add these changes to your local copy, you may "update" your local copy of the project. It is also advisable to update a project before committing it.

To update your local copy, do the following:

1. Right-click in the Project Explorer view on the project you want to update and select the **Team -> Update** option
2. Your local copy will be updated with all the incoming changes (non-conflicting, conflicting that can be automatically managed and conflicting that cannot be automatically managed). By default, conflicting changes are merged using the CVS text mark-up.
3. You can set the update behaviour using the **Window -> Preferences -> Team -> CVS ->Update/Merge** preferences page
4. Your local version will be updated to the latest version stored in the repository

If you want to know what the incoming changes are before performing the update, use synchronizing option:

1. In the Project Explorer view, select the resources you want to update
2. Select **Team -> Synchronize with Repository** option from the resource context menu
3. The Synchronize view will be opened
4. To see the incoming changes before the update, press the button. You will see the changes that have been committed to the branch since you last updated.
5. There are several tools used for updating, they are available from the context menu of any resource in the Synchronize view:
 - a. **Update** option processes all the selected incoming and auto-manageable conflicting changes. Conflicts are merged depending on the Update/Merge preferences
 - b. **Override and Update** option operates on conflicts and will replace the local resources with the remote contents, if a conflict occurs

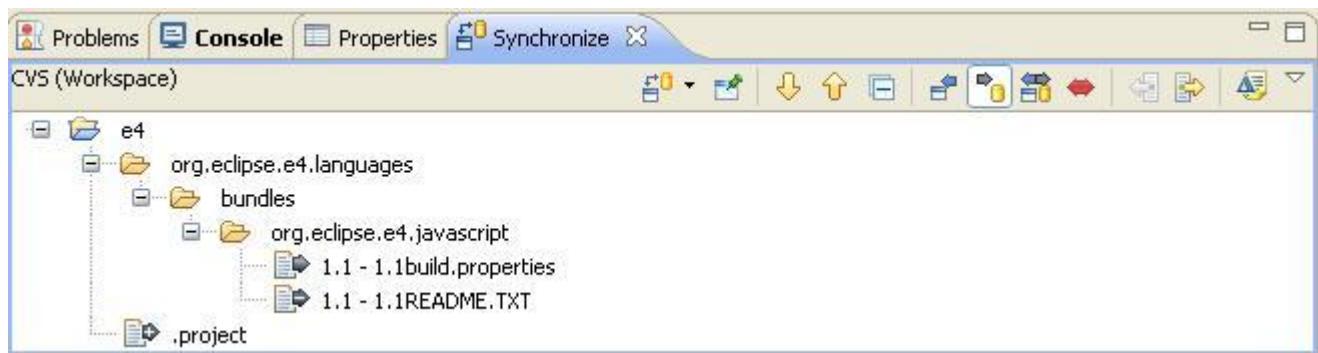
You can manage the conflicting changes and not replace them with the repository copies in the following way:



1. Select the **Open in Compare Editor** option from the context menu of the source file in the Synchronize view.
2. The local file and its repository copy will be opened in the Compare editor
3. Use the Compare editor to merge changes as described in the "Comparing" section of the "Using LyciaStudio" chapter of this guide.
4. Once completed, perform a **Mark as Merged** on the conflict to indicate that you are done. This will change the conflict into an incoming change.

Synchronize View

This view appears when **Team -> Synchronize with Repository** option is selected from the context menu of a resource. It allows you to inspect the differences between the local Studio resources and their remote counterparts as well as update resources in the Studio and commit them from the Studio to a repository.



The Synchronization view has 4 display modes:

1. - this button switches the view to Incoming display mode. The Synchronize view displays the changes incoming from the repository
2. - this button switches the view to Outgoing display mode. The Synchronize view displays the changes outgoing from your local copy
3. - this button switches the view to Incoming/Outgoing display mode. The Synchronize view displays both incoming and outgoing changes
4. - this button switches the view to Conflicts display mode. The Synchronize view displays only conflicts (resources modified both in the Workbench and in the repository).

The changes displayed in this view have the corresponding icons on their left which indicate the state of the resource compared to that of the repository:

- - A resource has been added to the repository. Use the **Update** option to transfer the resource to your workspace.



- ⓘ - A file has changed in the repository. Use the **Update** option to transfer the new file revision to your workspace.
- Ⓛ - A resource was deleted from the server. Use the **Update** option to delete your local resource.
- ⓘ - A file was added to your workspace and is not yet in the repository. **Adding** then **Committing** will transfer the new file to the repository.
- ⓘ - A file was changed locally. Use the **Commit** option to transfer the changes to the repository and create a new revision of the file.
- ⓘ - A resource that has been deleted in the local copy. Use the **Commit** option to cause the remote resource to be deleted.
- ⓘ - A resource has been added locally and remotely.
- ⓘ - A file has been changed locally and remotely. A manual or automatic merge will be required.
- ⓘ - A resource was deleted locally and remotely.



Web services

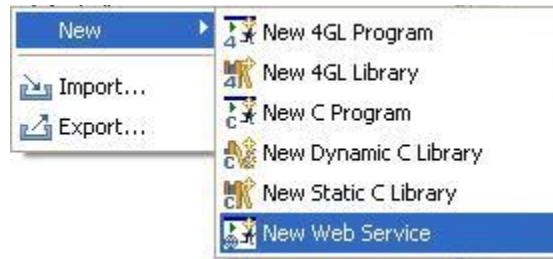
Lycia introduces some significant modifications in the command line tools used to create a web service and some minor changes in the process of their creation Using LyciaStudio. The deployment of the web services is performed practically in the same way as in 'Hydra' 4.x. For a thorough discussion on Web Services, please refer to the Web Services Getting Started Guide, available from our website.

	Note: To be able to compile the web services, you should have Microsoft Visual Studio installed on Windows
--	--

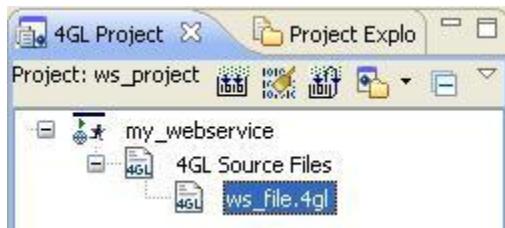
LyciaStudio Tools

In LyciaStudio, you cannot convert an existing 4GL program to a web service. You have to create a web service within a project in the same way as you create a 4GL program. To create a web service Using LyciaStudio interface, follow these steps:

1. Open a project in LyciaStudio
2. Right-click on the 4GL Project View and select the **New Web Service** option from the context menu:



3. Type the name for the web service and press **Finish**
4. The web service will appear in the 4GL view. Right click the web service and select **New -> 4GL Source File** option
5. Enter the name for the file and press **Finish**
6. Your web service, together with its source file, will be displayed in the 4GL Project View:



7. After you have added the contents to the file, build the web service in the same way as a usual 4GL program. This will create a .qar file which can then be deployed.

	<p>Note: To be able to compile the web services you should have Microsoft Visual Studio installed on Windows</p>
--	--

Command Line Tools

To create Web Services in Lycia using the command line environment you should use the modified command line tools. A web service can be created in 5 simple steps, they are:

1. Compile each 4GL source file into a .4o file using the **qfgl** command line tool.
2. Link all the necessary .4o files into one .4o library using the **qlink** command line tool.
3. Run the **wsmdc** command line tool against each 4gl file.
4. Run the **wslink** command line tool to create the web service binary executable (.exe).
5. Run the **wsact** command line tool to produce a compiled file of a web service with the .qar extension.

Create a .4gl file for the demonstration purpose and call it "my_ws". Copy the following into the "my_ws" file:

```
WEB FUNCTION add(a,b) RETURNS(c)

    DEFINE a,b,c INTEGER

    LET c = a + b

    RETURN c

END FUNCTION
```



The qfgl step

After you have created the sample file "my_ws", you need to compile it using the qfgl tool which is used to compile both files to be used in web services and in other 4GL applications.

To compile a file in order to be able to use it as a web service, you need to use special flags:

```
-m <metadirectory> |meta-data <metadirectory>  
-s | --web-service
```

The -m flag creates a special meta directory where meta-data are saved.

The -s flag instructs Lycia that the source file must be compiled as a web service compatible object file.

If we assume that "my_ws" file is stored in the directory "C:\work\Lycia\bin\progs" and you want your meta directory to be called "metadir", then you should call the following in Lycia command line environment:

```
C:\work\Lycia\bin\progs>qfgl -s -m metadir my_ws.4gl
```

As the result you will receive the "my_ws.4o" file, the "C:\work\Lycia\bin\progs\metadir" folder and the my_ws.xml file located in this folder.

You can check it with the help of the "dir" command:

```
Lycia Command Line  
C:\Program Files\Querix\Lycia\bin\progs>qfgl -s -m metadir my_ws.4gl  
C:\Program Files\Querix\Lycia\bin\progs>dir  
Volume in drive C has no label.  
Volume Serial Number is A015-16AF  
Directory of C:\Program Files\Querix\Lycia\bin\progs  
01/14/2010  11:19 PM    <DIR>          .  
01/14/2010  11:19 PM    <DIR>          ..  
01/14/2010  11:19 PM    <DIR>          metadir  
01/14/2010  08:47 PM           99 my_ws.4gl  
01/14/2010  11:19 PM          1,837 my_ws.4o  
                2 File(s)      1,936 bytes  
                3 Dir(s)  12,521,709,568 bytes free  
C:\Program Files\Querix\Lycia\bin\progs>cd metadir  
C:\Program Files\Querix\Lycia\bin\progs\metadir>dir  
Volume in drive C has no label.  
Volume Serial Number is A015-16AF  
Directory of C:\Program Files\Querix\Lycia\bin\progs\metadir  
01/14/2010  11:19 PM    <DIR>          .  
01/14/2010  11:19 PM    <DIR>          ..  
01/14/2010  11:19 PM           455 my_ws.xml  
                1 File(s)      455 bytes  
                2 Dir(s)  12,521,709,568 bytes free  
C:\Program Files\Querix\Lycia\bin\progs\metadir>_
```



We need to run this tool only one time and thus receive only one .xml file. If you have several files that need to be compiled in the web service mode, run this tool for each such file. Each time a corresponding .xml file is created in the meta directory.

The qlink step

If you have multiple .4o files that you have generated for your web service, you will need to link all of these together into one .4o file with the help of the qlink command line tool. In our example we use a single .4o file so this stage will be omitted.

The qlink tool does not have any additional flags and links the files in the same way as described in the qlink section of this guide.

The wsmdc step

The .xml file created by means of the qfgl tool needs to be passed to the wsmdc tool to create the associated java stub sources, c source stub and the associated meta files. In addition to the .xml file name you wish to compile, you also need to provide wsmdc with a base output directory where it can create these files as well as the location of the meta directory you wish to store the classnames.xml file into (normally it is the same as the metafile directory created by qfgl).

This tool supports the following flags:

```
-h | -?  
-p <output directory>  
-m <meta output directory> <meta source file>
```

We will direct the output of the java files created by this tool to the *wsmdc_out* directory by specifying the following command:

```
C:\work\Lycia\bin\progs>wsmdc -p wsmdc_out -m metadir  
metadir/my_ws.xml
```

This will create "classnames.xml", "WSMetaMeta.xml", and "my_ws.c" files in the *metadir* folder and the "wsmdc_out\com\querix" directory with the "my_ws.java" file:



```
Lycia Command Line
Setting environment for using Microsoft Visual Studio 2008 x86 tools.

C:\Program Files\Querix\Lycia\bin>cd progs\metadir
C:\Program Files\Querix\Lycia\bin\progs\metadir>dir
 Volume in drive C has no label.
 Volume Serial Number is A015-16AF

 Directory of C:\Program Files\Querix\Lycia\bin\progs\metadir

01/15/2010  01:54 AM    <DIR>      .
01/15/2010  01:54 AM    <DIR>      ..
01/15/2010  01:54 AM           124 classnames.xml
01/15/2010  01:54 AM          1,779 my_ws.c
01/15/2010  01:47 AM          457 my_ws.xml
01/15/2010  01:54 AM          86 WSMetaMeta.xml
                           4 File(s)     2,446 bytes
                           2 Dir(s)   12,414,734,336 bytes free

C:\Program Files\Querix\Lycia\bin\progs\metadir>
```



Note: To execute this step and the next steps you need to have Microsoft Visual Studio installed on Windows. No additional software needs to be installed on UNIX/Linux to compile the web services.

The wslink step

This stage is used to create the web service binary executable. The wslink command tool can accept the following flags:

```
-? | --usage

-V | --version

-v | --verbose

-p <output dir>| --basedir <output dir>

-m <meta output dir>| --metadir <meta output dir>

-o <output filename> | --output <output filename>
```

The **-metadir** flag specifies the meta directory set by qf1 and the **-o** flag specifies the name of the output binary file. The wslink command must be run for the pcode file. In our example, it is the "my_ws.4o" file. If you have more than one source files, it must be the file created by means of the qlink tool.



In our example, we will use the current working directory as the output directory and create the "my_ws.exe" binary file:

```
C:\work\Lycia\bin\progs>wslink -p . -m metadir -o my_ws.exe my_ws.4o
```

The "my_ws.exe" file will be created in the current directory. You will need to have the C environment configured correctly to be able to use wslink. This should be automatically setup as part of the Lycia command line environment unless there was no C compiler detected on Lycia install.

The wsact step

The final stage of web service compilation is wsact. This tool takes our files created in the previous steps and combines them into one .qar file. The wsact tool has the following command line flags:

```
-? | --usage  
  
-V | --version  
  
-v | --verbose  
  
-o <output filename> | --output <output filename>  
  
-p <output directory> | --basedir <output directory>  
  
-m <metafile directory> | --metadir <metafile directory>  
  
-w <ws binary file> | --wsbinary <ws binary file>  
  
-l <p-code binary file> | --PCODE <p-code binary file>
```

The <ws binary file> is the .exe file created by means of wslink. The <p-code binary file> is the .4o file created after linking by means of qlink tool. The <metafile directory> is the meta directory created at the qfgl stage.

In our example we will use the current working directory as the output directory, the qar extension is required for the output file:

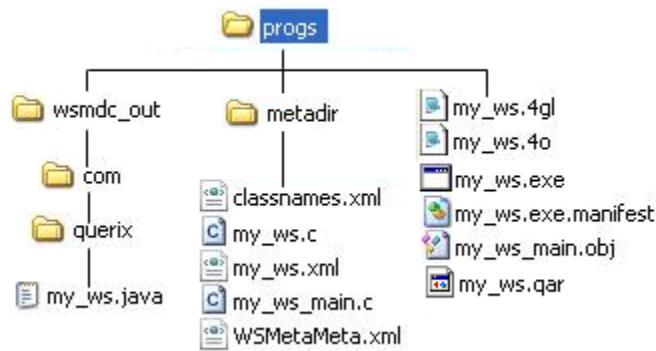
```
C:\work\Lycia\bin\progs>wsact -o my_ws.qar -p . -m metadir -w  
my_ws.exe -l my_ws.4o
```

The "my_ws.qar" file will be created in the current output directory. You can deploy this file using Querix deployment tools



Output Directory Structure

For the example above, the output directory structure for 4GL web services in Lycia after a web service has been created as follows:



- The "progs" folder is the base output directory
- The "metadir" folder is the output directory for meta-files
- The "wsmdc_out" folder is the output directory for java files

The names for both < output directory for java files > and < output directory for meta-files > must be unique during each LyciaStudio session.

If you create a web service using LyciaStudio, the web service files will be located in the project folder within the workspace. All the files generated by the web service compiler are placed in the output folder of the project. Auxiliary files such as .xml are deleted when compiling through the Studio after they have been used.

Web Services Deployment

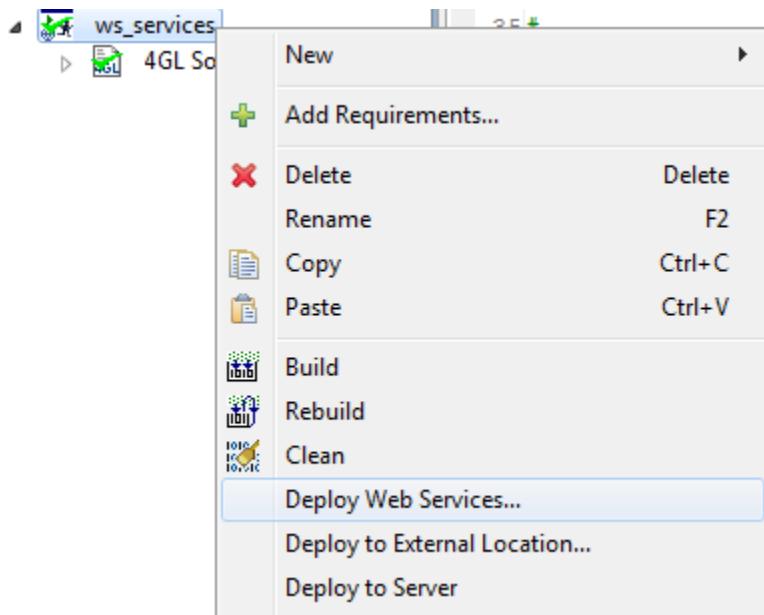
Web services deployment in LyciaStudio is described in details in "Lycia4GL Web Services" guide. This document is shipped together with Lycia and can be found in the Documentation folder.

To make a web service accessible/available for other applications to use, you need to publish this new web service to your web service application server by using the web service deployment tool.



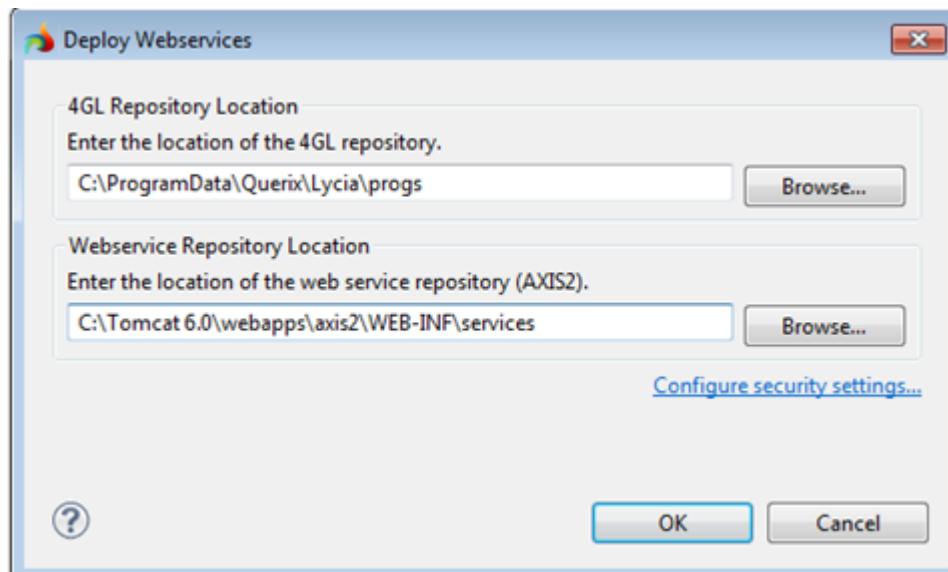
The deployment tool

In LyciaStudio, you can deploy a compiled web service by right-clicking on a compiled web service in the 4GL Project View and selecting the **Deploy Web Service** option from the context menu:



You cannot deploy a web service using the command line.

When you run the 'Web Services Deployment' tool, a dialog box will open. Here, you have to input 4GL and Web Service repository locations. Input the paths to the dialog box and press "Ok"

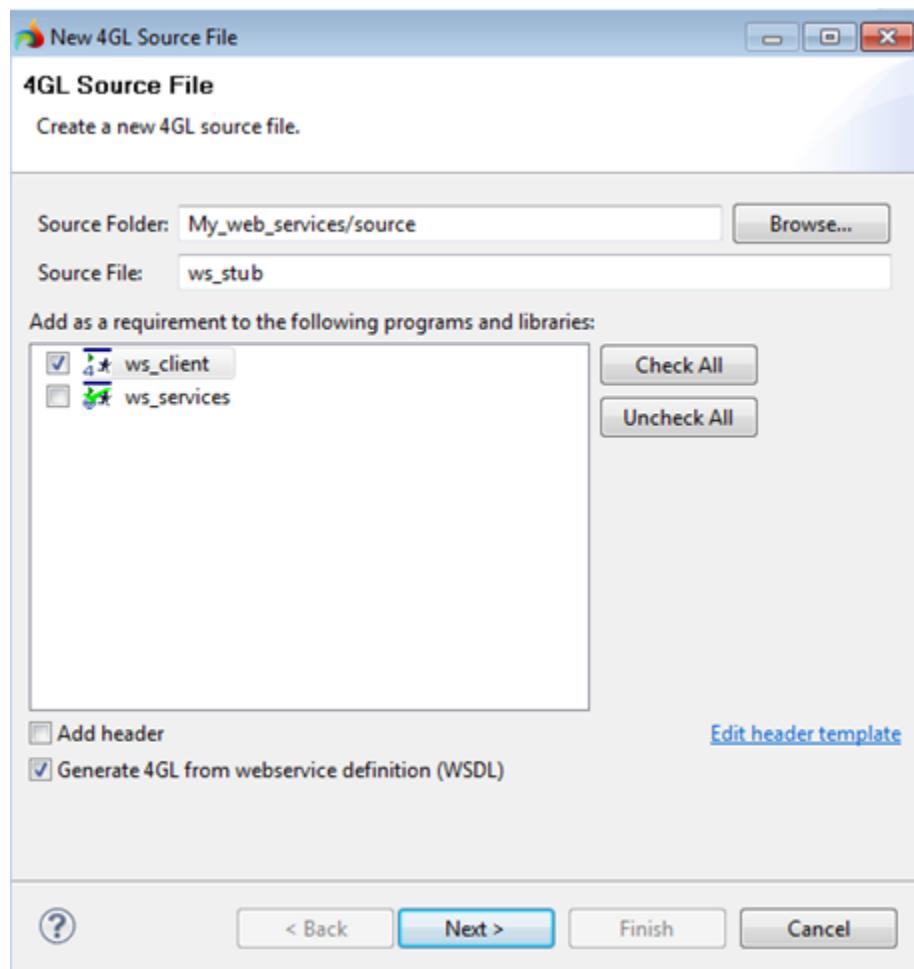


After you perform these operations, you can see your web service appear on the web server and you can access the WSDL document from the axis home page -> services (for example, <http://localhost:9090/axis2/services/listServices>)

Web service client creation

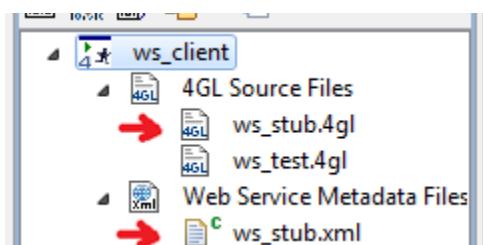
To create a web service client in Lycia, you have to add a stub and a meta data file to a 4GL program. The stub file will contain the descriptions of the functions available from the web service. You can reference the web functions using the function names listed in the stub file.

To run a tool which creates the stub file and the meta file, use the New 4GL source dialog. Enter the name for the new file, and select 'Generate 4GLfrom webservice definition' option at the bottom of the window. Then, click **Next**:



On the following page, you should enter the URL of the WSDL file which you want to use and click **Parse**. After that, chose the web services, functions, and ports you want your program to use, and click **Finish**:

When you click **Finish**, the new file creation tool will be closed and your application will get two new files - a stub file with the .4gl extension and an .xml meta data file.

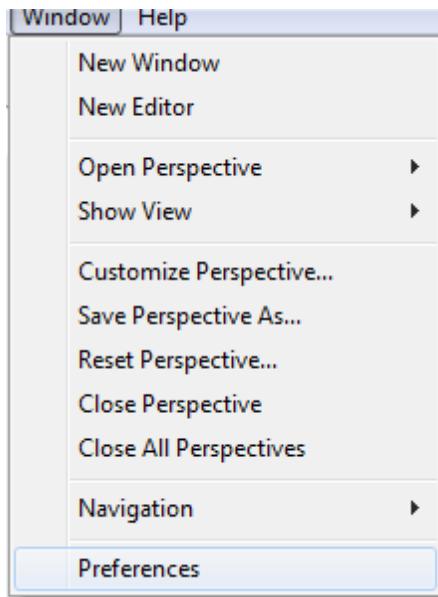


You can open the stub file to see the names by which the available web functions can be referenced and their structure. Now, you can reference these web functions in any of your program modules.

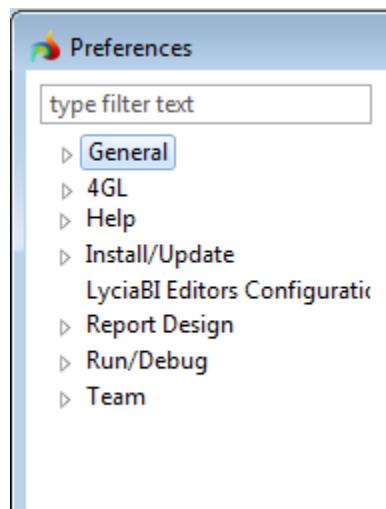


Appendix A: Preferences

The Preferences menu contains many settings which help to adjust the appearance and functions of the LyciaStudio, running preferences, debugging preferences, the team preferences, special 4GL preferences and so on. This menu can be called by selecting the **Window -> Preferences** option from the main menu, as shown below:



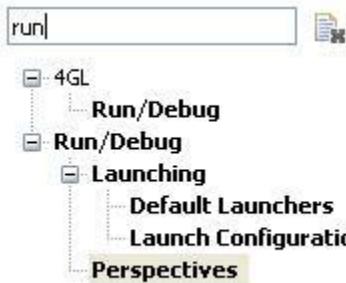
The Preferences dialog will then be opened. The Preferences dialog window is divided into two panes. The left-hand pane of the window contains the list of titles which are used to select the options to adjust. The titles can be expanded if the plus button is pressed, and the sub-titles will be displayed:



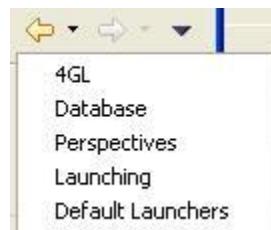


In this document those options are described which influence Lycia performance. It is possible that the actual list of the options may vary, because some of them are inherited from the original Eclipse IDE.

The field above the list of titles is a filter which can be used to limit the list of available titles. If a keyword is typed into this field and the Enter key pressed, only those titles which relate to this keyword are displayed:



The right pane of this window contains the settings for the selected option which can be adjusted. This part also contains the arrow buttons (and) which enable you to navigate through previously viewed pages. If you select the drop down arrow next to the left arrow button, a list of recently viewed pages will be displayed, to jump to the recently viewed page from the list, click on it.



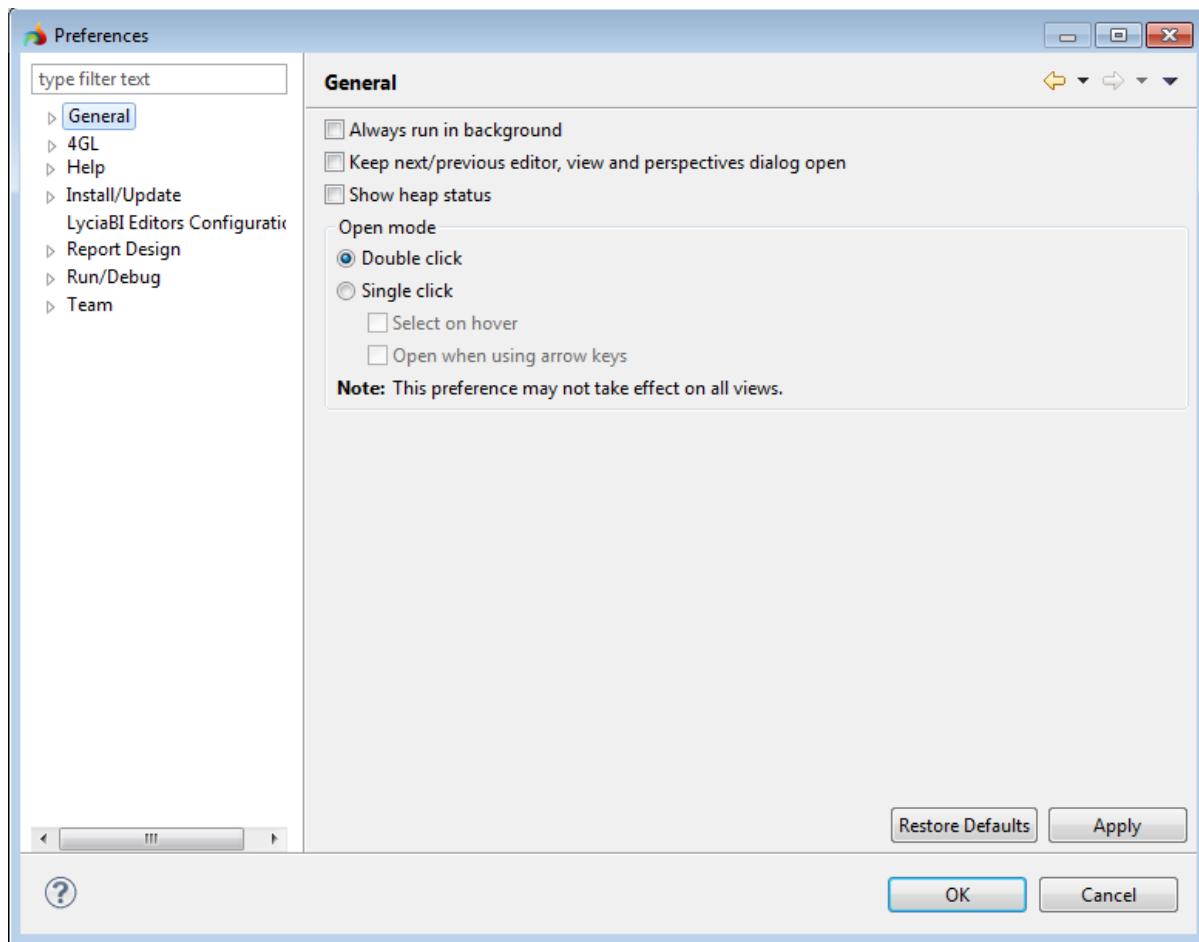
When you change the settings in the Preferences dialog, LyciaStudio may need to restart to apply these changes. All of the preferences pages include a **Restore Defaults** button, which returns the default values of the currently selected page to their original state.



General Preferences

The General preferences encompass settings which will affect the look and feel of the Studio. It deals with adjusting user perspectives, views, editors, keys, toolbars and suchlike.

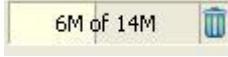
Below is what the General preferences page looks like:



The default values for the options listed in the table below correspond to the values selected in the picture above.

Option	Description
Always run in background	Select to perform long running operations in the background without blocking you from doing other work.
Keep next/previous editor, view and perspectives dialog open	Select to leave the editor, view and perspective dialogs open when their activation key is let go. By default, the dialog closes as soon as the key combination is released activating the option which has been highlighted.

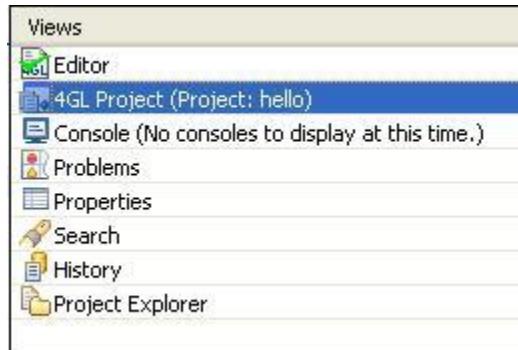


Show Heap Status	Select to display an indicator showing information about the current heap usage in the bottom left corner of the Studio. 
Open mode...	This option allows to select an open mode: <ul style="list-style-type: none">Double click – double clicking on a source file will open it in the appropriate editor, whereas single clicking on a source file will select it.Single click – this option has two modes:<ul style="list-style-type: none">Select on hover – select a source file by hovering the mouse cursor over itOpen when using arrow keys – a file is opened when it is selected by means of an arrow key <p>Note: Depending on the current view, selecting and opening a source file may have different behaviour.</p>

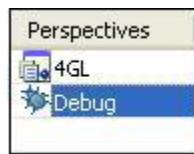
The editor dialog is invoked by pressing the Ctrl+F6 key combination; it allows you to choose a file in the current editor from the list of opened files:



The view dialog is invoked by pressing the Ctrl+F7 key combination, it allows you to choose a view from all of the open views within the current perspective and make it active:



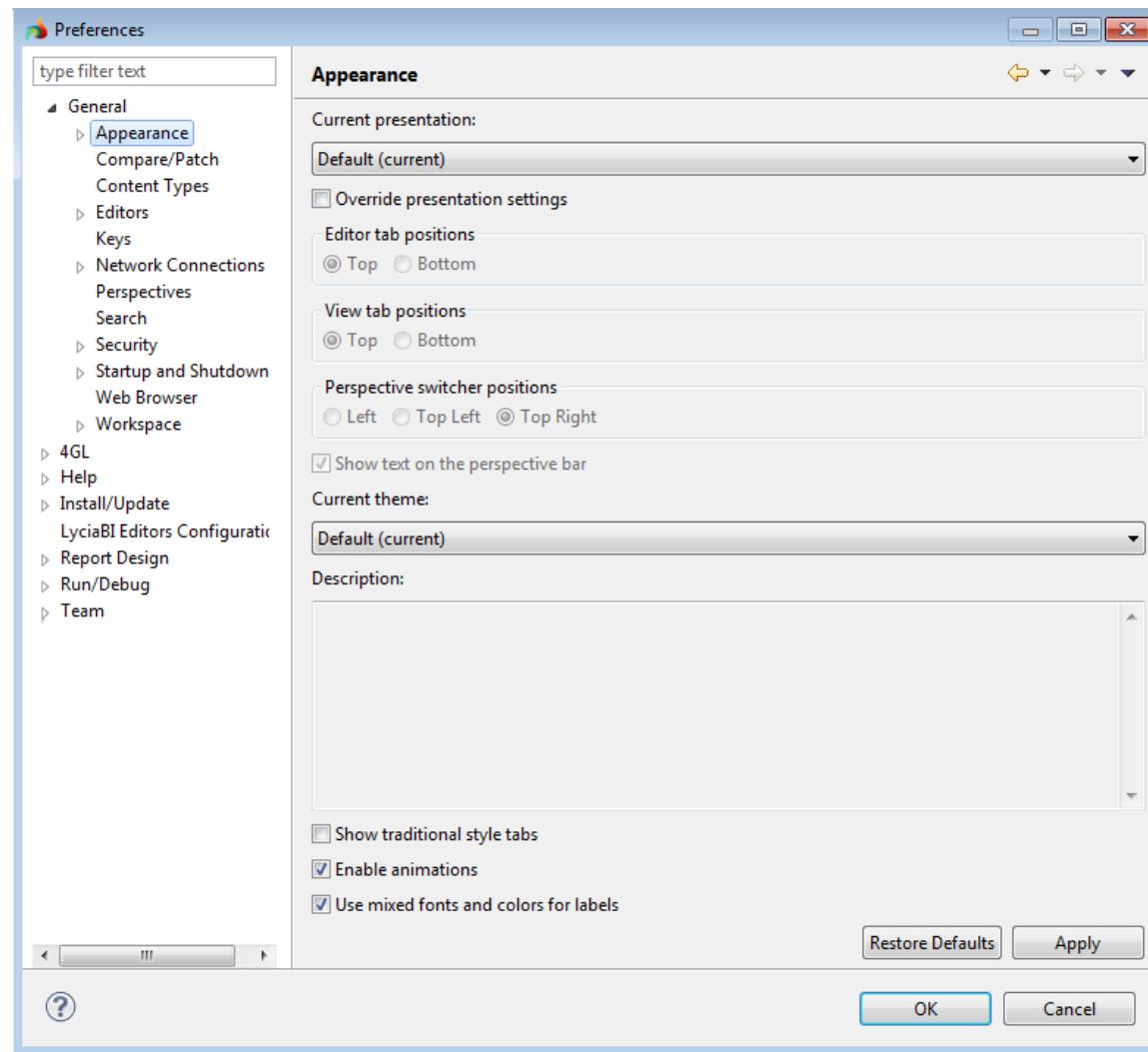
The perspective dialog is invoked by pressing the Ctrl+F8 key combination, it allows you to choose a perspective among all the open perspectives and make it the current perspective:



The General section comprises a number of options which can be viewed by pressing the plus button next to the General title in the left pane of the dialog window. The settings for these options are discussed below.

Appearance Preferences

The settings of the Appearance Preferences menu are used to adjust the look and layout of views in the Studio. The Appearance Preferences settings are included in the General Preferences block.



The default values for the options listed in the table below correspond to the values selected in the picture above.



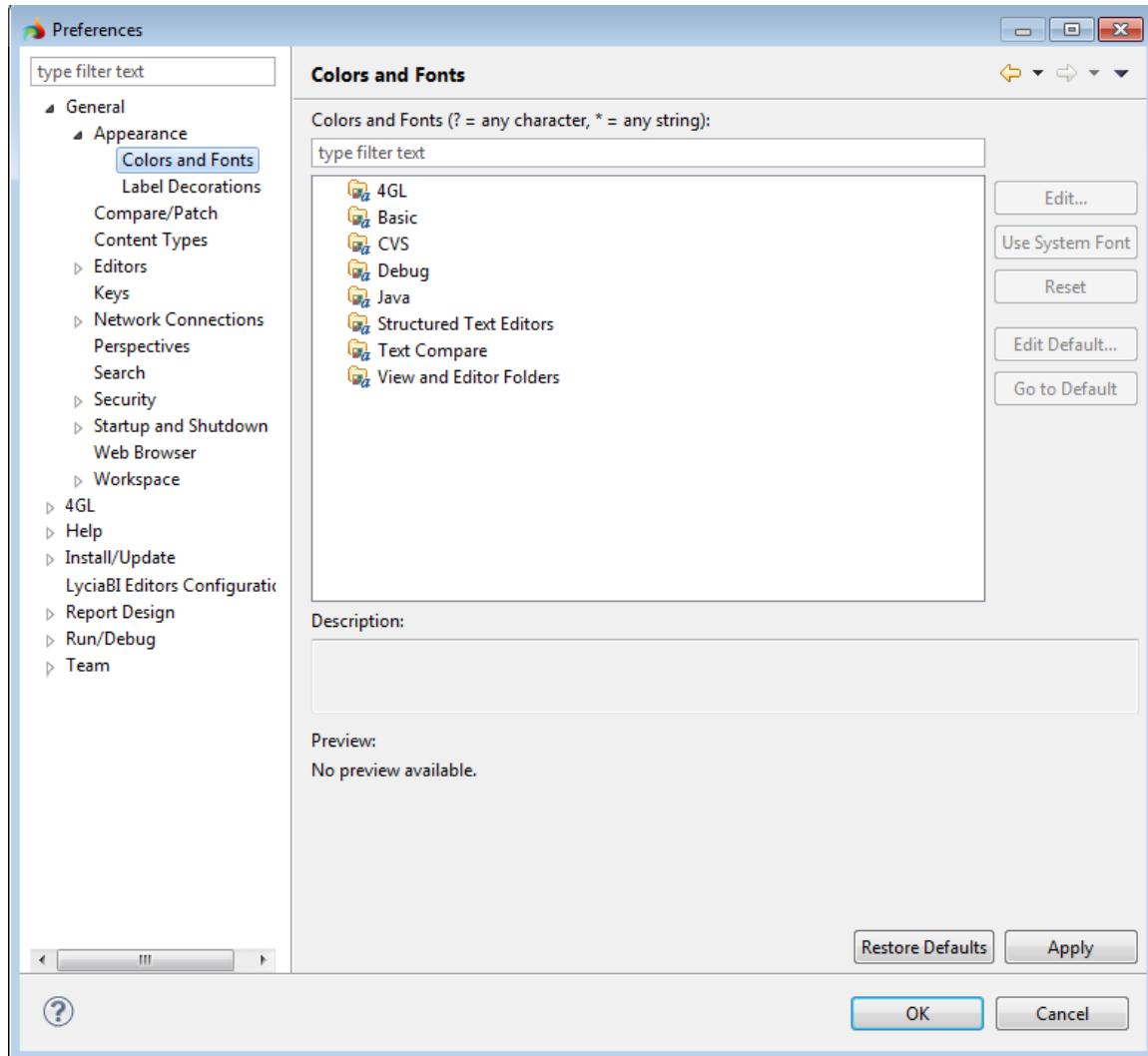
Option	Description
Current presentation	Specifies the currently active look-and-feel of the Studio
Override presentation settings	This option is used to locally override the default settings of the Studio
Editor tab positions	This option positions the tabs of the built-in editors (either at the top or at the bottom)
View tab positions	This option positions the tabs of the views (either at the top or at the bottom)
Perspective switcher positions	This option positions the perspective switcher bar relative to the Studio layout
Show text on perspective bar	This option turns on and off the text with the name of the perspective on the perspective bar
Current theme	This option specifies the currently used font and colour set.
Show traditional style tabs	Select to display square tabs instead of rounded ones

The Appearance Preferences option has two sub-options which can be viewed if the plus button next to it is pressed. These are the Colours and Fonts page and the Label decorations page.



Colours and Fonts Preferences

The colours and fonts used for the built-in editors are set on this page:



The tree is used to navigate amongst and show a short preview of the various colours and fonts. A label of a font represents its face, although the size is not reflected in the label. A label of a colour is used to preview the colour.



Some categories provide a more detailed preview of their contributions (e.g. the View and Editor Folders category) which is shown below the description, if available:

The screenshot shows the 'Colors and Fonts' preferences dialog. The left pane displays a tree view of color and font settings categorized by application. Under 'View and Editor Folders', there are numerous options for different parts like 'Active (non-focus) part background begin' and 'Inactive part foreground'. Below the tree view are two font samples labeled 'Part title font' and 'View message font'. The right pane contains buttons for 'Edit...', 'Use System Font', 'Reset', 'Edit Default...', and 'Go to Default'. The bottom section includes a 'Description:' label with a large empty text area, a 'Preview:' label with a window showing sample text, and buttons for 'Restore Defaults', 'Apply', 'OK', and 'Cancel'.

Font settings can be changed by selecting a font from the list and clicking the **Use System Font** button (to set the operating system font as the font for the option) or the **Change...** button (to open a font selection dialog).



Colour settings can be changed by selecting the colour in the tree and by clicking the colour button to the right from the tree.

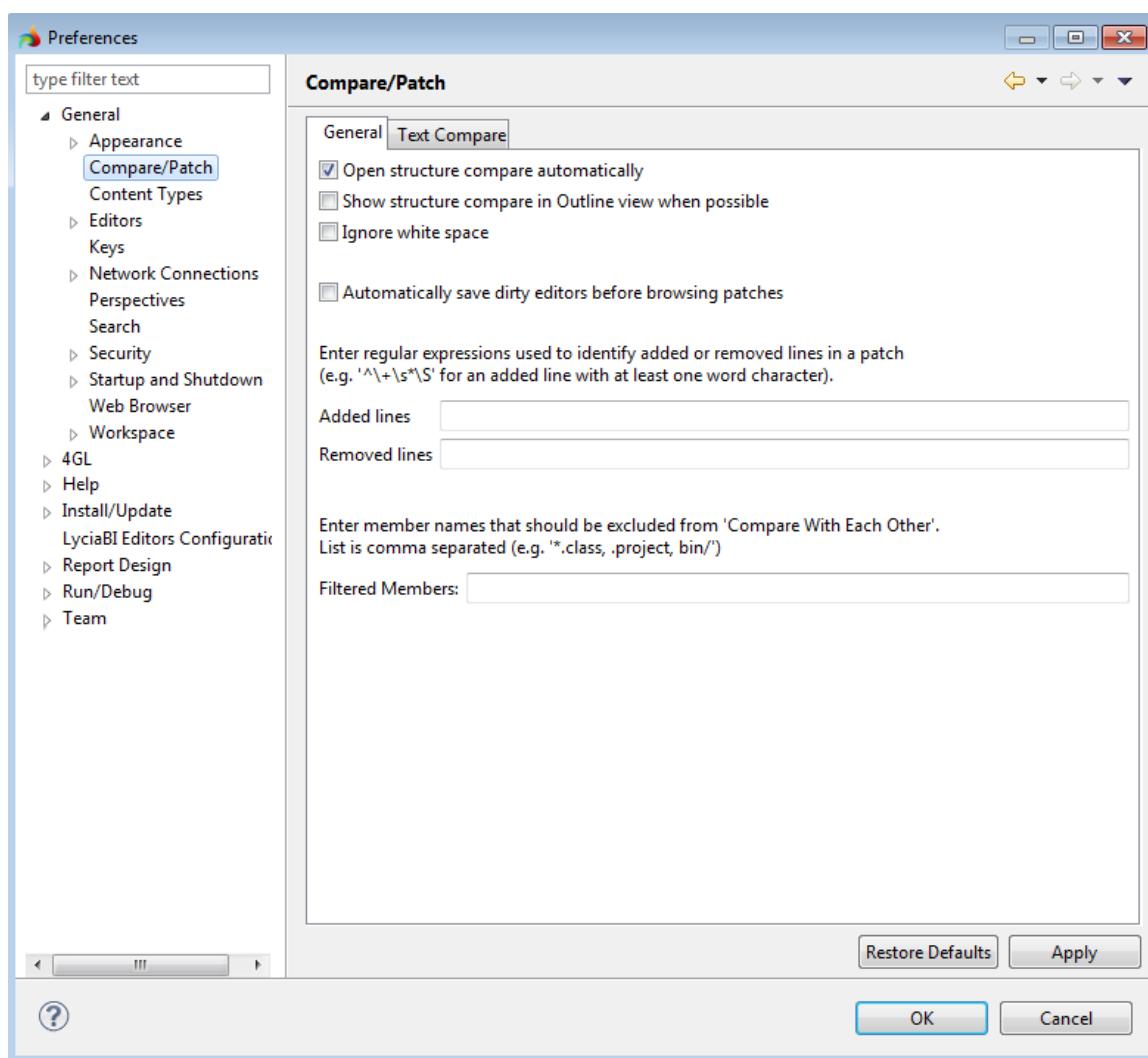
The **Reset** button returns the default value of the selected font or colour when pressed.

Label Decorations Preferences

The Label Decorations page allows you to specify which decorators should be displayed on items within the Studio. Decorators give you more information about an item. This page provides description for each decoration, when this decoration is selected.

Compare/Patch Preferences

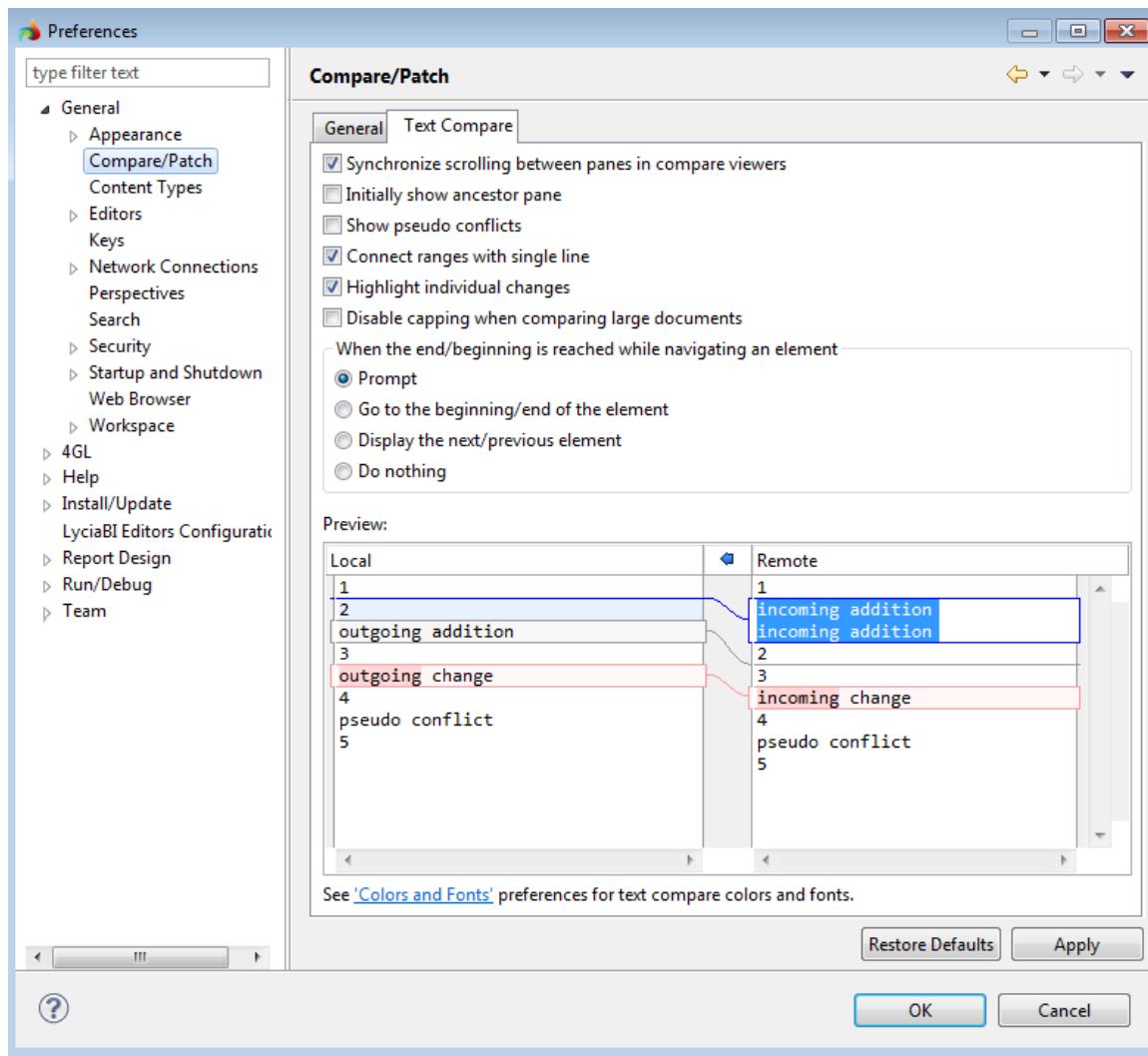
This preference page is used to set preferences for the compare, merge, and patch functionality of the Studio. It contains two tabs - the General tab and the Text Compare tab.





The default values for the options of the General tab correspond to the values selected in the picture above.

Option	Description
Open structure compare automatically	Deselect if you want to see only differences in content and do not want to see the structural differences
Show additional compare information in the status line	Select if you want to see the additional information about the compare in the status line
Ignore white space	Select to ignore the whitespaces when comparing
Automatically save dirty editors before browsing patches	Select if you want any unsaved changes to be automatically saved before a patch is applied
Added/Removed lines	These two fields define which lines should be counted as added and removed lines when applying a patch, these options are based on regular expressions
Filtered Members	This field may contain a comma separated list of members which should be excluded when executing the "Compare With Each Other" option



The default values for the options of the Text Compare tab correspond to the values selected in the picture above.

Option	Description
Synchronize scrolling between panes in compare viewers	Select to make the two panes with the compared code scroll "lock scroll" along with one another in order to keep identical and corresponding portions of the code in each pane side-by-side.
Initially show ancestor pane	If the two versions of a source file have derived from the same source file, such source file is called the common ancestor. Select to display the common ancestor on a separate ancestor pane at the beginning of each comparison.

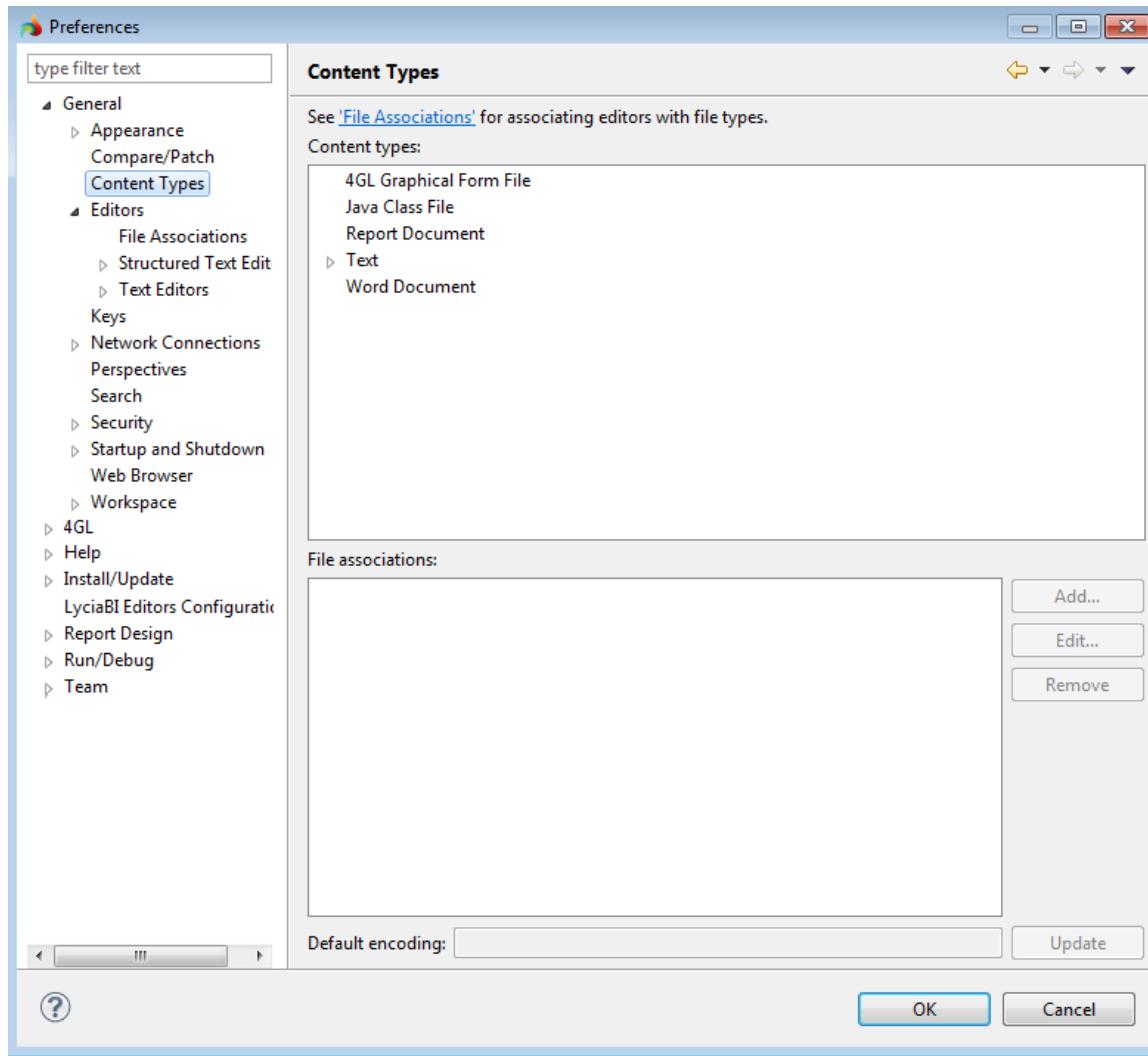


Show pseudo conflicts	Select to display pseudo changes. E.g. when two developers add exactly the same line into the same place.
Connect ranges with single line	Select to connect the differing ranges visually with a single line.
Highlight individual changes	Select if you want the individual changes to be highlighted inside the conflicts
When the end/beginning is reached while navigating an element	<p>This option specifies which action Studio should take when the end/beginning of a current element is reached while navigating</p> <ul style="list-style-type: none">• Prompt – when you reach the beginning or end of an element you are asked whether you want to go to the beginning or end of this element (if you compare a single element) or to the previous/next element (if you compare several elements). The choice will be remembered by the Studio• Loop back to the beginning/end - when you reach the beginning or end of an element you will go to the beginning or end of the same element respectively• Go to the next/previous element – if two or more elements are compared, the next/previous element will be opened when the beginning/end of the current element is reached



Content Types Preferences

The Content Types preferences page enables edition of the content types, their associated file names, and character sets. You can also associate unconditioned file names or file extensions with content types by means of this preferences page. Here is what the Content Types preferences page looks like:



Each content type contains a description of a class of files (4GL source files, XLM files, etc.). This description is used in the editor look-ups and file comparisons.

The names and extensions of a content type can be changed by selecting the content type from the tree and selecting the **Edit** button. However, the items which are marked as (*locked*) cannot be edited or removed; you can edit or remove only user-contributed content type items.

To add a new content type item, press the **Add** button; you will see a pop-up dialog prompting you to enter a file extension.

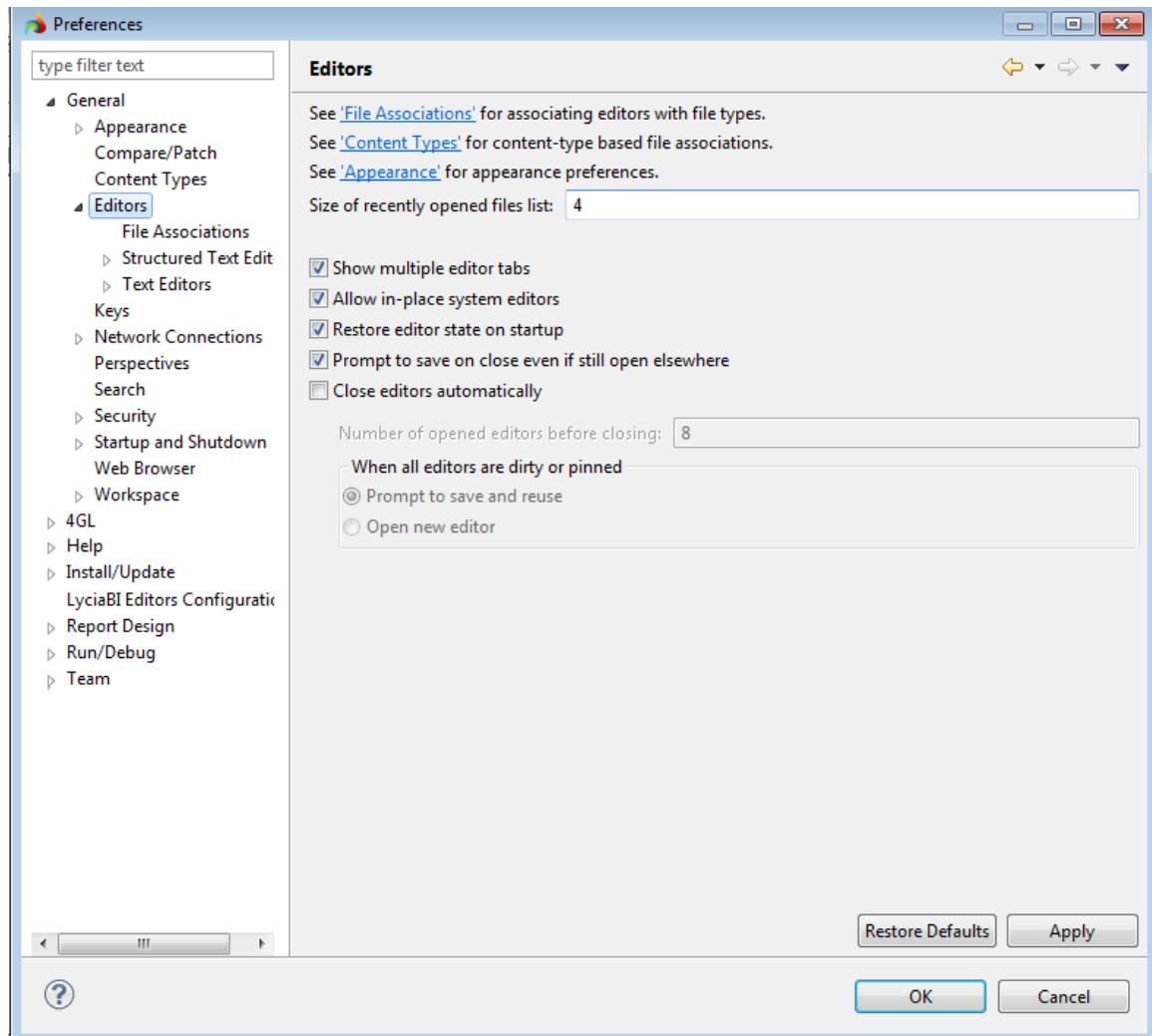


You can add and remove file names or extensions, you can also set the default encoding for a given content type. To do this, simply enter the encoding name in the provided field and click the **Update** button.

Default encoding:

Editors Preferences

The Editor Preferences page is used to adjust the settings of the editor area in the Studio. It consists of the main Editors preferences page and a number of sub-options.



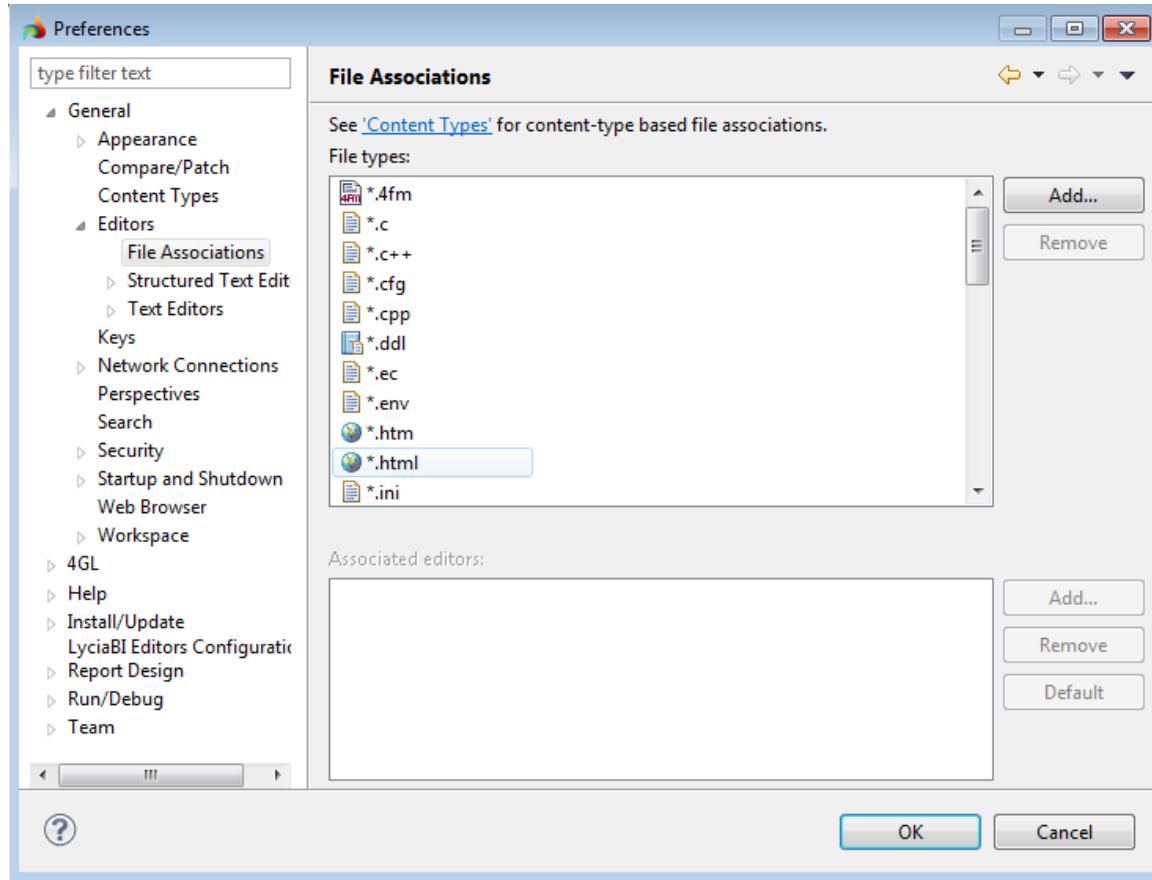
This page allows you to specify whether separate editor tabs should be used or it should be one tab for all files open in the editor, which is chosen by means of deselecting the "Show multiple editor tabs" option. If the "Close editors automatically" option is not selected, the editors can be closed automatically when the number of open editors exceeds the number specified in the "Number of opened editors before closing". The default values for the general editor related



settings are displayed above. You can also specify the "Size of the recently open files" option, which specifies the number of source files that will be displayed in the "File" option of the main menu.

File Associations Preferences

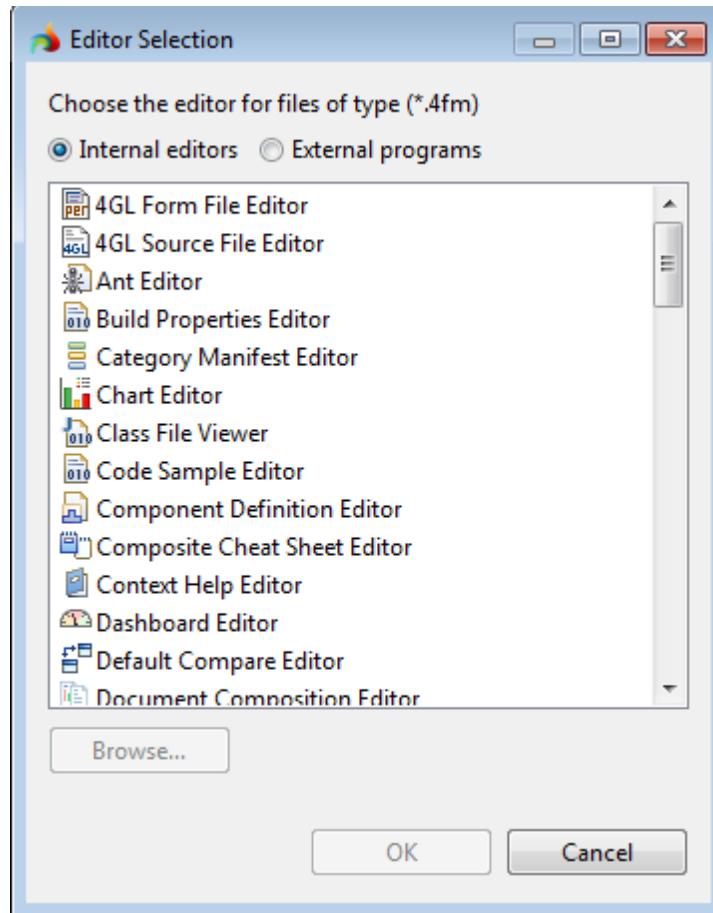
This preference page allows you to associate editors with file types in the file types list:



You can add or remove a file type to and from the list of the file types by pressing the **Add...** or **Remove** buttons respectively. You can also add or remove the file editor associated with the selected file type. In the diagram above, the *.4fm file (a graphical form file) is associated with the 4GL graphical form editor and so will be opened in this editor. The 4GL graphical form editor is the default editor for this type of file and cannot be removed from the list but another editor can be added.

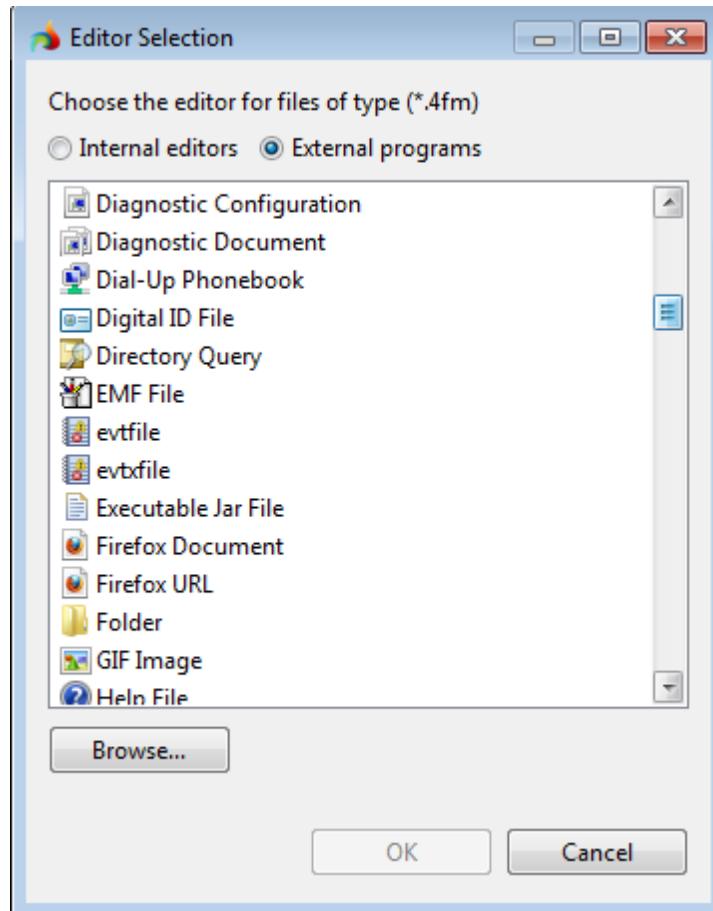


When the **Add...** button is pressed to add a new editor to the list, you are presented with a pop-up window which offers you a list from which you can choose a built-in editor:





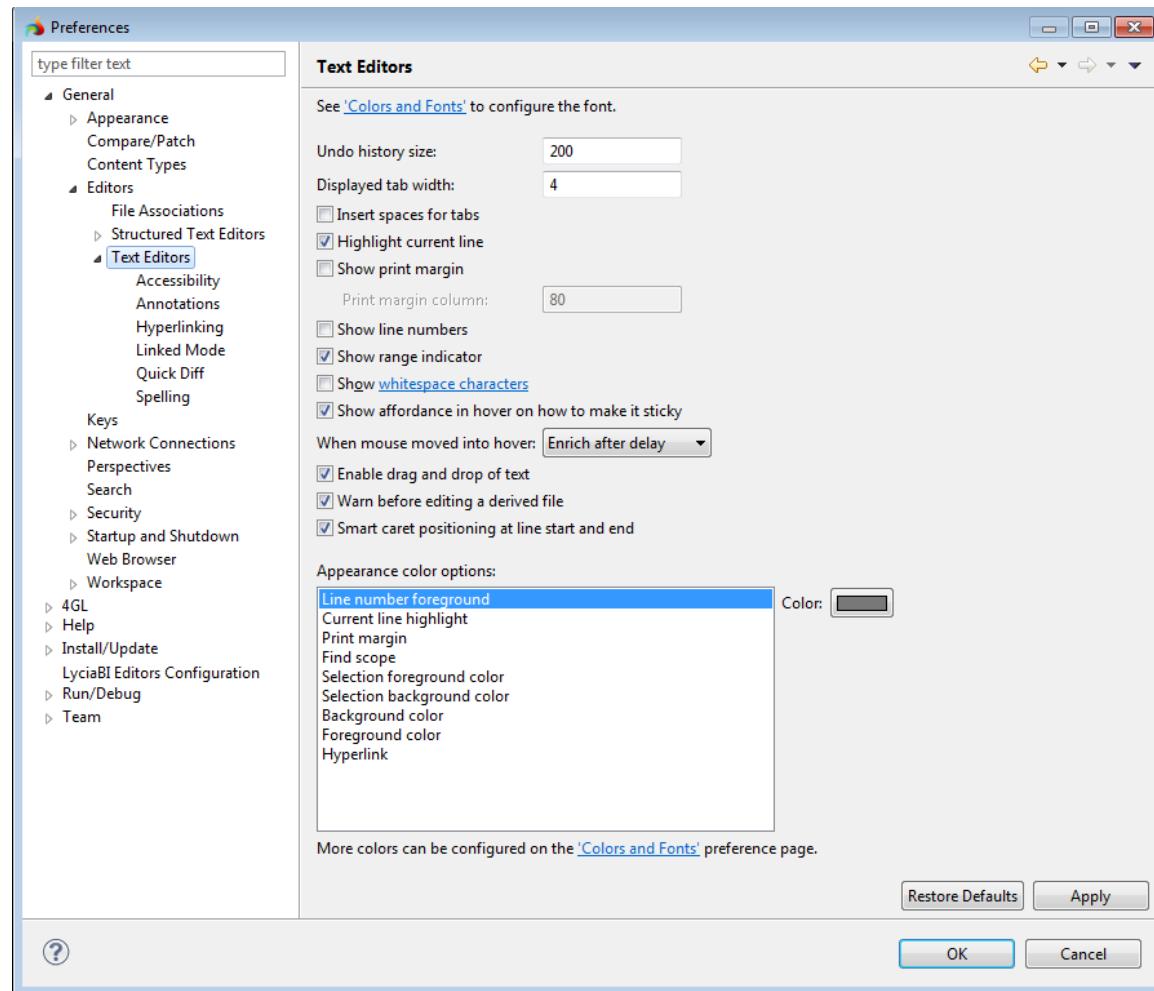
You can choose an editor from the list of available internal editors or choose from the list of external editors available on your system by choosing the "External programs" option:





Text Editors Preferences

This page is used to adjust the settings which are specific to text editors.



The default values for the settings are displayed above. The Text Editors preferences tree contains a number of pages which can be viewed if you press the expand button next to the Text Editors title in the left pane:

- The **Accessibility page** allows you to track the changes made in the source text with the help of the vertical ruler and allow change to the caret options
- The **Annotations page** allows you to choose the way in which different marks and annotations (bookmark symbols, breakpoints, etc.) will be highlighted in the editor
- The **Hyperlinking page** allows you to adjust the hyperlink navigation
- The **Linked Mode page** allows you to adjust settings for the linked mode
- The **Quick Diff page** allows you to enable quick diff options and to set the colours with which the changes made in the source text will be marked on the vertical ruler
- The **Spelling page** allows you to enable spelling check



Keys Preferences

This page provides you will the key strokes and key sequences for all the available actions and commands. The key strokes and key sequences are assigned to invoke particular commands within the Studio.

The screenshot shows the 'Keys' preferences dialog box. On the left, a tree view lists categories like General, Editors, Keys, and Help. The 'Keys' node is selected. The main area contains a table of key bindings:

Command	Binding	When	Category	User
&Rename XSD element			Edit	
About			Help	
Activate Editor	F12	In Windows	Window	
Add Block Comment	Ctrl+Shift+/	Editing in Structured ...	Edit	
Add Block Comment	Ctrl+Shift+/	Editing Java Source	Source	
Add Bookmark			Edit	
Add Class Load Breakpoint			Run/Debug	
Add Import	Ctrl+Shift+M	Editing Java Source	Source	
Add Java Exception Breakpoint			Run/Debug	
Add Javadoc Comment	Alt+Shift+J	In Windows	Source	
Add Memory Block	Ctrl+Alt+M	In Memory View	Run/Debug	
Add Repository Location			CVS	
Add Task...			Edit	
Add to Ignored			CVS	

Below the table are buttons for 'Copy Command', 'Unbind Command', and 'Restore Command'. Further down are fields for 'Name' (Add Bookmark), 'Description' (Add a bookmark), 'Binding', and 'When'. To the right is a 'Conflicts:' table and buttons for 'Filters...', 'Export CSV...', 'Restore Defaults', 'Apply', 'OK', and 'Cancel'. A question mark icon is at the bottom left.

A "key stroke" is the pressing of a key on the keyboard whilst optionally holding down one or more of these modifier keys: Ctrl, Alt, and Shift. For example, holding down Ctrl then pressing X produces the key stroke combination of Ctrl+X. The pressing of the modifier keys themselves does not influence the Studio; they do not constitute keystrokes by themselves.

A "key sequence" is one or more key strokes. Traditionally, the Emacs scheme assigns 2 or 3 key stroke key sequences to particular commands. For example, the normal key sequence assigned to "Close All" in Emacs is Ctrl+X Ctrl+C. To enter this key sequence, you should press the key stroke Ctrl+X followed by the key stroke Ctrl+C. It is recommended that keyboard shortcuts be 4 key strokes in length (or less).

A 'key binding' is the assignment of a key sequence to a command.



There are two available schemes of key strokes and key sequences in the Studio, which can be chosen from the combo box at the top of the preferences page:

- The Default scheme contains a general set of bindings, in many cases recognizable as traditional key sequences for well known commands. For instance, Ctrl+A is assigned to Select All, and Ctrl+S is assigned to Save.
- The Emacs scheme contains a set of key bindings familiar to users of Emacs. For instance, Ctrl+X H is assigned to Select All, and Ctrl+X S is assigned to Save.

Customizing Key Bindings

If you want to assign a new key binding to a command, select this command. E.g. select the About command. Currently it does not have any key binding assigned to it so the "Binding" field is empty and the "When" field is absent:

The screenshot shows the 'About' command in the preferences dialog. The 'Name:' field is 'About'. The 'Description:' field contains 'Open the about dialog'. The 'Binding:' field is empty, indicated by a single vertical bar character. The 'When:' field is also empty.

To assign a key binding to it, place the cursor in the Binding field and press the keys combination you wish to use as a key stroke for invoking the About option (e.g. hold down the Control key and then press "6"). The "When" field will appear and you will be able to set the context for the new key binding:

The screenshot shows the 'About' command in the preferences dialog. The 'Name:' field is 'About'. The 'Description:' field contains 'Open the about dialog'. The 'Binding:' field now contains 'Ctrl+6'. The 'When:' field is populated with 'In Windows'.

There are a finite number of simple common key strokes available that can be applied to a multitude of commands. The conflicts can appear between key bindings if you assign the same key sequence to different commands in the same context. Such conflicts are shown in the "Conflicts" field:



Conflicts:	
Command	When
About	In Windows
Activate Editor	In Windows

Perspectives Preferences

This page allows you to manage available perspectives.

The screenshot shows the 'Perspectives' section of the Eclipse Preferences dialog. The left sidebar lists various preference categories, and the main area displays settings for managing perspectives. The 'Available perspectives:' list includes: 4GL (default), CVS Repository Exploring, Database Debug, Database Development, Debug, Java, Java Browsing, Java Type Hierarchy, LyciaBI, Plug-in Development, Report Design, Resource, Team Synchronizing, and XML. Buttons for 'Make Default', 'Revert', and 'Delete' are visible on the right. A note at the bottom states: 'Note: 'Revert' removes the customization from the selected perspective. This only applies to newly opened perspectives.'

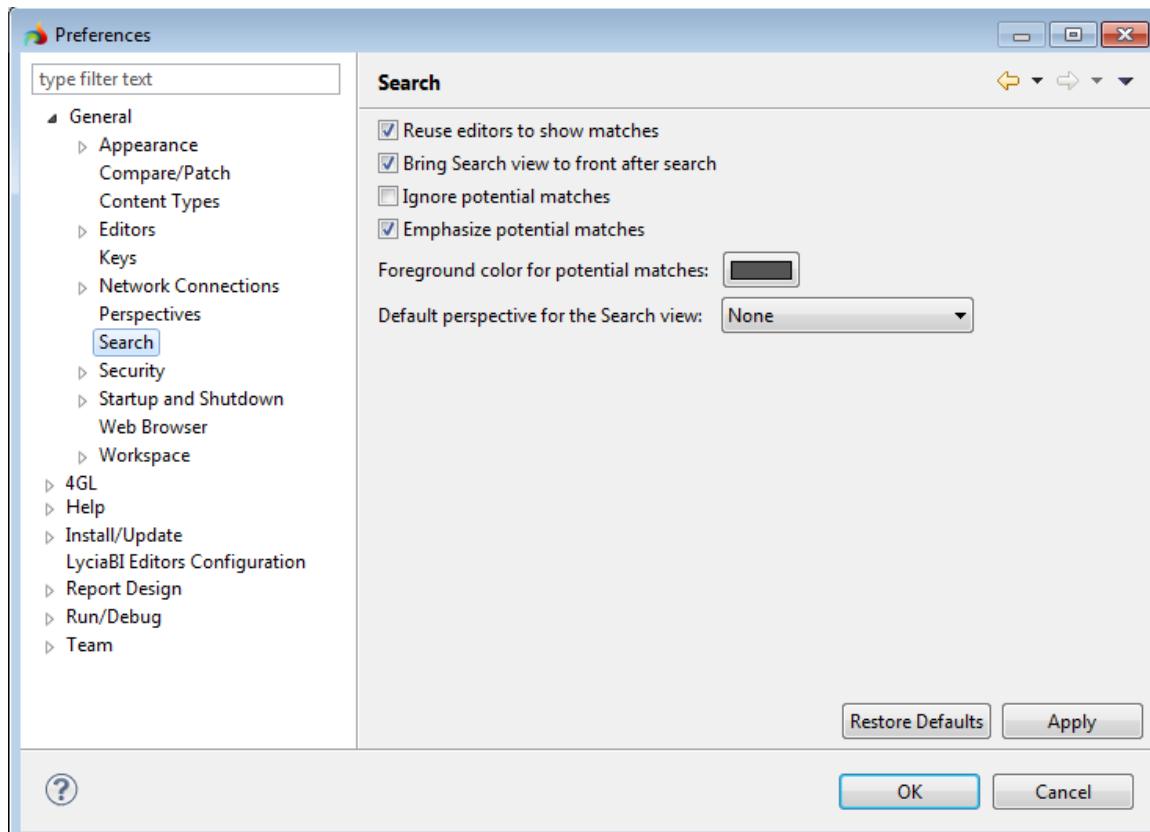
A perspective can be opened in the same Studio window or in a new Studio window depending on the "Open a new perspective" setting. The "Open a new view" option defines whether a new view is opened in its default position within the current perspective, or it is opened as a fast view and is docked to the side of the current perspective.



This page also allows you to adjust which perspective should be opened when a new project is created, reset the perspectives and delete a perspective. Only user-created perspectives can be deleted. A user-created perspective is a customized perspective saved by the user by means of the **Window -> Save Perspective As...** option of the main menu.

Search Preferences

This page adjusts the preferences for the search option.



The default values of the options are displayed in the image above.

Option	Description
Reuse editors to show matches	Select to display the search results in the same editor. Otherwise each search result will be displayed in a separate editor.
Bring Search view to front after search	Select to make the Studio open the Search view and bring it in the front when the search is finished.
Ignore potential matches	Select to display only exact matches.
Emphasize potential	Select to highlight potential matches in the Search view. A



matches	potential match is not a 100% match.
Foreground color for potential matches	You can select the colour with which the potential matches are highlighted.
Default perspective for the Search view	Select the perspective which becomes active when the search results are displayed.

Startup and Shutdown Preferences

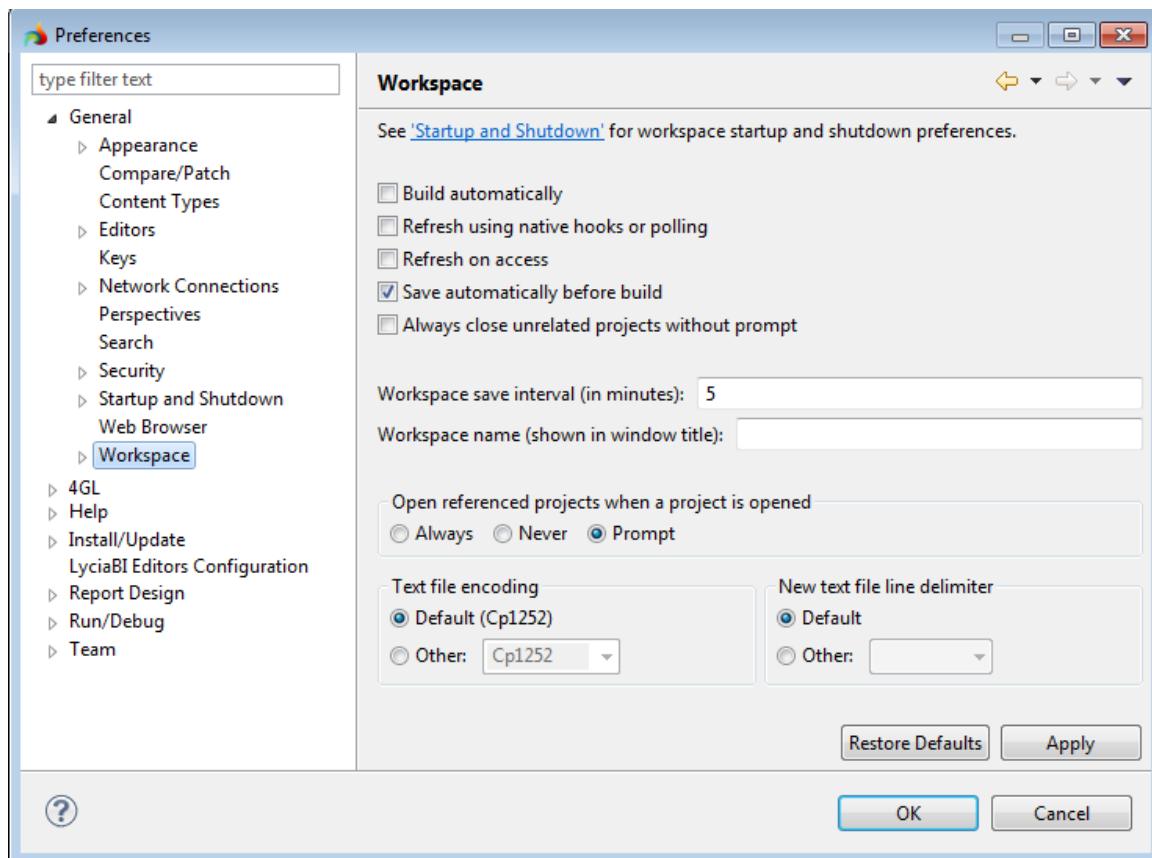
This page allows you to select plug-ins and to define whether they will be activated automatically during the startup.

Option	Description
Prompt for workspace on startup	Select to make the Studio prompt for the workspace on startup.
Refresh workspace on startup	Select to synchronize the contents of the Studio with the file system on startup.
Confirm exit when closing last window	Select to make the Studio ask if you wish to exit, when closing the last window.



Workspace Preferences

This preference page is used to set the Studio-specific settings related to the workspace.



Option	Description
Build automatically	Select to enable automatic building every time a source file is saved
Refresh automatically	If this option is selected, the workspace resources will be synchronized with their corresponding resources in the file system automatically. Note: This can potentially be a lengthy operation.
Save automatically before build	Select to enable the option which automatically saves the manually built files
Workspace save interval (in minutes)	The number on this field indicates how often the Studio state is saved to disk
Open referenced projects	Select to make Studio open projects which are referenced



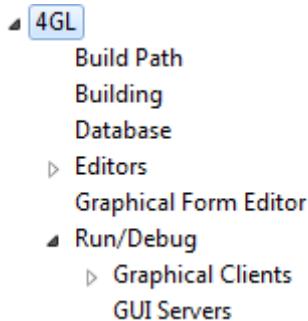
when a project is opened	by the current project
Text File encoding	Use this option to specify the encoding to use when saving text files in editors.
New text File line delimiter	Use this option to specify the line delimiter to be used for the new text files. (It will not affect the line delimiter of the existing files).

The Workspace preferences set also contains a number of pages which can be viewed if you press the plus button next to the Workspace title in the left pane tree. They are:

- The **Build Order page** allows you to change the default order in which the open projects are built, if the "Build All" option is used
- The **Linked Resources page** allows you to define path variables
- The **Local History page** allows to adjust the size and storage period of the local history

4GL Preferences

The 4GL preferences page deals with the specific settings such as running/debugging 4GL programs, setting the application server and the database etc. To see the preferences pages grouped under the 4GL title, expand the tree next to it:

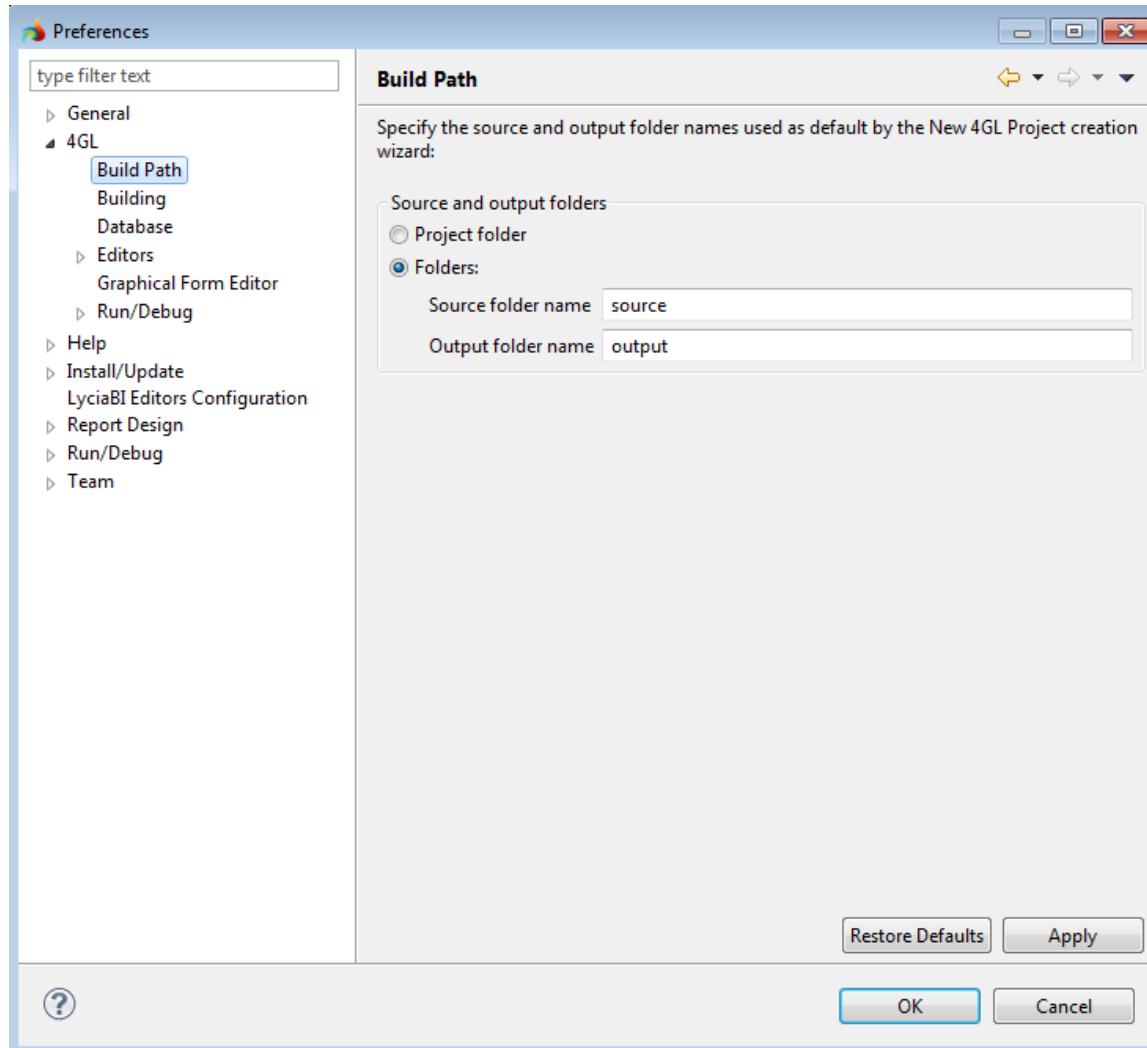


- The **Building preferences page** is used to specify whether the Console view will be cleaned before starting a new build.
- The **Database preferences page** is used for setting the database type to which you want to connect (Informix, Oracle, etc.)



Build Path Preferences

This preferences page is used to specify the folders used as the output and source folders by default when a new project is created. These settings do not affect the existing projects.



If the "Project Folder" option is selected, all the files (source and output) are stored in one folder which is the root project folder.

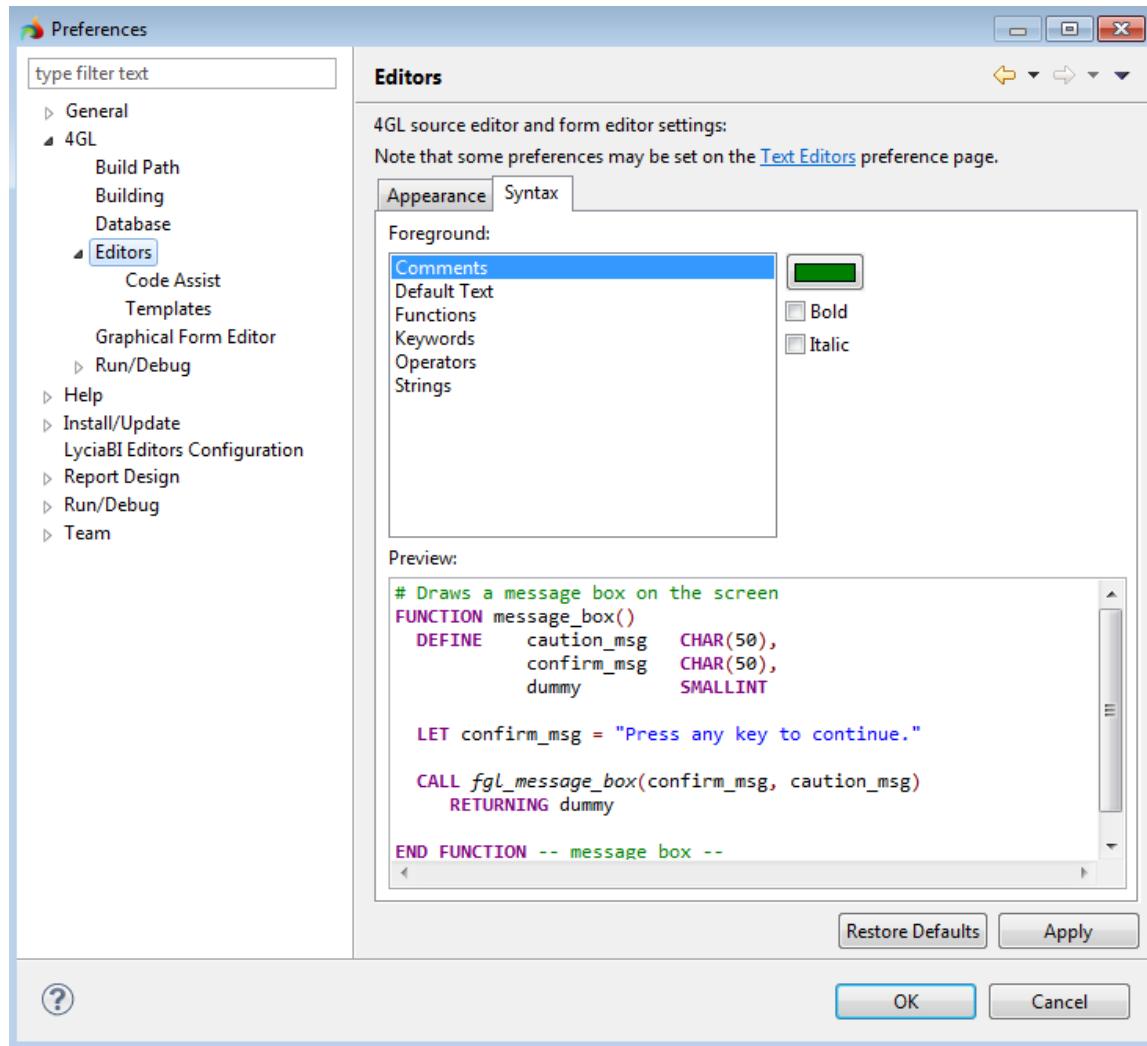
If the "Folders" option is selected, the source and output files are stored in different folders the names for which can be specified (the default folder names are "source" and "output").

Editors Preferences

This preferences page specifies the settings which apply to the 4GL source editor and 4GL form editor in addition to the Text Editors preferences in the General section. These settings affect only the text form editor and do not affect the graphical form editor.



It has two tabs: The Appearance tab allows you to specify whether you want to insert spaces for tabs when typing, whilst the Syntax tab which is used to adjust the appearance of the 4GL source code:



Here you can choose the font weight and colour for each component of the 4GL source code and see the changes in the Preview field.

The Editors preferences section of the 4GL preferences contains a number of preferences pages which can be viewed, if you press the plus button next to the Editors title in the left pane tree.

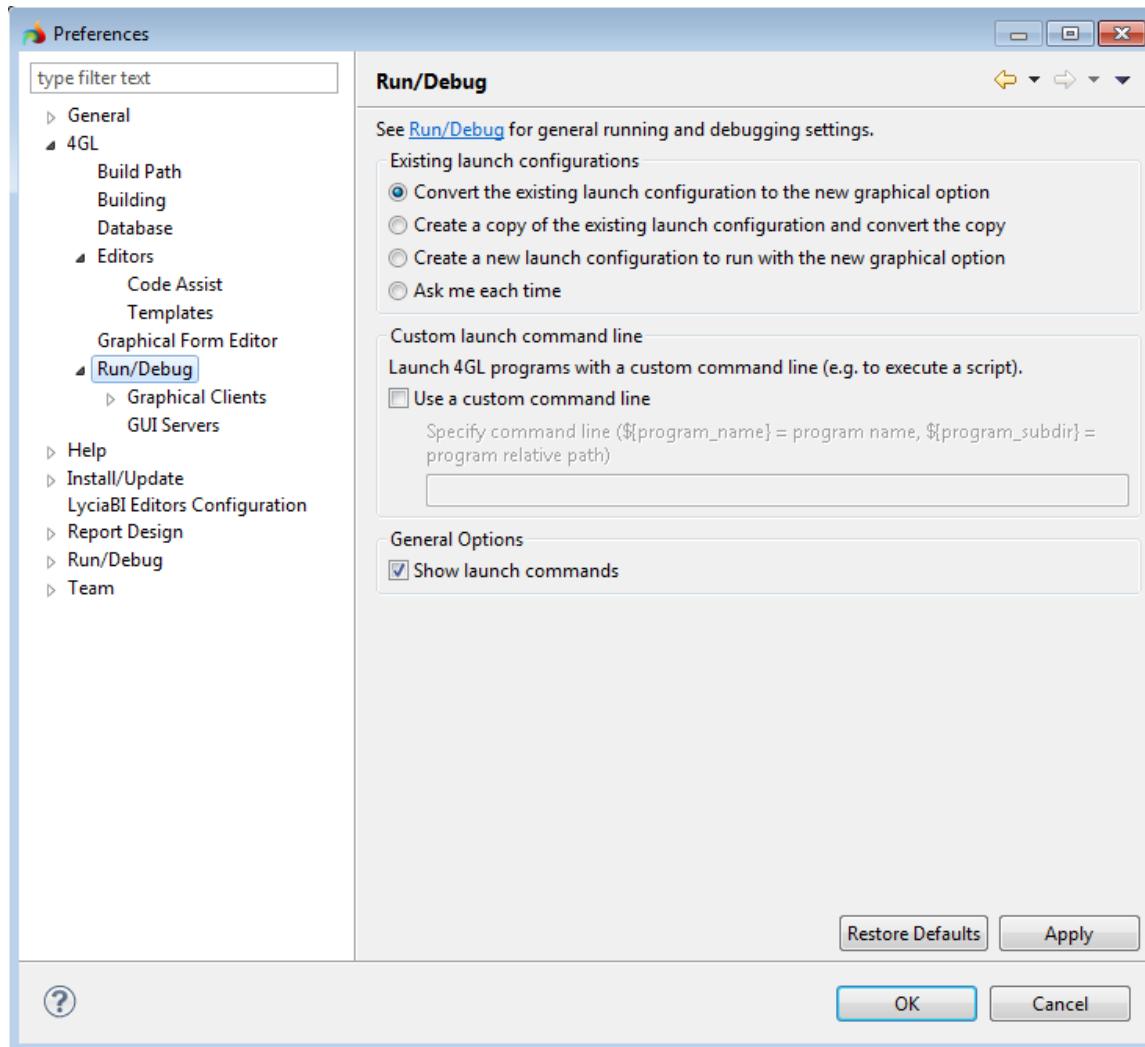
- The **Code Assist page** allows you to adjust the settings which convert the case of the keywords on completion (into uppercase or lowercase)
- The **Code style page** allows you to set the default delimiter symbols used in screen forms



- The **Templates page** allows you to enable/disable, create or remove the templates for 4GL source files, GUI script files, 4GL form files, C source files, etc.

The Run/Debug Preferences

The Run/Debug preferences page of the 4GL preferences section offers additional run/debug settings which specifically apply to running and debugging 4GL programs. For general Run/Debug settings, see the general Run/Debug preferences section:



Option	Description
Action on double-clicking	Here you can choose the default mode in which a program will be run when double clicked in the 4GL Project View. You can specify whether the program should be run in character mode or in GUI mode



a 4GL program	(LyciaDesktop or LyciaAjax). This setting applies only to those programs which do not have their own run configurations.
Existing launch configurations	This option specifies what should be done when a program which has its own run/debug configuration is run by means of the "Run As..." or "Debug As..." option which applies a particular option (LyciaText, LyciaDesktop, LyciaWeb, etc.) different from the current configuration option. If you choose the "Convert the existing launch configuration..." option, the existing launch configuration will be updated in accordance with the new choice (it is the default value). If you choose the "Create a copy of the existing launch configuration..." or "Create a new launch configuration...", one more run/debug configuration for the same program will be created and you will be asked which configuration you want to use each time the program is run by double clicking.
Custom launch command line	This option allows you to specify a custom command line which is executed when a program is run. You can use it to execute a script, etc.
General options	Select if you want Studio to build a program before it is executed, when the program is not built.

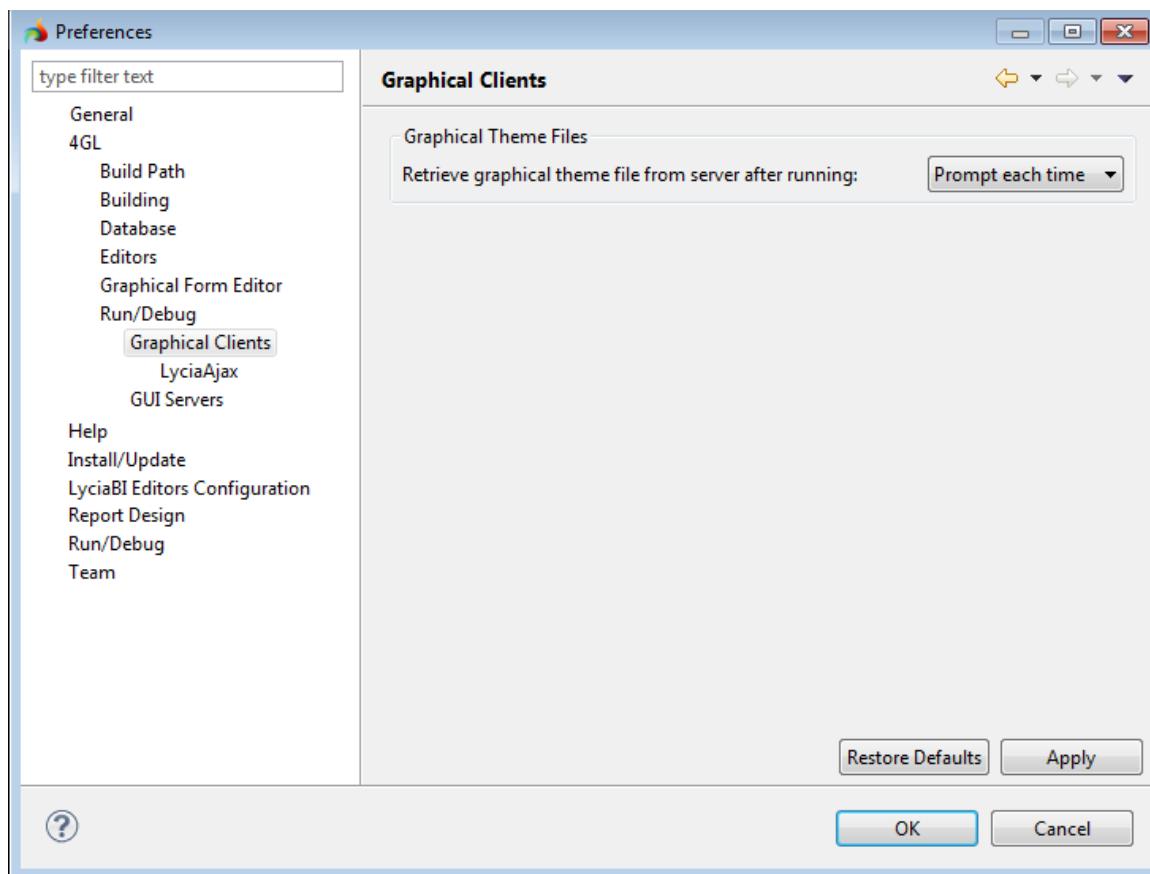
The Run/Debug 4GL preferences section contains another preferences page which deals with the Application server configurations.

Graphical Clients Preferences

This page allows you to specify graphical clients settings. On the Graphical Theme Files page you can specify whether Lycia Desktop should apply the graphical themes file, if any is present, to the application run.

There are three available options:

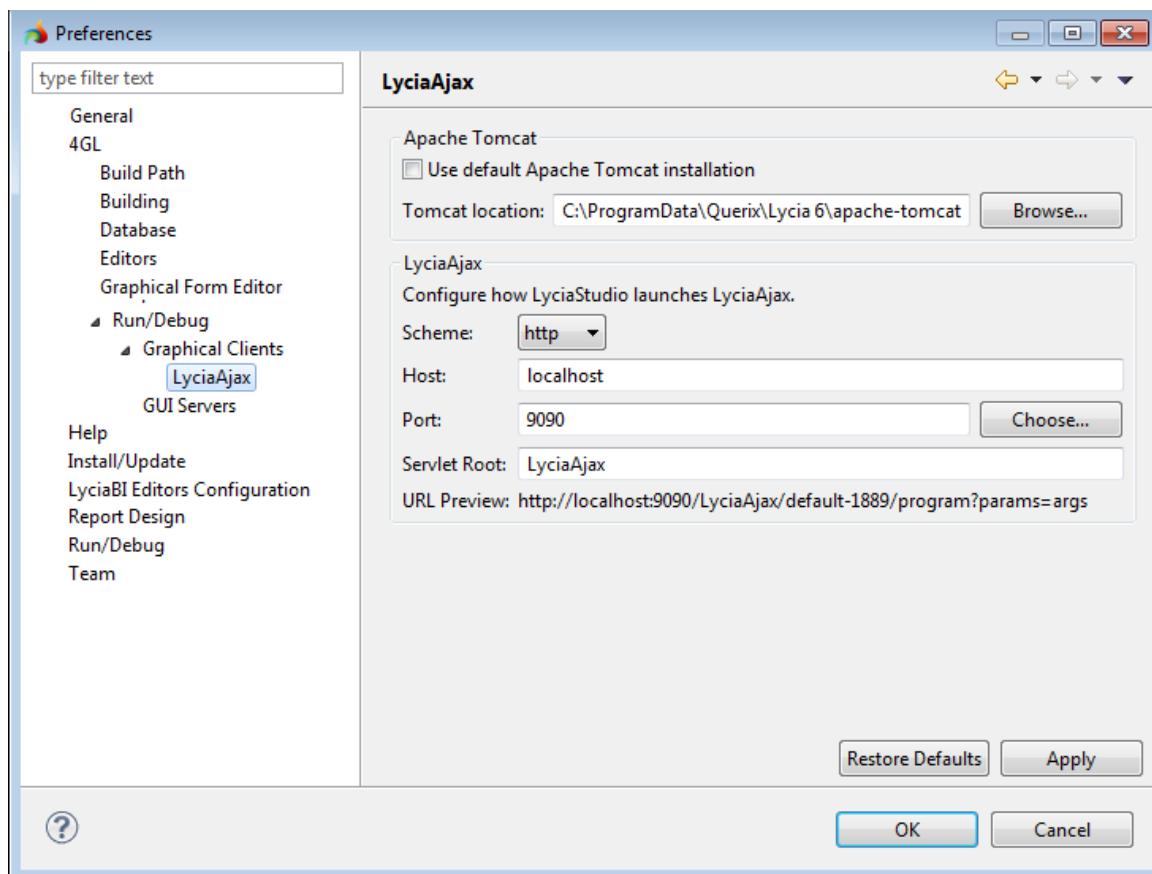
- Always - Lycia Desktop will automatically apply the specified theme file
- Never - no theme file will be applied, even if one is specified
- Prompt each time - Lycia Desktop will prompt the user each time so that they could decide whether the theme file should be applied



LyciaWeb Settings

On this page, one can specify the Web settings different from the default ones, specified during the Lycia Installation.

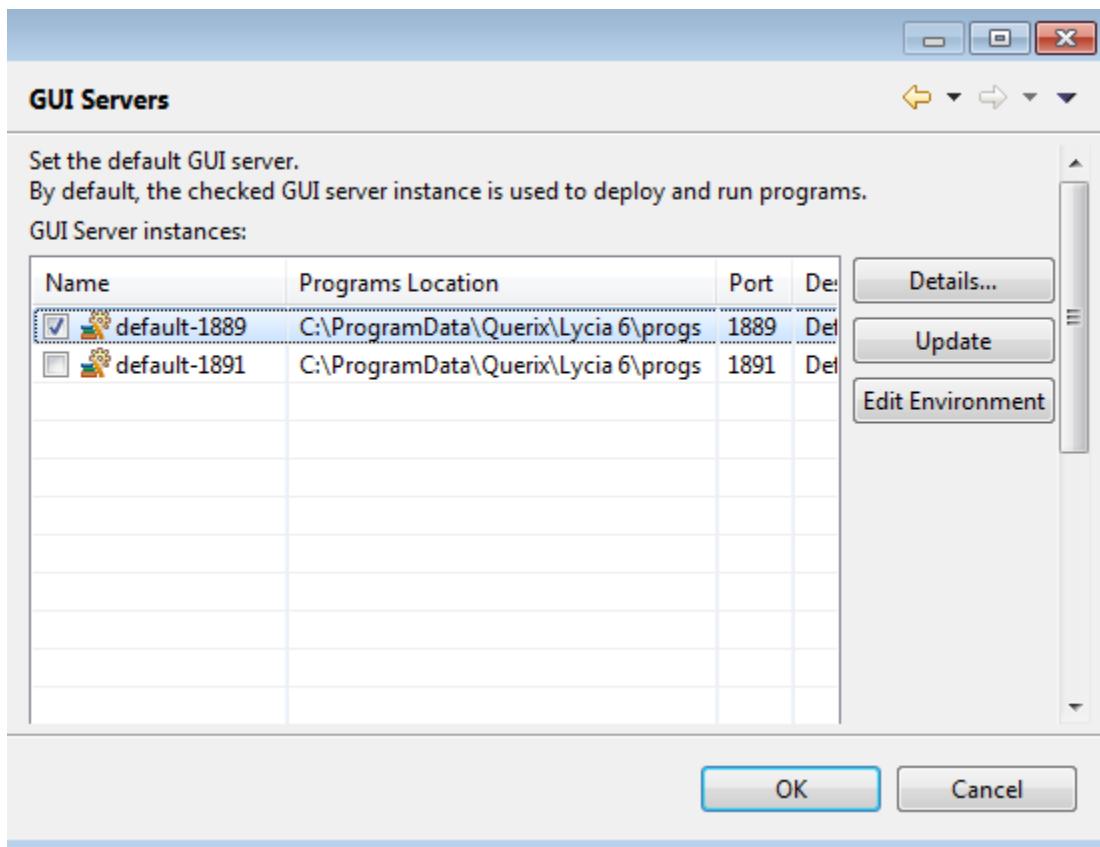
Here, you can specify the new web services server location, as well as server scheme, host, port, and servlet root.





GUI Servers Preferences

This page allows you to choose the default GUI server used to deploy and run programs and can change the environment settings.



Select the server you wish to use as the default server for running and deploying programs. You can also specify an individual server for running or debugging a program by using the **Run -> Run Configurations** menu or the **Run -> Debug Configurations** menu of this program respectively.

Select the **Details** button to see the information about the selected application server.

The **Edit environment** button allows you to see and edit environment variables. The environment settings are stored in the \$QUERIXDIR\Lyacia\Lyacia\etc\net.env file; this file is opened in the built-in text editor of the Studio when this button is pressed:



```
inet.env
LYCIA_DIR=C:\work\Querix\Lycia\Lycia
MSGPATH=C:\work\Querix\Lycia\Lycia\msg
LINES=24
COLUMNS=80
LYCIA_GUI=gui
LYCIA_DRIVER_PATH=C:\work\Querix\Lycia\Lycia\lib
FGLPROFILE=C:\work\Querix\Lycia\Lycia\etc\fglprofile.std
QXDEBUG=!
QXBREAKCH_START="+=-*"
QXBREAKCH_END=":"
PATH=C:\work\Querix\Lycia\Lycia\bin;C:\work\Querix\Lycia\Common\bin;$PATH
```

Close the Preferences menu, view and edit the file if necessary and save the file. To update the list of the available GUI servers, select the **Update** button.

The Install/Update Preferences

This preferences are used to configure the update and install settings. The general Install/Update page contains the settings which specify the reaction of the Studio, is something during the update or install goes wrong:

The screenshot shows the Eclipse Preferences dialog with the 'Install/Update' page selected. The left sidebar lists various preference categories, and the right panel displays specific settings for managing software updates and installations.

Browsing for updates:

- Show only the latest versions of available software
- Show all versions of available software

When software selected for an install wizard may not be compatible:

- Open the wizard anyway to review or change selections
- Report the problems and do not open the wizard
- Ask me what to do when it happens

[Uninstall or update](#) software that is already installed

Buttons at the bottom: Restore Defaults, Apply, OK, Cancel, ?

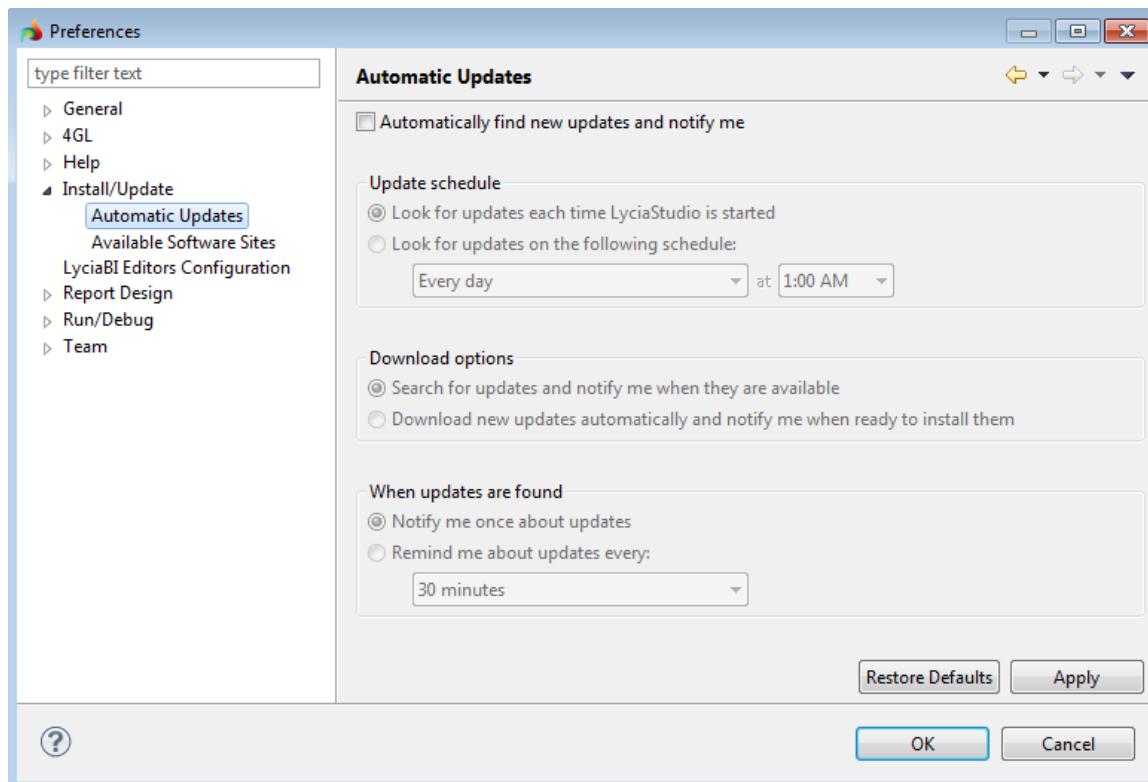


This preferences section contains two sub-pages:

- The Automatic Updates page contains the settings used for scheduling updates
- The Available Software Sites page stores the web sites which can be used for updating the installed Studio features and for installing new ones.

Automatic Updates

The **Window -> Preferences -> Install/Update -> Automatic Updates** page contains the settings for the automatic Studio update. Here is what this page looks like:



To enable the automatic updates you should check the box at the top of the page. Otherwise none of the schedule options will be enabled.

- Update schedule - you can decide whether you want the Studio to look for updates each time it is started or just look for updates on a regular basis. E.g. you can adjust these settings so that the Studio will look for updates every day or once a week on the specified day of week at the specified hour. If the application is not active at a scheduled time and the search is past due, it will immediately start on the next startup.
- Download options - you can either instruct your Studio to download the updates automatically and then notify you, or to just notify you without downloading anything. In both cases, the update scheduler will execute in the background when necessary based

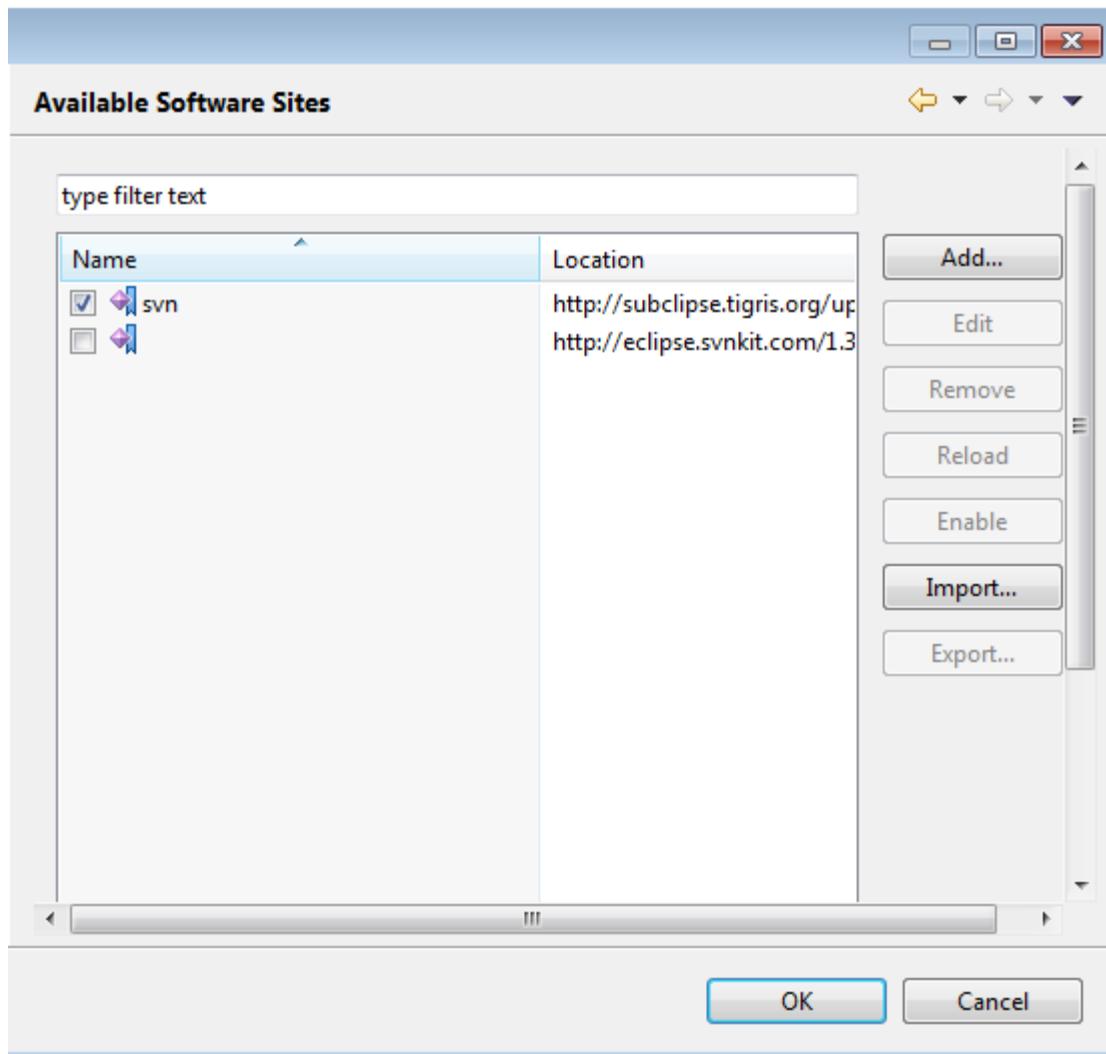


on the scheduling options. There will be no messages, if no updates are found. If there are updates, the behaviour will depend on the chosen download option.

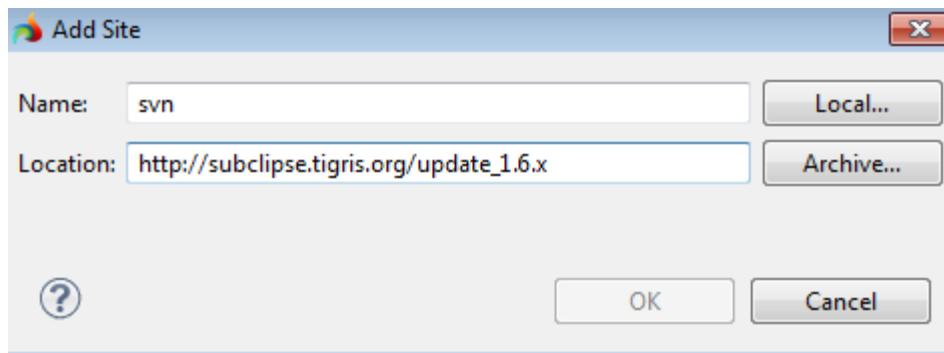
- When updates are found - you set a single notification or a repeated notification.

Available Software Sites

The **Window -> Preferences -> Install/Update -> Available Software Sites** preferences page allows you to specify the sites from which the new plug-ins can be downloaded and installed and the site from which the Studio updates can be downloaded.



You can add a new site by pressing the **Add...** button. In the example below we specify the site where the eclipse SVN plug-in subclipse is stored:



For more details about installing this plug-in see [Installing a new plug-in](#).

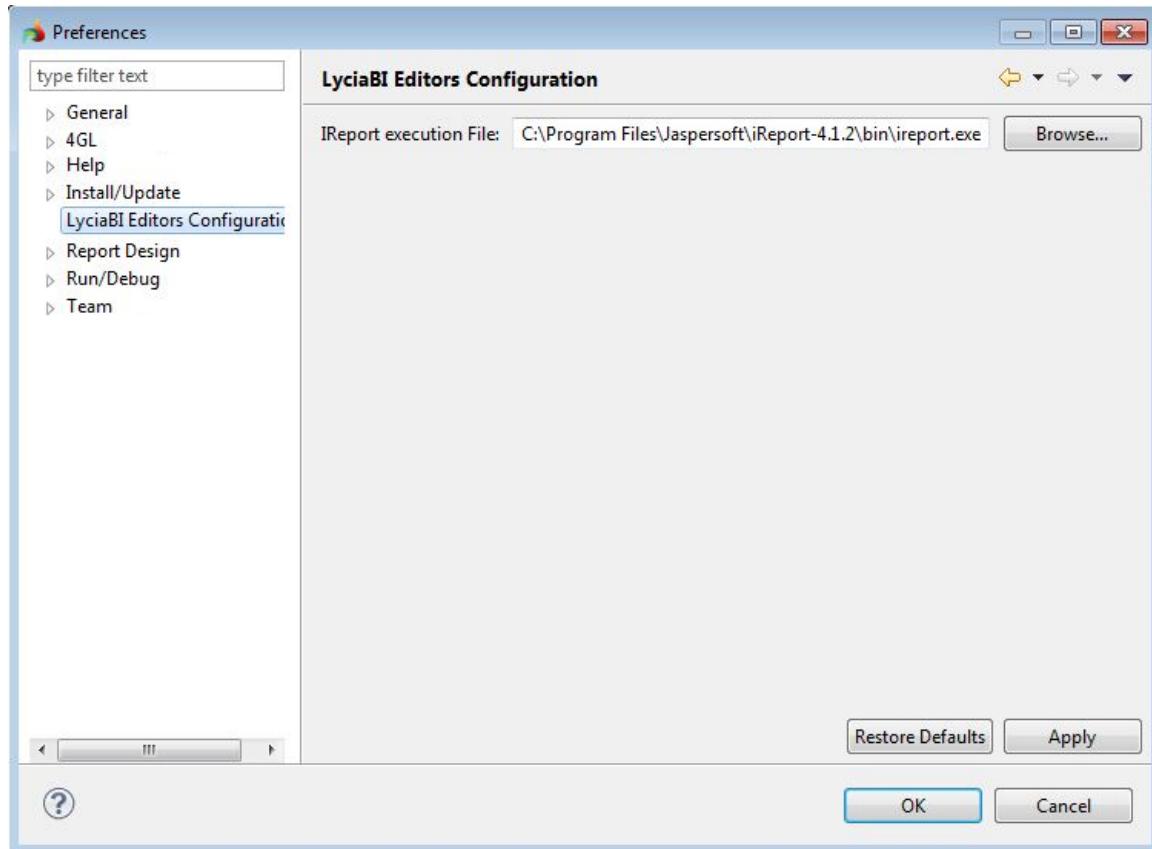
To delete a site click **Remove**, to change the address or the name, click **Edit**. You can verify if the site is up and whether you have typed the correct address by clicking **Test Connection**. The disabled sites will not be shown in the list of available sites in the dialog for installing new software and will not be referenced when you search for updates. To make a site available, you should **Enable** it.



LyciaBI Editors Configurations

LyciaBI Editors Configurations option is used to set up the path to a third-party tool that will be used to create Jasper reports.

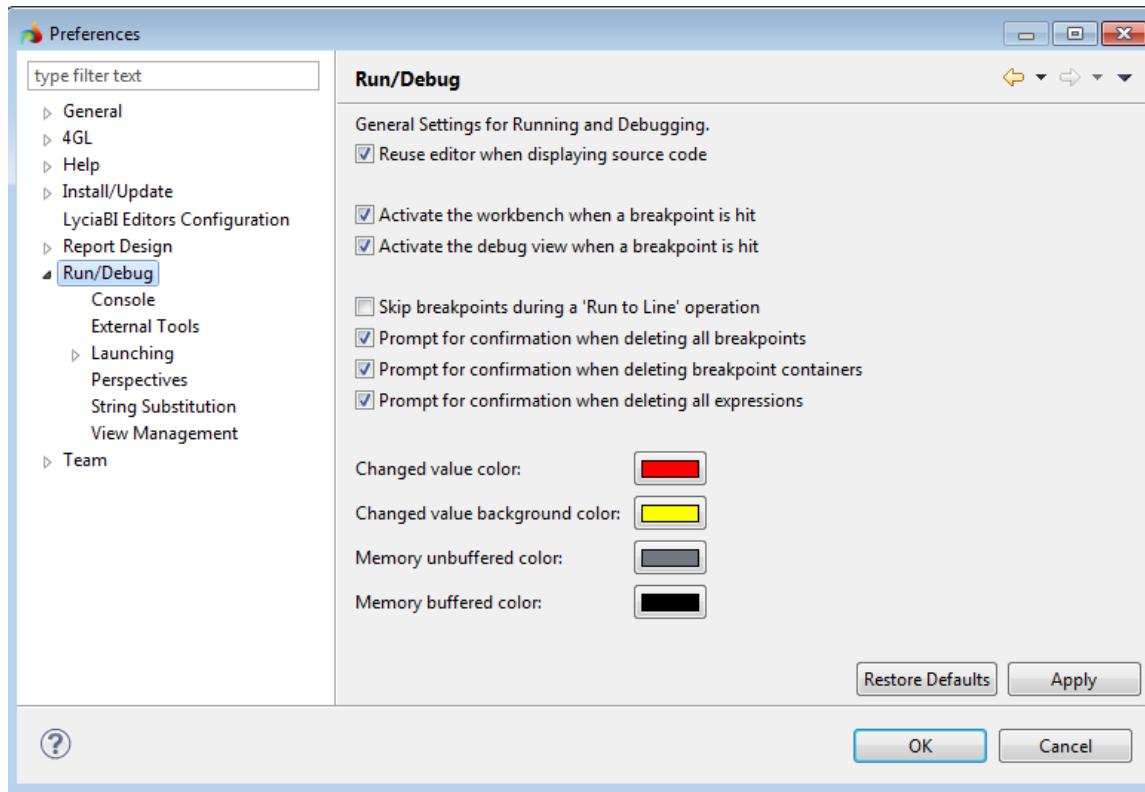
To create a Jasper report you need to have Jasper IReport installed on your computer. Specify the path to the IReport executable file in the LyciaBI Editors dialog window. Once you have done this, Lycia will automatically run the Jasper IReport tool for you to create a Jasper report.





Run/Debug Configurations

This preferences section contains general settings for running and debugging activities.



The diagram above illustrates the default values for the Run/Debug general settings.

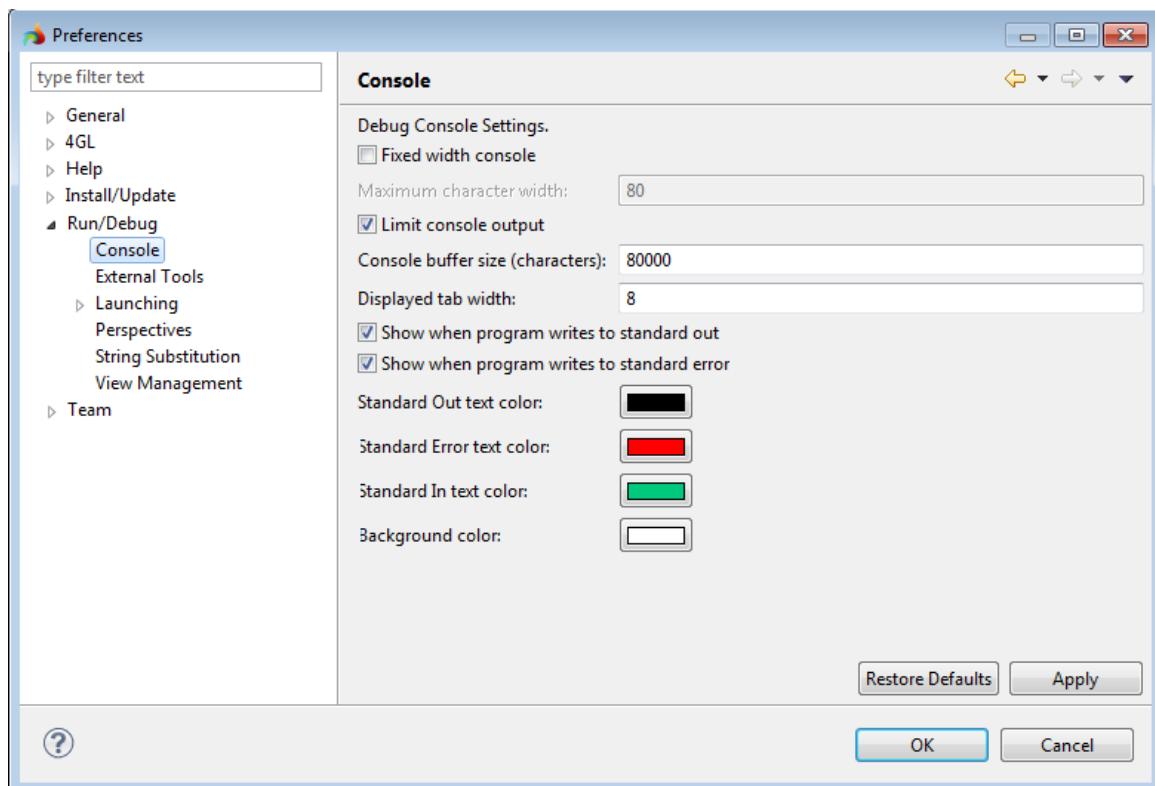
Option	Description
Reuse editor when displaying source code	Select this option to make the debugger reuse the editor for opening the source files the source code of which is currently stepped through and not to open each file in a separate editor.
Activate the workbench when a breakpoint is hit	Select to make the Studio active and shift the running application into the background, when a breakpoint is hit.
Activate the debug view when a breakpoint is hit	Select to make the Studio active, open the Debug perspective and shift the running application into the background when a breakpoint is hit.
Skip breakpoints during a 'Run to Line' operation	This option controls whether breakpoints are ignored when performing a 'Run to Line' operation. When the option is on, the debugger does not suspend at breakpoints encountered when a 'Run to Line' operation is invoked. When the option is off,



	breakpoints behave normally.
Prompt for confirmation when deleting all breakpoints	Select if you want to be prompted for confirmation when you try to delete all of your breakpoints.
Prompt for confirmation when deleting breakpoint containers	Select if you want to be prompted for confirmation, when you try to delete a breakpoint container, e.g. a breakpoint working set.
Changed value colour	This option allows you to specify the colour of a changed value in the Variables View, Expressions View and other views there the running program variables are rendered.
Changed value background colour	This option allows you to specify the colour of the background of a changed variable, e.g. in the Variables View showing columns.

Console Preferences

This preferences page is used to adjust the console settings.

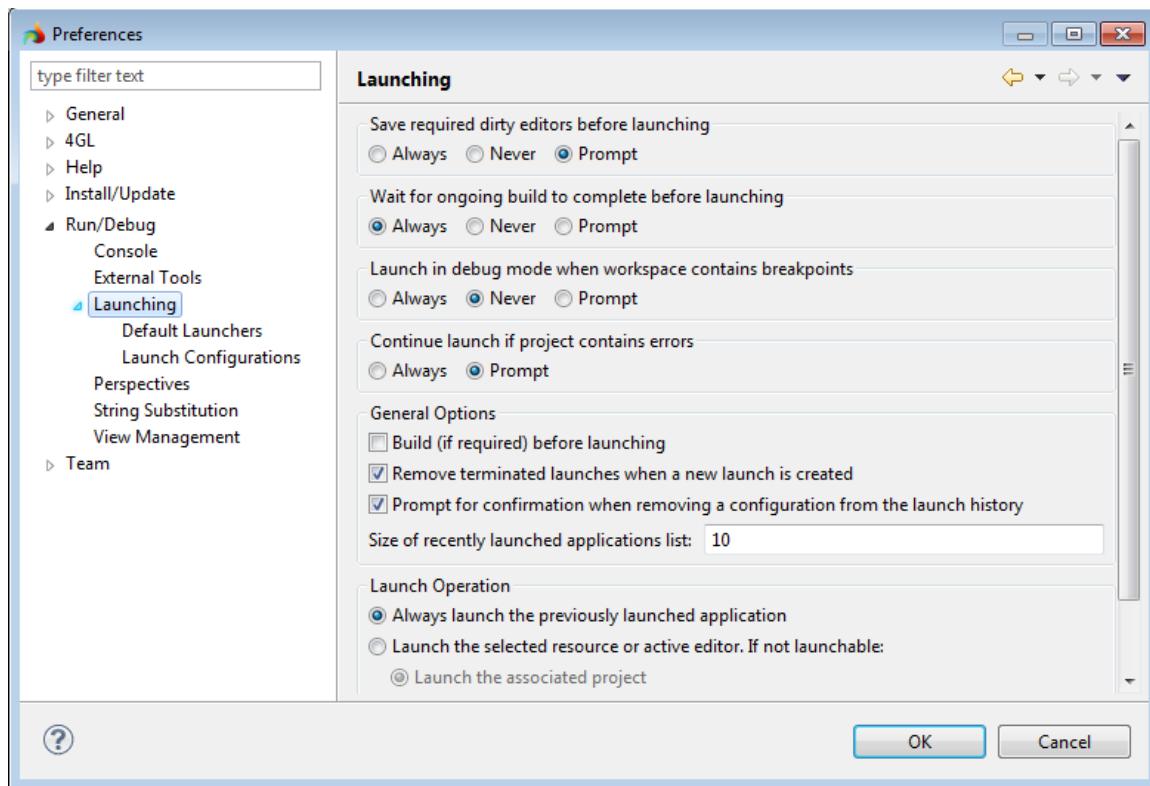




Option	Description
Fixed width console	Select to prevent applications from writing long console, which will require horizontal scrolling. When the fixed width of the console is set, the console output is automatically wrapped.
Limit console output	If you check this option, you must specify the limit for the number of characters buffered in the console. When there are more characters in the console output than the number specified in this option, the exceeding characters are truncated from the beginning of the buffer.
Displayed Tab Width	Here you can specify the default width of the console in characters.
Show When Program Writes to Standard Out	Select if you want the Studio to open the console or to bring it to the top, if it is already open, when something is written to the system out stream.
Show When Program Writes to Standard Error	Select if you want the Studio to open the console or to bring it to the top if it is already open, when something is written to the system err stream.
Standard Out Text Color	Here you can specify the colour of the text written to the standard output stream by an application.
Standard Error Text Color	Here you can specify the colour of the text written to the standard error stream by an application.
Standard In Text Color	Here you can specify the colour of text written to the console to be read by an application.
Background Colour	Here you can specify the background colour of the console

Launching Preferences

The Launching preferences can be used for configuring launching options.



The diagram above illustrates the default launching settings.

Option	Description
Save required dirty editors before launching	This option specifies whether the Studio should save the dirty editors before an application is launched.
Wait for ongoing build to complete before launching	This option specifies whether the Studio should wait till the ongoing building is complete before launching an application.
Launch in debug mode whenever the workspace contains breakpoints	This option specifies whether an application which contains breakpoints should always be run in the Debug mode even if the run button is pressed.
Continue launch if project contains errors	This option specifies whether the current project should be run, if a project related to it contains a compile-time error.
Build (if required) before launching	If the project requires building, an incremental build will be performed prior to launching an application.
Remove terminated	When a new application is run in the Debug mode, other



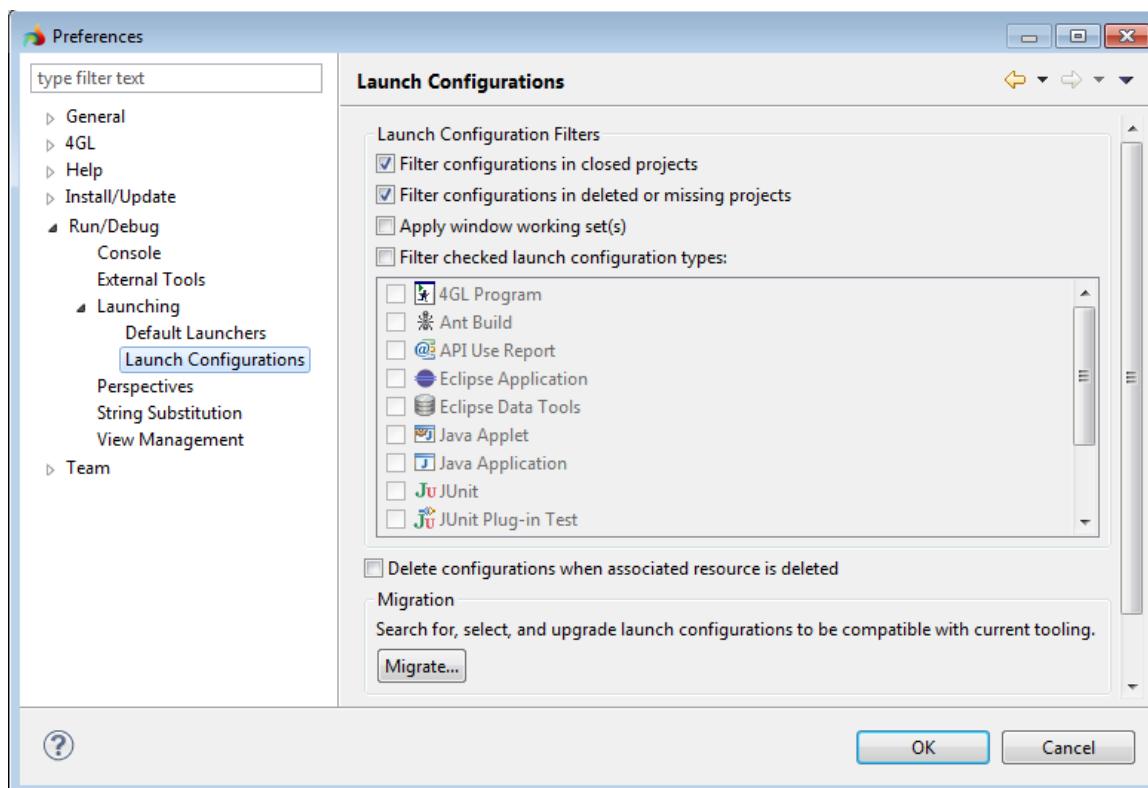
Launches when a new launch is created	terminated applications are removed from the Debug View.
Size of recently launched applications list	Here you can set the limit to the number of launches which will appear in the Run/Debug pulldown launch history menus.
Launch Operation	This option specifies what resource should be launched, when the run or debug button is pressed on the toolbar.

- The **Default Launchers page** allows you to select what tool is to be used for launching an application, if more than one tool exists for the same thing. This page remains disabled most of the time, because the conflict of tools is very rare.



Launch Configurations Preferences

This preferences page allows you to set the launch configuration filters. Here you can set filtering options that are used throughout the Studio to hide some kinds of launch configurations. These filtering setting affect the launch dialog, launch histories and the workbench.



Option	Description
Filter configurations in closed projects	Select to filter out the launch configurations for the currently closed projects.
Filters configuration in deleted or missing projects	Select to filter out the launch configurations for the deleted or missing projects.
Apply windows working set	Applies the filtering from any working set currently active to the visibility of configurations associated with resources in the active working sets.
Filter check configuration types	Select to filter all configurations of the selected type regardless of the other filtering options.
Delete configurations when associated	Select if you want the launch configurations associated with a project to be deleted when you delete the project.



project is deleted	
Migration	Sometimes new features may be added to the launching framework of the LyciaStudio. Some of these features may be implemented automatically and some may require migration. Upon pressing the Migrate... button, if there are any configurations requiring migration, they are displayed, and the user can select the ones they want to migrate. The configuration migration can be reversed.

Perspectives Configurations

This preferences page allows you to adjust the behaviour of the perspectives when a program is run.

Here you can specify:

- The default perspective which will be automatically opened, when a program is launched.
- The default perspective which will be automatically opened, when a program is suspended.

You can also specify whether:

- The default perspectives should always be opened automatically.
- The default perspectives should never be opened automatically.
- The Studio should ask you each time whether you want to open a default perspective.

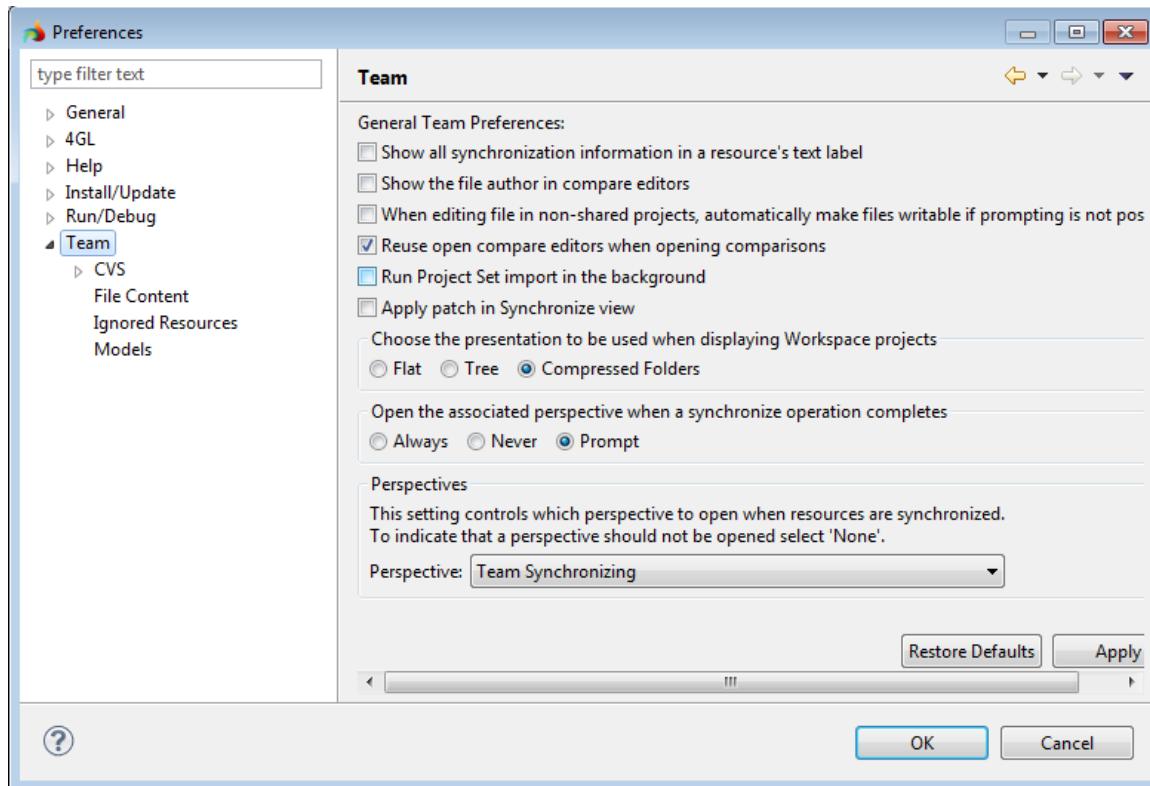
View Management Preferences

This preferences page is used to define which views will support automatic view opening. Here you can select the perspectives which support automatic view opening. By default the views which have been manually opened or closed are not closed or opened automatically, but you can remove the check mark from this option to allow automatic closing/opening of the views.



Team Preferences

This preferences page contains options which affect the version management Team support.



The default values for the general team settings are shown in the picture above. Some of these options are described in the table below:

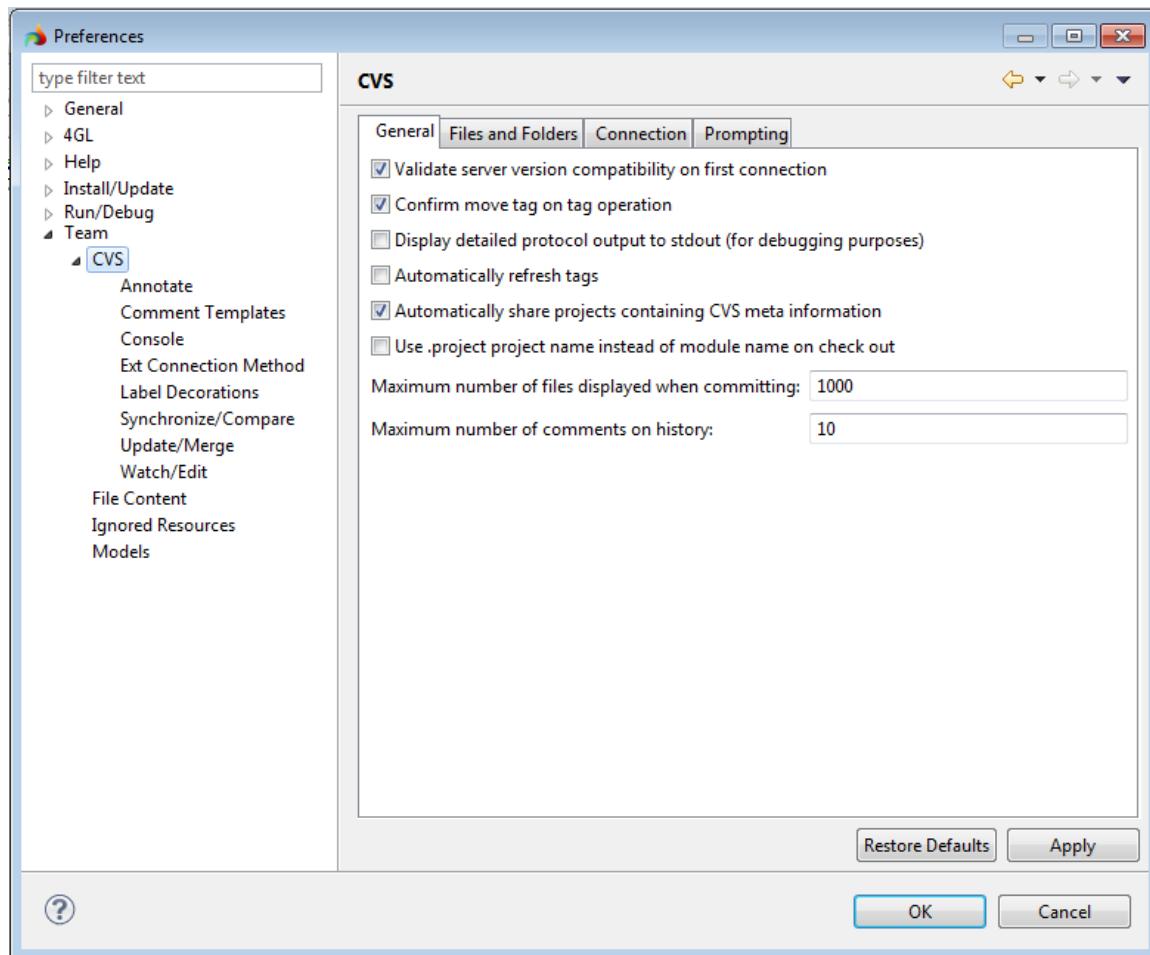
Option	Description
Show all synchronization information in a resource's text label	Select to have the synchronization state of a resource displayed as text in the resource's label. Otherwise only the icon decorator will be used to show the synchronization state.
Show the file author in compare editors	Select to display the file author in the compare editors opened on a repository provider (if this option is supported by the repository provider)
Reuse open compare editors when opening comparisons	Select if you want a new comparison be opened in the existing compare editor to decrease the number of opened editors.
Choose the presentation to be used when	Select to configure the default layout used when a synchronization issue is first added to the Synchronize view. The layout can be changed by means of the view drop down



displaying Workspace projects	menu at any time.
Open the associated perspective when a synchronize operation completes	This option specifies whether the associated perspective should be opened when the synchronization operation is run.
Perspectives	Here you can choose the perspective which is opened automatically when the synchronization operation is run.

CVS Preferences

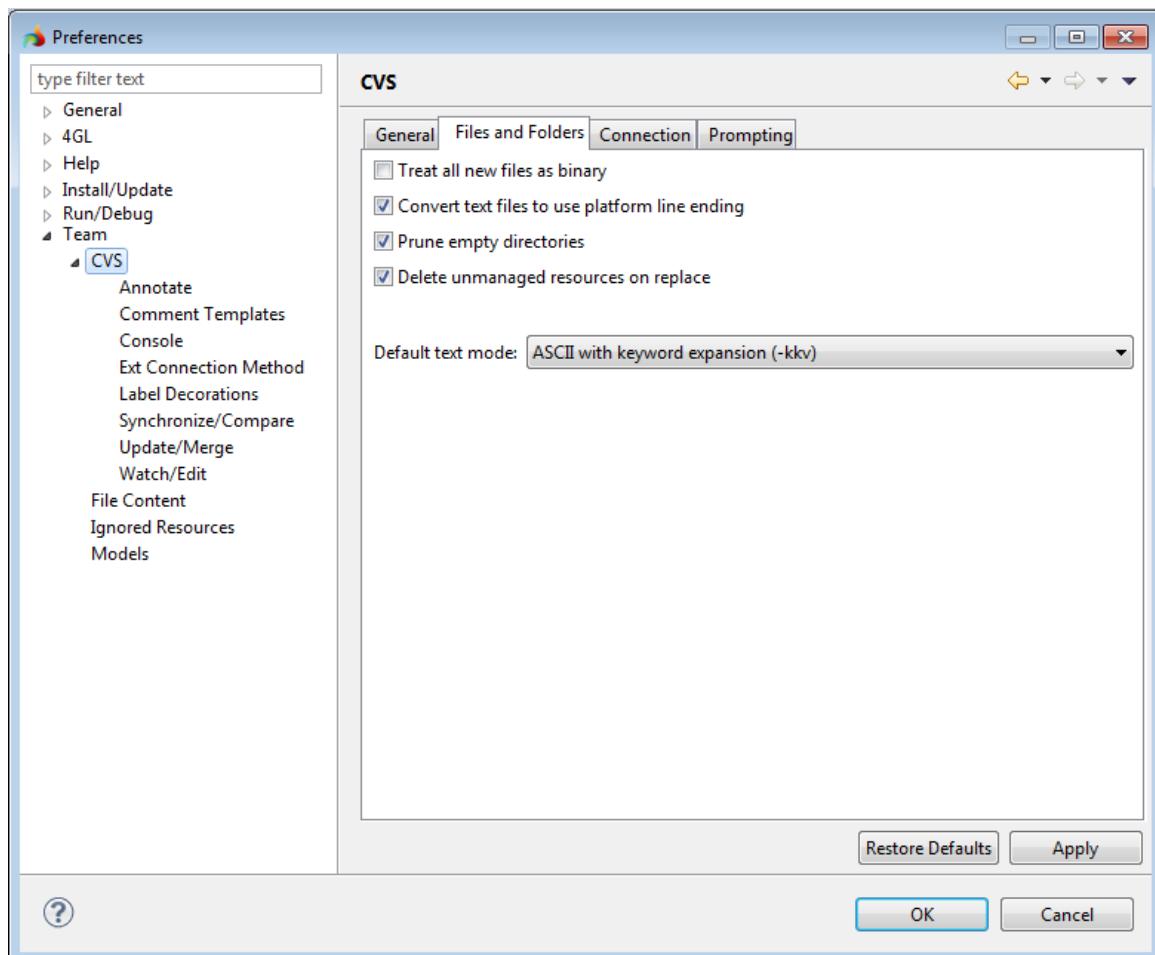
You can customize the settings related to a number of CVS team views and operations with the help of this preferences page. This page contains four tabs. The "General" tab deals with the general CVS configurations:





Option	Description
Validate server version compatibility on first connection	Select to enable a query of the CVS server version on the first connection to determine server compatibility. The server version number will be output to the console, a warning message will be produced, if the server is not compatible.
Confirm move tag on tag operation	Select to be prompted when the Move tag option is chosen when tagging.
Display detailed protocol output to stdout	Select to display the communication trace between the Studio and a CVS server.
Refresh tags when comparing or replacing tags	Select to have the Compare with and Replace With tag dialogues refresh the known tags.
Automatically share projects containing CVS meta information	Select to have any imported project that was checked out from CVS using a different CVS tool automatically shared with CVS.
Use .project project name instead of module name on check out	Select use the project name stored in the .project file, if it is available. Otherwise the module name will be used.
Maximum number of files displayed when committing	Here you can set the limit for the number of files that are displayed in the commit dialog.
Maximum number of comments on history	Here you can set the limit for the number of comments that are displayed in the commit dialog

The “Files and Folders” tab contains several additional CVS options:



Option	Description
Treat all new files as binary	Select to override the file content settings and treat all newly created files as binary files.
Convert text files to use platform line ending	Select to convert the line endings of text files to the line ending used by the platform. If you are checking out resources to a *nix drive that is mounted on a Windows machine, the check mark may be removed. This setting also applies to the CVS control files.
Prune empty directories	Select if you want empty directories to be removed on update. This option may prove useful, while CVS does not provide a client with the ability to remove directories from the server manually.
Delete unmanaged	Select to enable deleting of the resources not under CVS



resources on replace	control when replacing with resources from the repository.
Default text mode	Here the default keyword substitution for the text files can be specified.

The “Connection” tab allows you to adjust connection settings. Here the connection timeout, the compression ratio and the quietness level can be set.

The “Prompting” tab contains the settings which define which actions will invoke prompts.

The CVS preferences section contains a number of other preference pages, they are:

- The **Annotate page** allows you to configure whether the annotate operation should attempt to annotate a binary file.
- The **Comment Templates page** allows you to create, edit and remove comments that will show up in the comments section of the commit dialog.
- The **Console page** allows you to configure the CVS console.
- The **Ext Connection Method page** allows you to configure the EXT connection method.
- The **Label Decorations page** allows you to configure the decorations of labels and icons.
- The **Password Management page** allows you to manage passwords stored on disk in the CVS repository.
- The **Synchronize/Compare page** allows you to configure various aspects of CVS synchronizations and comparisons.
- The **Update/Merge page** allows you to customize how CVS handles updates and merges.
- The **Watch/Edit page** supports configurations for CVS watching and editing



File Content Preferences

This preferences page can be used to associate file names or extensions with the type of data the file contains. You can choose either binary content or ASCII text content.

The screenshot shows the 'File Content' preferences page within a 'Preferences' dialog. The left sidebar lists various categories like General, 4GL, Help, etc., with 'File Content' selected. The main area is titled 'File Content' and contains a table with two columns: 'Name/Extension' and 'Content'. The table lists numerous file types and their content types, such as *.api_filters (ASCII Text), *.bmp (Binary), *.cat (ASCII Text), *.class (Binary), and so on. On the right side of the table are four buttons: 'Add Extension...', 'Add Name...', 'Change', and 'Remove'. Below the table is a note: 'Note that some of these entries are built in and can only be changed, but not removed.' At the bottom are 'Restore Defaults' and 'Apply' buttons, along with 'OK' and 'Cancel' buttons at the very bottom.

Name/Extension	Content
*.api_filters	ASCII Text
*.bmp	Binary
*.cat	ASCII Text
*.class	Binary
*.classpath	ASCII Text
*.cvsignore	ASCII Text
*.dadx	ASCII Text
*.dll	Binary
*.doc	Binary
*.dtd	ASCII Text
*.ecore	ASCII Text
*.ecore2ecore	ASCII Text
*.ecore2xml	ASCII Text
*.emof	ASCII Text
*.ent	ASCII Text
*.exe	Binary
*.exsd	ASCII Text
*.factorypath	ASCII Text
*.genmodel	ASCII Text
*.gif	Binary
*.history	ASCII Text
*.htm	ASCII Text
...	ASCII Text

Entries are added to the File Content preferences page in two ways:

- The Studio itself defines the file content type for file extensions that are common and appear frequently in the workbench (e.g. 4gl, 4fm, etc.).
- The file content types can be added explicitly on the File Content preference page.

Only those entries that were manually added can be removed. Entries contributed by the Studio can only be changed.

Ignored Resources Preferences

You may wish to exclude certain resources from version control. Resources which names match any pattern ticked off in the list on this preferences page will not be released. The wildcard characters '*' and '?' are permitted. Examples: '/*/a/*' matches all files and sub-folders in '/x/a/'



and also in '/x/y/z/a', '/?/a/*' matches all in '/x/a', '/?/a/*.doc' matches all .doc files in '/x/a/' folder and its sub-folders. These settings will not affect resources already under version control.

Index

Building	
automatically	97
configurations.....	100
tools	90, 95
working sets.....	92
Command line tools	
4make	158
qbuild	159
qxpt	159
qfgl.....	156, 319
qform.....	158
qlink	156, 320
qmsg	159
qrun	157
qxcc.....	266
qxld	266
qxlm	13
wsact	322
wslink	321
wsmdc	320
Creating	
a new project	80
from existing resources	77
libraries.....	133
medis files.....	87
source files.....	85
CVS repository	
Checking out	307
committing	314
connecting	292
sharing	304
updating	315
Debugging	
managing breakpoints.....	234
setting breakpoints	221
step-by-step execution.....	223
viewing values.....	230
Encryption	274
Environment settings	
\$QUERIXDIR	270
environment variables.....	254
fglprofile	255
Export/import	
archive files.....	62, 73
complete existing projects	46
file system.....	56, 68
'HydraStudio' V4 project.....	41
programs and libraries	52, 74
Form widgets	
browser	179
button.....	177, 200
calendar.....	178, 198
check box.....	177, 197
combo box	178, 194
dynamic label	176, 183
function field	178
hotlink	178, 202
image	177, 187
line/box	176, 181
radio button	177, 193
screen array/grid	179, 206
screen records.....	203
static label.....	176, 180
text field	176, 184
GUI Server Security	
Classic Mode	272
Full Mode	272
Launching a program	
in debug mode	222
in run mode	110
Libraries	
create a 4GL library	133
external 4GL.....	140
external 4GL libraries	140
external dynamic C.....	146
external static C.....	144
internal	138
listener	
definition.....	270
installation	270
LyciaStudio	
editors	28
menu	20
perspectives	24, 31
preferences	326
project properties	21
toolbar	22
views	19, 24
Supported databases.....	4
Supported operating systems	4

