



# Getting started with g9

**g9**

Version 2.9

August 12, 2020

# Table of Contents

1	Audience .....	5
2	Introduction .....	6
3	g9 Documentation .....	8
4	Context sensitive help.....	10
5	Installation .....	11
5.1	Prerequisites .....	11
5.2	Installing g9 .....	11
5.3	g9 Trial License.....	11
5.4	Installing Apache Maven .....	13
5.5	Install g9 Java runtime using Maven install scripts.....	14
6	Creating the Quickstart Swing Project.....	15
6.1	Browsing g9 .....	16
6.1.1	Browsing the qs-guimodel Project.....	17
6.2	Java Swing Code Generation.....	19
6.2.1	Prerequisites .....	19
6.2.2	Swing Build Properties and Code Generation .....	19
6.2.3	Compiling and running your Swing application.....	22
6.3	Modeling your Dialogs .....	24
6.3.1	Modify the Resources File .....	24
6.3.2	Create an Object Selection .....	26
6.3.3	Generate the Dialog Model .....	28
6.3.4	Edit the Dialog Model.....	30
6.3.5	Add an Event .....	35
6.3.6	Edit a Dialog Box Model .....	37
6.3.7	Compile and rerun .....	38
7	Creating the Quickstart ReactJS Project .....	40
7.1	Javascript ReactJS Code Generation.....	41
7.1.1	Prerequisites for ReactJS .....	41
7.1.2	React Build Properties and Code Generation.....	41
7.1.3	Compiling and running the React application.....	44
7.1.4	Edit the Dialog Box Model.....	48
8	Creating the Quickstart ICEfaces Project.....	50

---

8.1	JSF ICEfaces Code Generation .....	51
8.1.1	Prerequisites for ICEfaces.....	51
8.1.2	ICEfaces Build Properties and Code Generation .....	51
8.1.3	Compiling and running the ICEfaces application .....	54
8.1.4	Edit the About Dialog Box Model.....	57
9	Working with the Xcore domain model .....	60
9.1	Synchronizing changes.....	60
10	Creating Derby Database .....	61
10.1	Installing Apache Derby Database .....	61
10.2	Creating the schema and Derby database.....	61

This manual is intended to provide an introduction to the functionality and features of g9 capabilities and demonstrate modeling and code generation with g9. g9 is a domain driven application development tool. It consists of a series of plug-ins used in the Eclipse open development platform providing a modern integrated development environment (IDE).

This manual also includes a tutorial that guides you through certain tasks required to build an application. You will import g9 example projects (QuickStart) that is based on the domain model of a fictitious record shop. You will then prototype dialog models and generate Java code for the user interfaces and the services needed.

The tutorial is not meant as a guide for implementing a real-world application. Rather, it is meant to show the power and flexibility of g9 for prototyping user interfaces and for generating code.

# 1 Audience

The audience for this manual is an experienced architect, analyst or software developer who is familiar with their application domain. It is designed for those who are new to the g9 product.

The guide will review certain Eclipse features and direct users to the appropriate Eclipse documentation for further information.

You should, however, have a general knowledge of object-oriented coding practices and concepts and user interface design. Also, since g9 generates code for the Java target platform, knowledge of Java is necessary.

## 2 Introduction

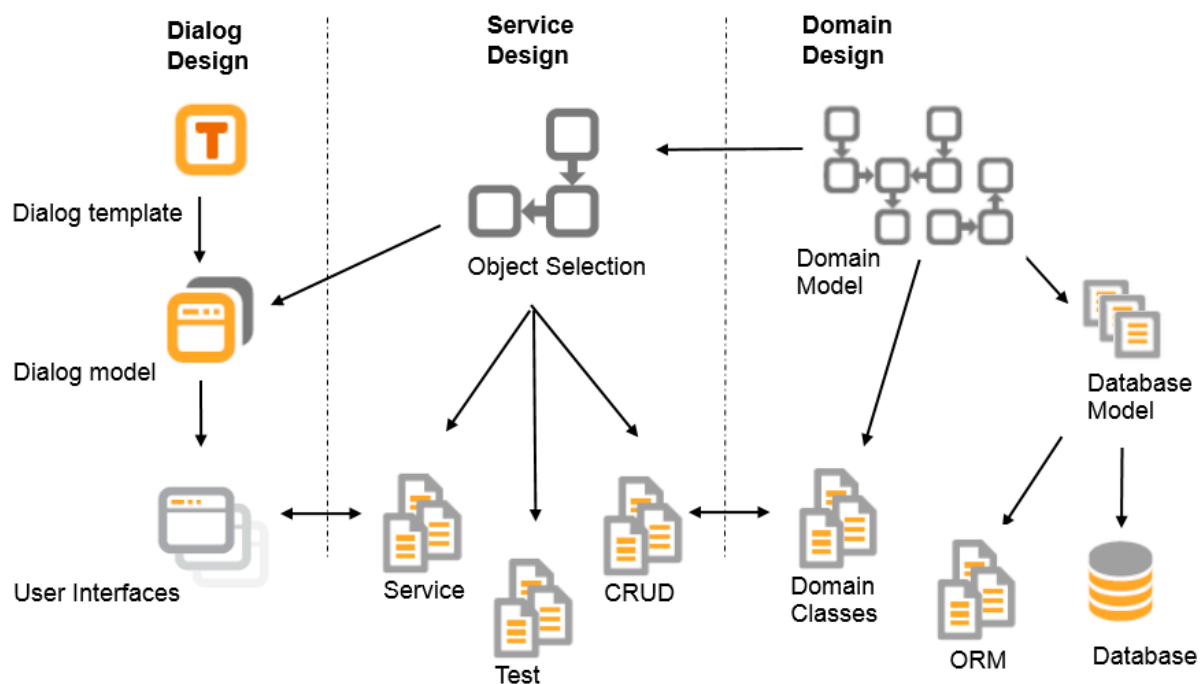
Following the domain driven development approach to software development, *g9* is a development environment that can manage the complexities of application development. It can import a domain and application model into its workspace and add abstract data and user interface modeling (dialog modeling).

It is written as a plug-in for the Eclipse development platform. Eclipse is an open source platform that has multiple projects that provide extensible framework, tools and runtimes for building, deploying and managing software across the life-cycle.

With *g9* you can build complete java enterprise applications without any other programming than your necessary business logic. Some of the *g9* concepts are:

<b>g9 model artifacts</b>	<b>Description</b>
Domain model	Application concepts defined as domain/class models in Enterprise Architect UML, EMF/Xcore or Java source Domain models. The model is synchronized into the <i>g9</i> workspace.
Database model	Representation of the persistent part of the domain model suitable for database modeling.
Object Selection	Subset of the Domain model. Selected objects and association roles from the domain model. The subset supports a use case/task.
Dialog template	Resources and guide lines for user interface design.
Dialog model	User interface design based on Object Selections and Dialog templates.

Many *g9* concepts are introduced and briefly described in the Getting started documentation.



From the models, g9 can generate many artifacts where the generated results may form a complete running application from client to database.

Generated artifacts	Description
Domain Classes	Java domain classes with or without JPA
Database	Database schemas
ORM	Object-relational mappings
CRUD	Java CRUD services on any subset of the domain model
Service	Web Services or REST services
User Interfaces	Clients based on ReactJS, Swing or JSF/ICEfaces

Some of the benefits of using g9 include increased productivity from generating code from models, better maintainability as newer technologies can be more easily adopted, and better consistency of code.

### 3 g9 Documentation

g9 documentation is available at *Help>Help Contents*. Open the **g9 Documentation** node:

g9 Documentation > User Guide

## Introduction

Following the model-driven approach to software development, *g9* is a development environment that can manage the complexities of application development. It can import a domain and application model into its workspace and add abstract data and user interface modeling (dialog modeling).

It is written as a plug-in for the Eclipse development platform. Eclipse is an open source platform that has multiple projects that provide extensible framework, tools and runtimes for building, deploying and managing software across the lifecycle.

With *g9* you can build complete java enterprise applications without any other programming than your necessary business logic. Some of the *g9* concepts are:

g9 artifact	Description
Domain model	Application concepts defined as domain/class models in Enterprise Architect UML, EMF/Xcore or Java source Domain models. The model is synchronized into the <i>g9</i> workspace.
Database model	Representation of the persistent part of the domain model suitable for database modeling.
Object Selection	Subset of the Domain model. Selected objects and association roles from the domain model. The subset supports a use case/task.
Dialog template	Resources and guide lines for user interface design.
Dialog model	User interface design based on Object Selections and Dialog templates.

Many *g9* concepts are introduced and briefly described in the [Getting started with g9](#).

The documentation consists of:

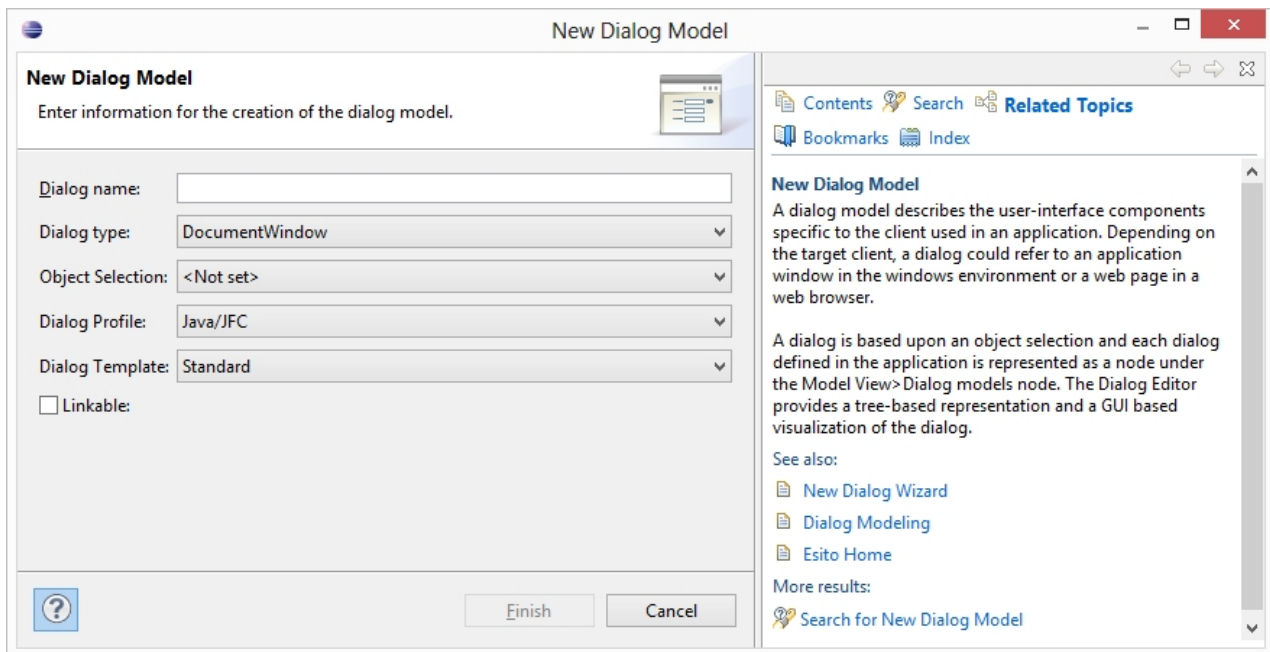
- Getting started:
  - Getting started with g9: Describes how to start with g9.
    - Creating the Quickstart ReactJS Project
    - Creating the Quickstart Swing Project
    - Creating the Quickstart ICEfaces Project
  - From database to running application.
  - Getting started with Custom Generators. See how you can create your own code generator.
  - Getting Started with WSg: Describes how to start with WSg, a subset of g9. WSg is available under a free to use license.
- User Guide: Concepts, Tasks and Reference
  - Modeling with Ecore/Xcore, Enterprise Architect, Java code as Model Source or WSDL/XSD.
- Programmer Guide: How to use the generators and program towards the g9 run-time.
- General documentation:
  - Dialog Export to Spreadsheet (Excel)
  - Dialog Print (JasperReports)
  - Testing with Jouteur



- Web Integrator (Application Partitioning)
- Web Services Generator
- RESTful Services
- Migration from Genova 8 to g9
- Release Notes
- News and Changes: All news/changes from previous releases.

## 4 Context sensitive help

All g9 wizards have context sensitive help. Use the question mark in the lower left corner:



## 5 Installation

### 5.1 Prerequisites

Before you begin to work with the sample application:

- Install Eclipse IDE for Java and DSL Developers (This comes with Maven integration and Xcore, Xtext and Xtend)
- Install Java Standard Development Kit (JDK) 8 or later. Check that you have the JAVA\_HOME and PATH environment variables set with the installed JDK values.
- [Install g9](#)
- [Install Apache Maven](#) (version 3.1.1 or later)
- [Run Maven Install Script](#) (install g9 runtime)
- These sample projects use Xcore as modeling tool. Eclipse DSL versions support this.
- All example projects use Derby databases. Installation and creation of databases are explained in [Creating Derby Database](#).

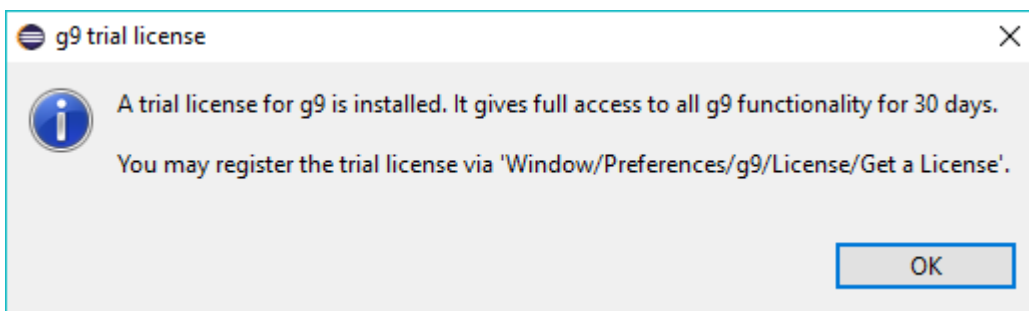
### 5.2 Installing g9

The installation of g9 will be performed using the Eclipse Update site installation.

1. In Eclipse, go to *Help > Install New Software ...*
2. In the *Install dialog*, click the *Add...* button located at the top right of the dialog to add an install site. This brings up the *Add Repository* dialog
3. You can install the most recent version from the remote update site.
  - a. In the *Name* field, type **g9 update site**
  - b. In the *Location* field, type **<http://www.esito.no/updatesite/g9>**
4. Click **OK** to return to the *Install* dialog. Uncheck the '*Group items by category*' and select the g9 checkbox. Then select the **Next** button.
5. Click **Next** to confirm installation
6. Read and accept the license agreement. To continue installing, select "**I accept the terms of the license agreement**" and click **Finish**.
7. Click **OK** in the Security Warning if it is displayed.
8. When prompted to restart Eclipse, click **Restart Now** to restart.
9. Install a [g9 License](#).

### 5.3 g9 Trial License

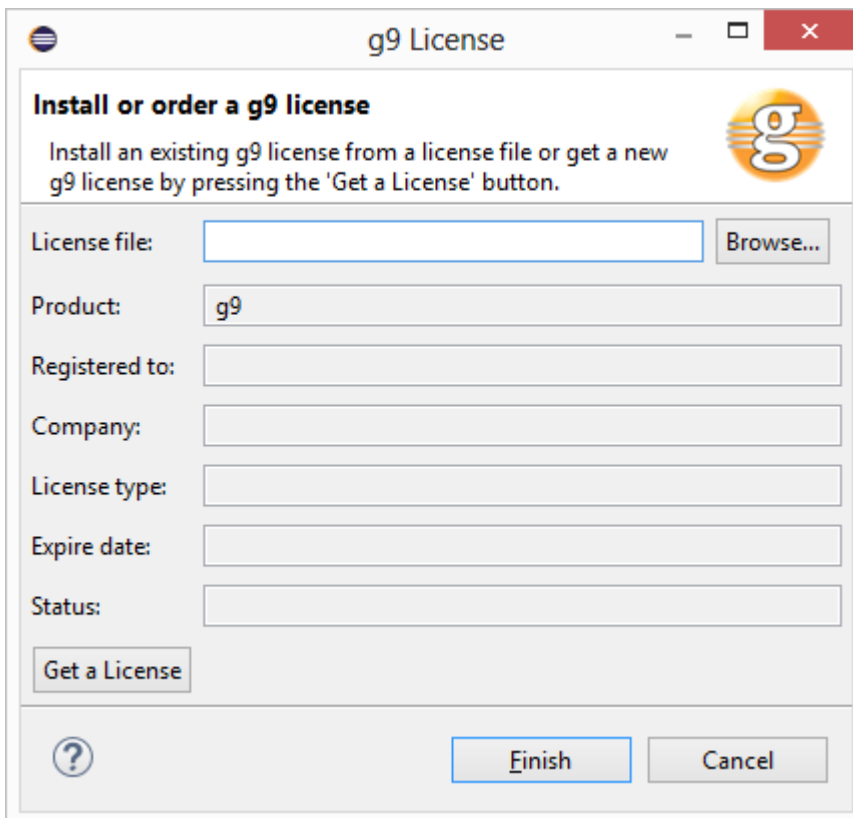
If the g9 plugin does not have a license installed, a trial license will automatically be installed the first time Eclipse starts and any g9 project is created or refreshed. A Message telling about the installation is displayed:



If a license is installed, but is illegal for some reason, an "Install g9 license" message will be displayed.

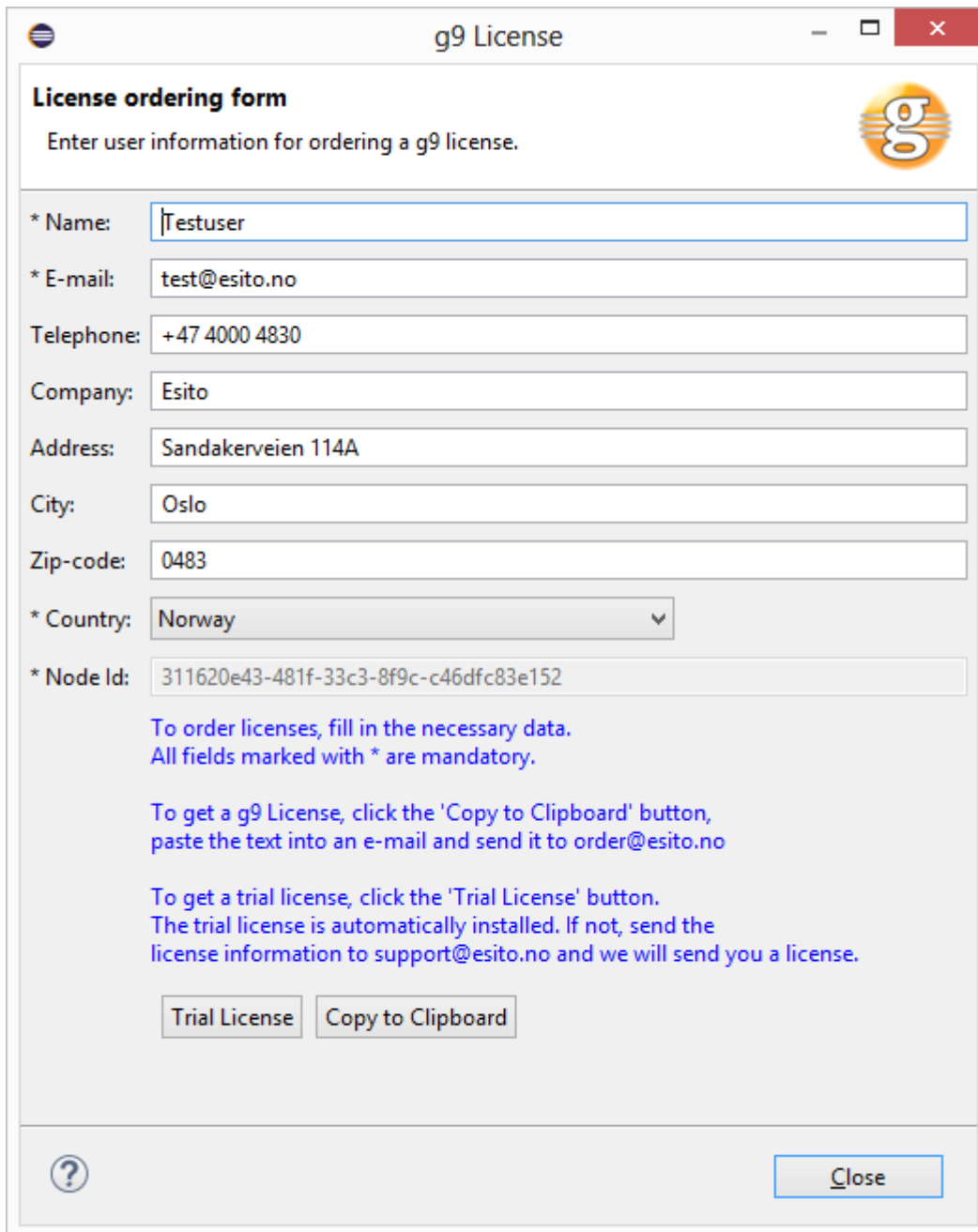


Click Yes to order either a full or trial license.



If you have received a license file, click *Browse* and install it.

If not, click *Get a License*, fill in the fields and click *Trial License*:



The screenshot shows a window titled "g9 License" with a standard Windows title bar. Inside the window, there is a section titled "License ordering form" with a sub-instruction: "Enter user information for ordering a g9 license." To the right of this text is a circular logo with a stylized 'g'. Below the instruction, there are several input fields, each preceded by an asterisk indicating it is mandatory:

- \* Name:
- \* E-mail:
- Telephone:
- Company:
- Address:
- City:
- Zip-code:
- \* Country: - \* Node Id:

Below the input fields, there is instructional text in blue:

To order licenses, fill in the necessary data.  
All fields marked with \* are mandatory.

To get a g9 License, click the 'Copy to Clipboard' button,  
paste the text into an e-mail and send it to [order@esito.no](mailto:order@esito.no)

To get a trial license, click the 'Trial License' button.  
The trial license is automatically installed. If not, send the  
license information to [support@esito.no](mailto:support@esito.no) and we will send you a license.

At the bottom of the form area, there are two buttons: "Trial License" and "Copy to Clipboard".

At the very bottom of the window, there is a footer bar containing a help icon (a question mark inside a circle) on the left and a "Close" button on the right.

A trial license is automatically installed and it will work for a number of days (usually 30 days). See Licenses to read more about it.

## 5.4 Installing Apache Maven

Apache Maven is a software management tool that helps manage a Java project's build. Maven dynamically downloads Java libraries that have a dependency on the generated g9 Java projects. For example, if a project requires the *Hibernate* library, Maven will automatically download the required files and those dependencies that Hibernate itself needs. This makes the configuration of the Java projects much easier.

To install Maven:

1. Go to the <http://maven.apache.org/download.html> site and select the Mirror site for the Maven Binary zip.
2. Create a directory (i.e. `c:\java`) and extract the contents of the zip file.
3. In Windows, you need to set two environment variables in the *Environmental Variables* dialog.

- Create new system variable MAVEN\_HOME  
Variable value: Maven install directory
- Edit the Path system variable and add the following at the end of the Path value: **;%MAVEN\_HOME%\bin**  
(Don't forget the semi-colon)

## 5.5 Install g9 Java runtime using Maven install scripts

The Java code that is generated in g9 relies on various g9 runtime jar files that are available from the Maven Central repository.

If you can't access the Maven Central, you may install these jar files: an install script is available that copies the required files into the Maven local repository. (The default value is `${user.home}/.m2/repository/`). You must ensure that Maven has already been installed and configured.

To run the install script:

1. In Eclipse, select *Preferences* and the *g9* page.
2. Click on the link: *Installation of g9 runtime files*. It will copy the install script location into clipboard.
3. Start a command shell by clicking Start and then typing CMD and press enter
4. Write "cd " and paste the text from the clipboard into the command shell. It will look like this (dependent of eclipse installation path and g9 version):  
**cd c:\eclipse\plugins\no.esito.g9.runtime\_2.5.0.v201606220934\jar\mvn**
5. Run the install script by typing **maven-install** and press enter
6. Once it is finished, you may close the command window

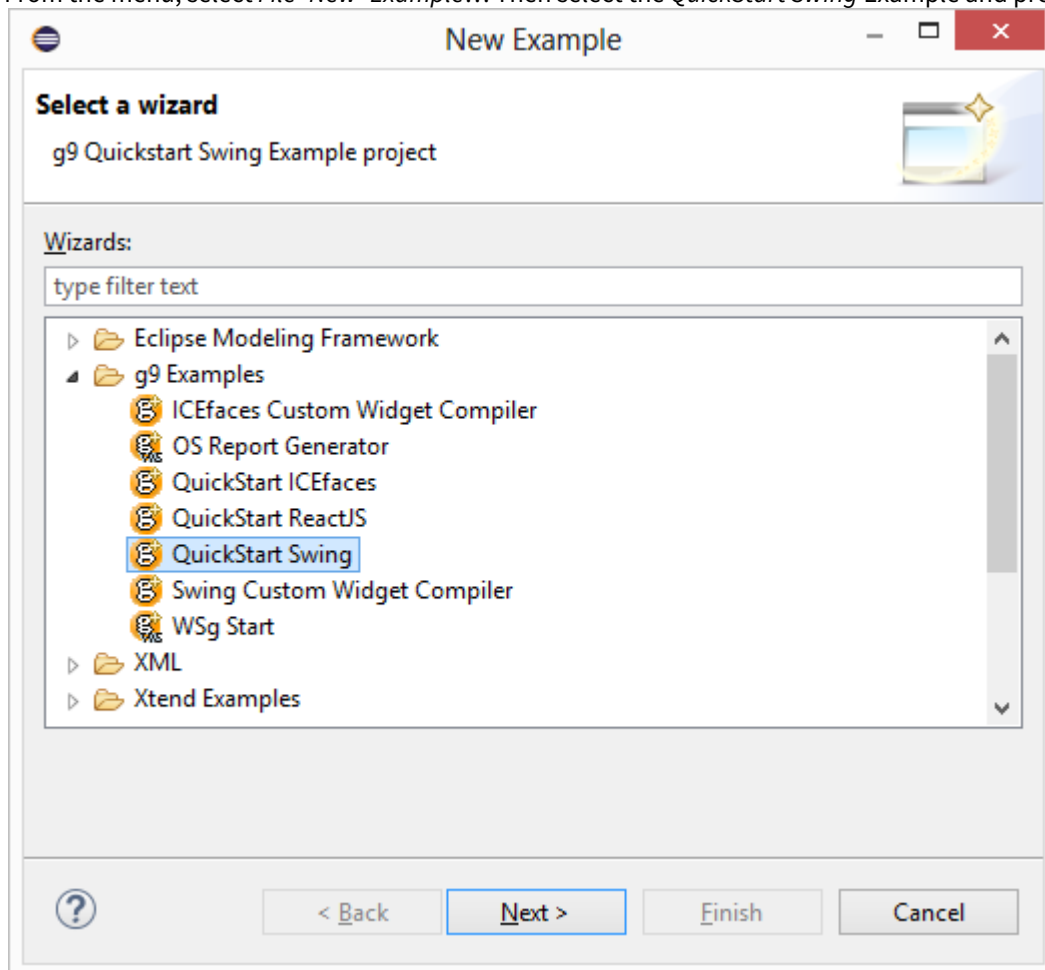
## 6 Creating the Quickstart Swing Project

A project is where the g9 objects are stored including the domain class model, objects selections and dialog models. To create the Quickstart Swing project, you can import an existing project into the workspace.

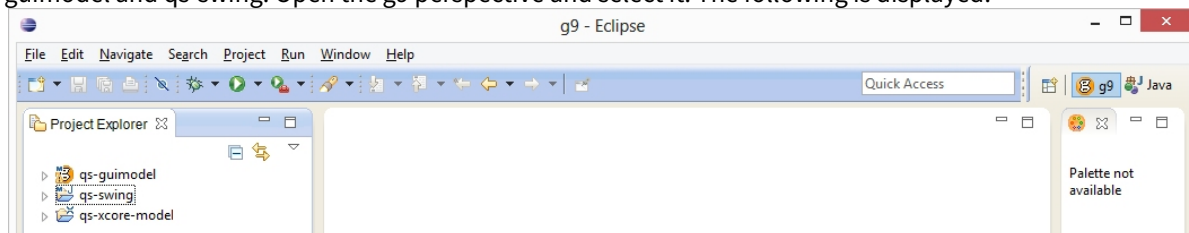
In order to quickly get you started, the *Quickstart Swing* project contains an application based on an existing domain model designed using *Xcore*. This project contains predefined database mappings, object selections and dialog models, as well as all the necessary generated Java code for running parts of the application in a Java Swing client.

To create the Quickstart Swing projects:

1. From the menu, select *File>New>Example...* Then select the *QuickStart Swing* Example and press *Next*.



2. Press the **Finish** button. Several projects are opened and showed in the explorer: *qs-xcore-model*, *qs-guimodel* and *qs-swing*. Open the g9 perspective and select it. The following is displayed:



3. The *Quickstart* project *qs-guimodel* contains the model which includes the Domain Classes, the Object selections and Dialog models. The other projects are the definition of the *Xcore* domain model (*qs-xcore-model*) and a Java project that have been created for the generated code (*qs-swing*).

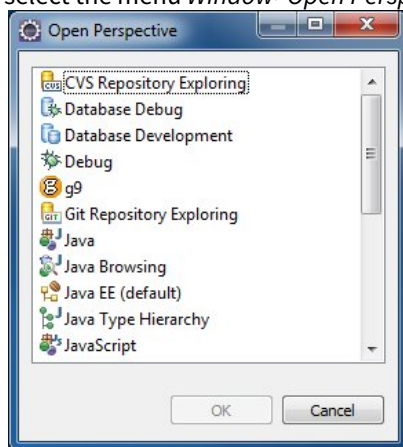
4. The system should begin building the project automatically and display the progress in the status bar. You might notice error markers (red x) next to the projects. These should disappear once the system finishes building the projects and Maven resolves the dependency issues.
5. If there are still error markers, it is possible the build did not occur automatically or correct. Close and Open all projects. If there are still error markers, you can select the menu option *Project > Build* or *Project > Clean*. If there are still errors, it is possible Maven did not resolve the dependencies. You can select all the projects then right-click on the project and select *Maven > Update Dependencies*.

## 6.1 Browsing g9

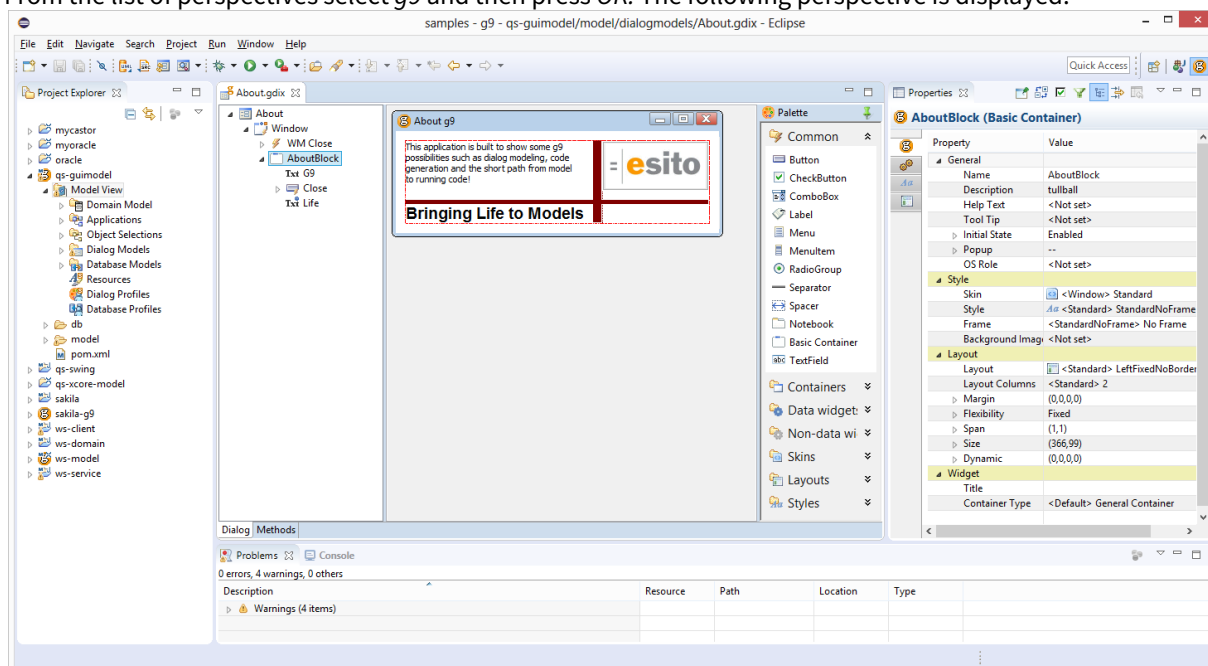
The g9 perspective defines the initial set and layout of *views* in the Workbench window needed to model your application.

To open the g9 perspective:

1. Click the perspective button located at the top left to the left and then select *Other*. You can alternatively select the menu *Window>Open Perspective>Other..*



2. From the list of perspectives select *g9* and then press *OK*. The following perspective is displayed:





This perspective displays the *Project Explorer* view at the left. This is a tree view representation of your projects. The *Quickstart* project *qs-guimodel* contains the various logical views of your application as defined in your domain model. The other projects are Java projects containing generated code. You can right click on any node to get a list of applicable menu options.

The *Properties* view at the bottom is a common view used in Eclipse to display and modify property names and values of a selected item. As you click on various nodes in the *Project Explorer*, the *Properties* view displays a table of properties with their corresponding value.

The *Palette* inside the *Dialog Editor* on the right is used for dialog modeling (discussed later) and contains groups of user interface elements for building user interfaces.

The *Problems* view is used to display problems with your project. This includes both errors and warnings for the various elements such as an invalid attribute value.

The *Console* view displays information related to various actions performed. For example, when you generate code, the console will display those objects that are generated

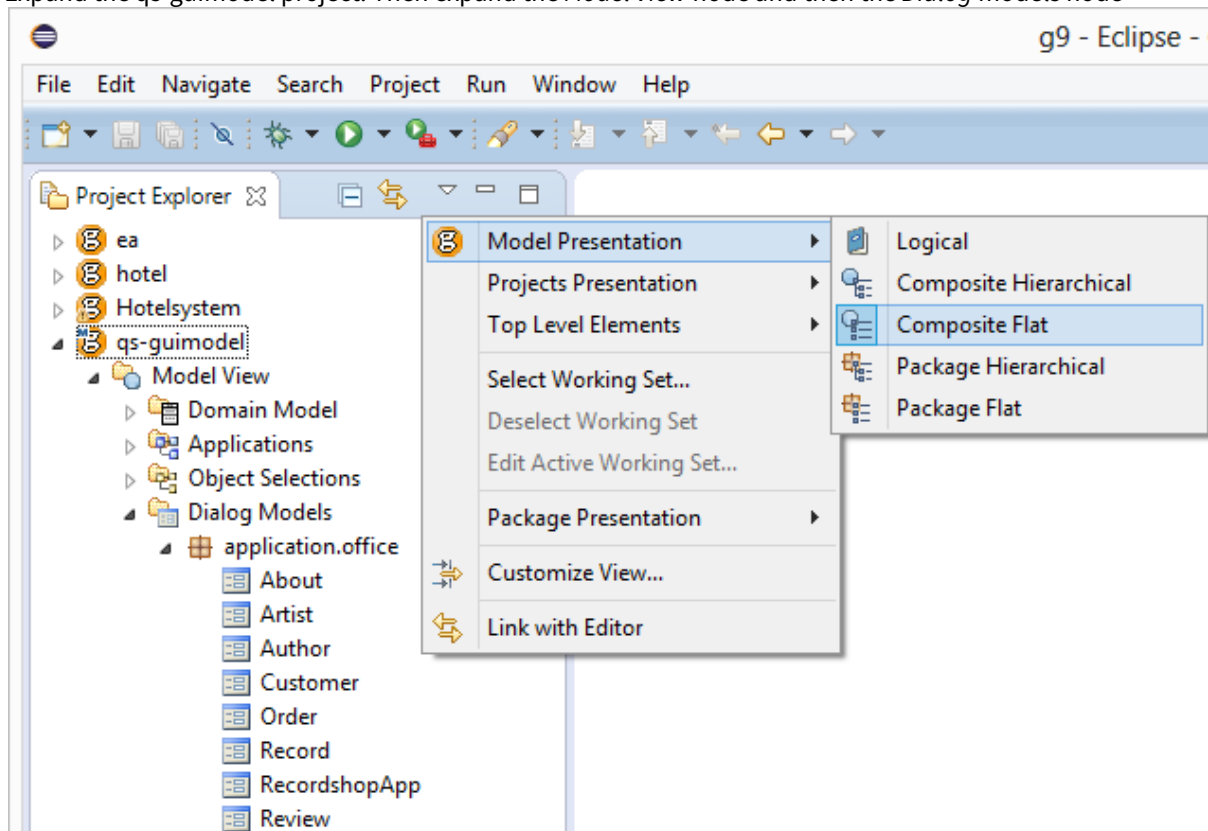
The *Undo History* displays a list of actions that can be undone (e.g. changing attribute values, modifying dialogs)

- ✓ For those new to Eclipse, the views displayed can all be resized by hovering the mouse over the edge of the view and dragging it. Views can also be repositioned by clicking on the tab label and dragging it in the workbench. To reset the perspective, select *Window > Reset Perspective*

### 6.1.1 Browsing the qs-guimodel Project

The g9 project *qs-guimodel* is displayed as a node in the *Project Explorer*.

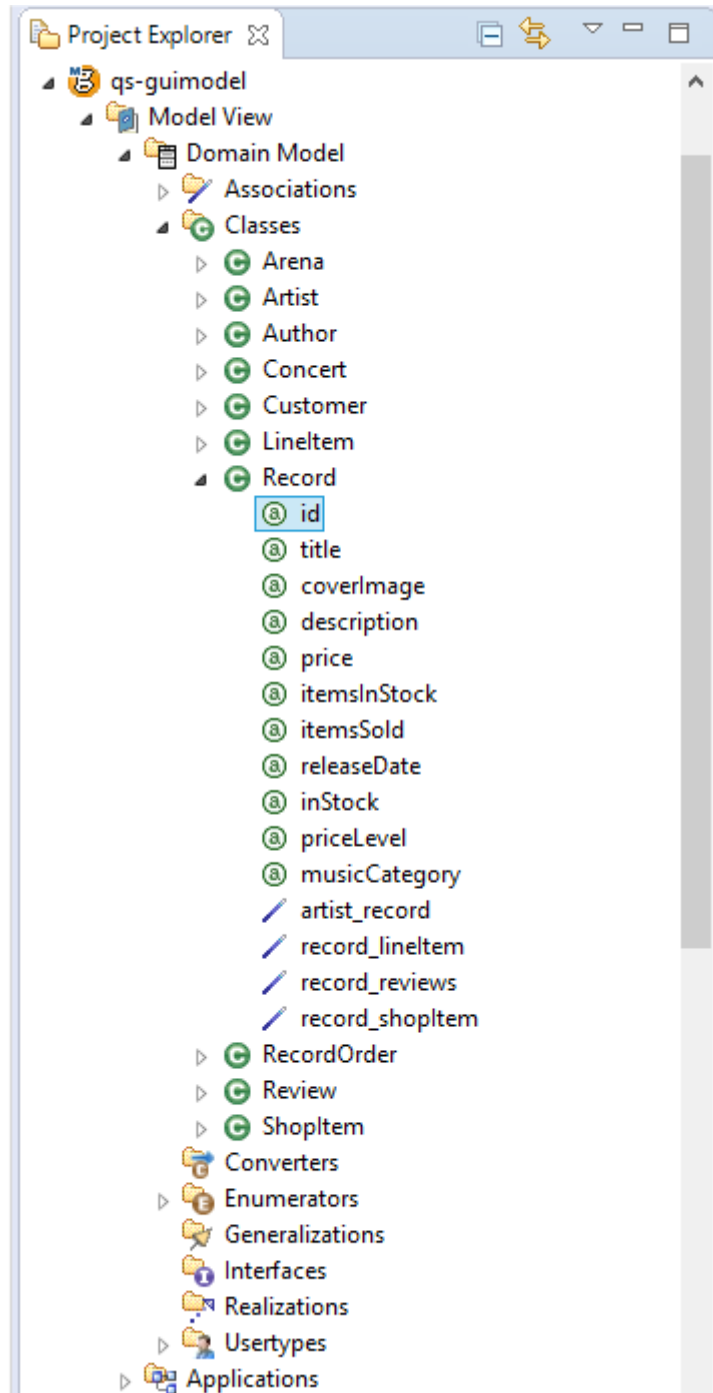
1. Expand the *qs-guimodel* project. Then expand the *Model View* node and then the *Dialog models* node



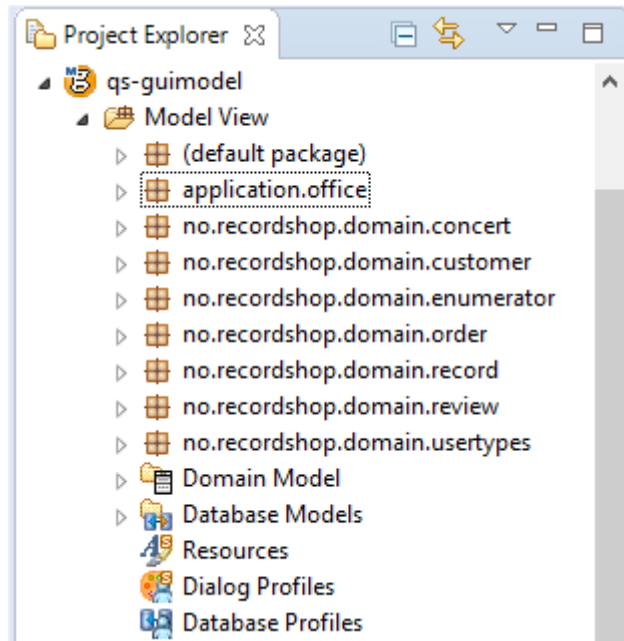
Use the *Project Explorer View* menu  to show package structures. This logical view presentation of the

domain model uses Composite Flat. You can now browse the various elements by expanding the nodes and clicking on a specific element. Note that when you click on an element, the Properties view displays the attributes and their values.

2. Select the View menu in the Project Explorer and select the *Model Presentation > Logical* menu option. See how the Model View are reorganized. Expand the *Domain model* node, then *Classes*, then *Record*. Now click the *id* node in the tree view. The *Properties* view displays the properties defined for that attribute in the domain model.



3. Select the View menu in the Project Explorer view and select the *Model Presentation > Package Flat* menu option. This changes the presentation of the domain model to display objects organized by application package



4. Select the *Model Presentation>Composite Flat* menu option to combine package and model elements view.

## 6.2 Java Swing Code Generation

In this section you will generate client code for a Java Swing client for the dialog that you modeled and then compile and run the application. The sample application already has a working java swing application for other client windows and has already generated code for the various classes, services and database and hibernate mapping files. This sample will still take you through the steps for generating this code.

### 6.2.1 Prerequisites

Necessary pre-installation steps is described in the [Installation](#) part.

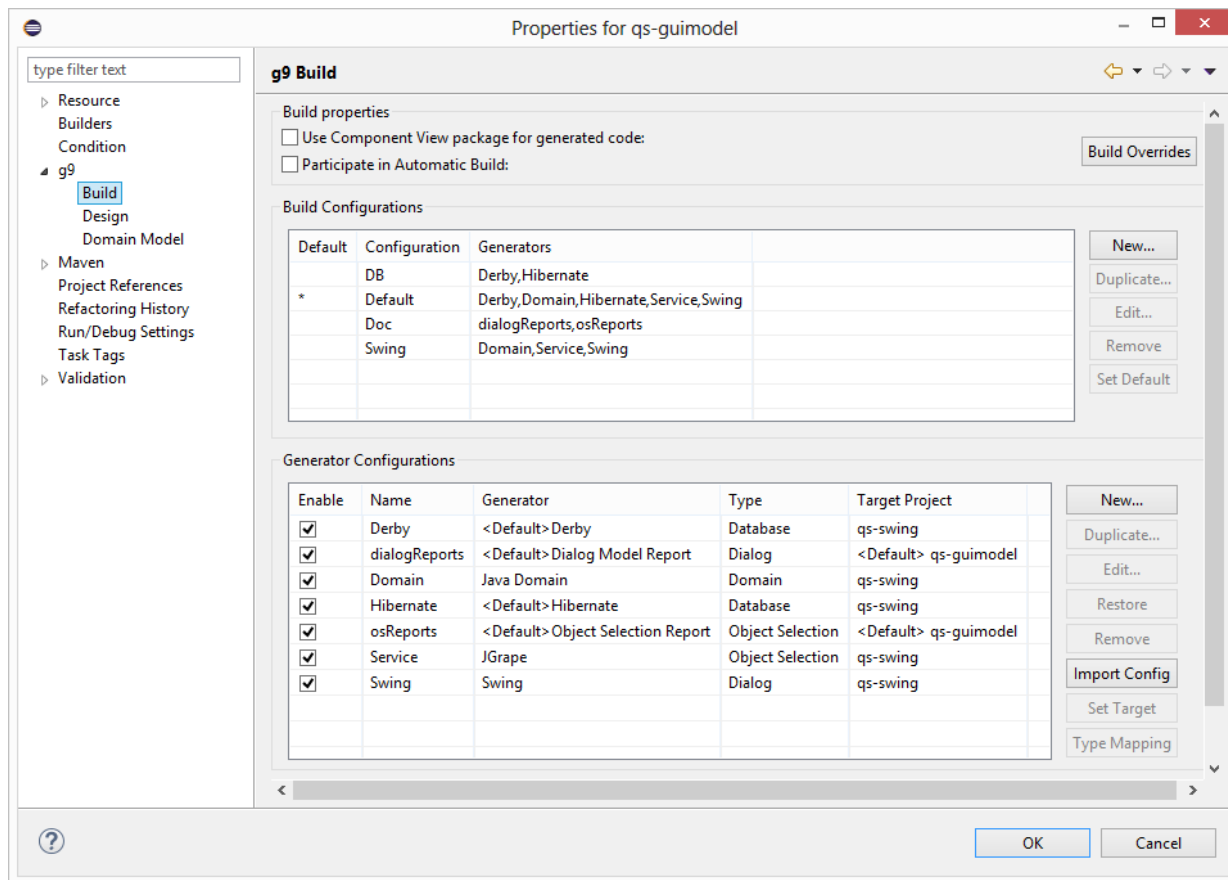
The application code that is generated is dependent on various libraries. These libraries should be automatically installed if you have successfully installed Maven and the Maven Eclipse plug-in. The required libraries are:

- *hibernate core*, along with all dependencies available from <http://hibernate.org> (Requires version 5.1.17.Final or later).
- *spring.jar*, available from <http://www.springframework.org>. (Requires version 4.3.25.RELEASE or later). Some Spring versions depend on *apache commons logging* 1.2 or later, available from <http://commons.apache.org/logging>.
- *log4j*, available from <http://logging.apache.org/log4j/2.x/>. (Requires version 2.11.2 or later)
- *derby.jar*, available from <http://db.apache.org/derby>. (Requires version 10.14.2.0 or later)

### 6.2.2 Swing Build Properties and Code Generation

g9 is structured to allow the generation of domain classes, client code, service code, database schema and hibernate mapping to occur alongside the Eclipse project build. You can configure the system to automatically generate code when Eclipse does a project build, or you can generate code manually. The Build section in the g9 project properties is where you can configure the code generation.

- Click the *qs-guimodel* model project in the *Project Explorer* and then right-click and select Properties. When the Property Dialog appears, expand the *g9* node and then click on the *Build* node. The following dialog is displayed:



## Generator Configurations

You can create multiple *Generator Configurations* for the different components of the application. A *Generator Configuration* is a user-defined term that consists of a supported code generator, a set of parameter settings for that specific generator, and a specified Target Project for separating code among multiple Java projects.

In addition to the *Generator Configurations*, you can create multiple *Build Configurations* which consists of a series of Generator Configurations. These can be associated to g9 objects or be performed on Build/Build Other commands. A default Build is marked with an asterisk and will be performed on a Build command.

If you wanted, you could select the option '*Participate in Automatic Build*' and the code generation would be performed on save of g9 models.

### Generate all in one Build

The default Build Configuration contains all necessary Generator Configurations. Select the g9 project root and push F6 (accelerator for the Build menu). This will build all code in this project.

### Separation of generated and manually maintained code

The generators supports splitting of generated and manually maintained code into two separate source hierarchies. This behavior is triggered by setting the "Source directory" generator parameter to a different value than the "Target directory" parameter. With this behavior, if a file is present in the "Source directory" it will not be generated to the "Target directory". Default values are src-gen/main and src/main.

## Model specific Builds

If you want to see what the specific generators do, you may follow the instructions below. However, all necessary code are generated and ready to run.

### Console View

The Console View might show terminated views. To toggle between different Console Views, select *Display Selected Console* in the View toolbar.

### Domain Class Generator

The *Domain Generator* generates Java code for the domain model which includes java classes, interfaces and enumerations. The *Quickstart Swing* project is configured to generate the domain classes into the *qs-swing* project.

1. Double click the **Domain** Generator Configuration in the bottom list to see the various standard and customized parameters and their values. Note the various parameters. Close the dialog.
2. Right click on the *qs-guimodel/Model View/Domain Model* node in the *Project Explorer* and select the menu option **Build**.
3. Click on the Console View tab. It should display a list of all the generated Java classes.

### Dialog Code Generator

The *Dialog Generator* generates code for the client part of an application. The code generated is based on the dialogs modeled and their object selections. There are one *Dialog Generator Configuration* for the *qs-guimodel* project. The *Swing* configuration uses the Swing generator for generating Swing based client and is configured to generate the classes into the *qs-swing* java project.

1. Double-click the **Swing** configuration to display the parameters. Close the dialog and then close the *Properties for qs-guimodel*.
2. In the *Project Explorer*, expand the *Applications* node and right click on the **RecordshopAdm** node and select **Build**. This starts the code generation process for the application main program and some helper classes.
3. Right click the *RecordshopAdm* node again and select **Build Application > Swing**. This starts the code generation process for the complete client part of the application including application main and all dialogs.
4. Click on the Console View tab. It should display information regarding the code generation. If there is an error, it will be displayed here.

### Service Generator

The *JGrape Generator* generates code for the service part of the application. This includes the code for all necessary CRUD actions. The *qs-guimodel* project is configured to write all service code for the Swing clients into the *qs-swing* java project.

1. Open the *qs-guimodel* properties dialog again and double-click the **Service** configuration to display the parameters. Note the location of the *ServicePackage*. Close the dialog and also the properties dialog.
2. In the *Project Explorer*, right click the *Object Selections* node and select the menu **Build**. This starts the code generation process for the services for the Swing client.
3. Click on the Console View tab. It should display the various code elements that were generated.

### Database Schema Generator

The Database Generator is used to generate database schema scripts to help in creating databases. It works from the database model that has already been provided for the Derby database.

- Open the *qs-guimodel* properties dialog again and double-click the **Derby** configuration to display the parameters. Note the target project is set to *qs-swing* and the target directory is *db*. Close the dialog.

1. Double-click the **Hibernate** configuration to display the parameters. Note the target project is set to *qs-swing* and the target directory is *src-gen/main*. Close the dialog.
2. In the *Project Explorer*, expand the *Model View/Database Models* node and right-click on the *recordshop\_swing* node and select the menu **Build**. This starts the code generation process of the database schema files and the hibernate mapping files.
3. Click on the Console View tab to display the various code elements that were generated.

### Hibernate Mapping Generator

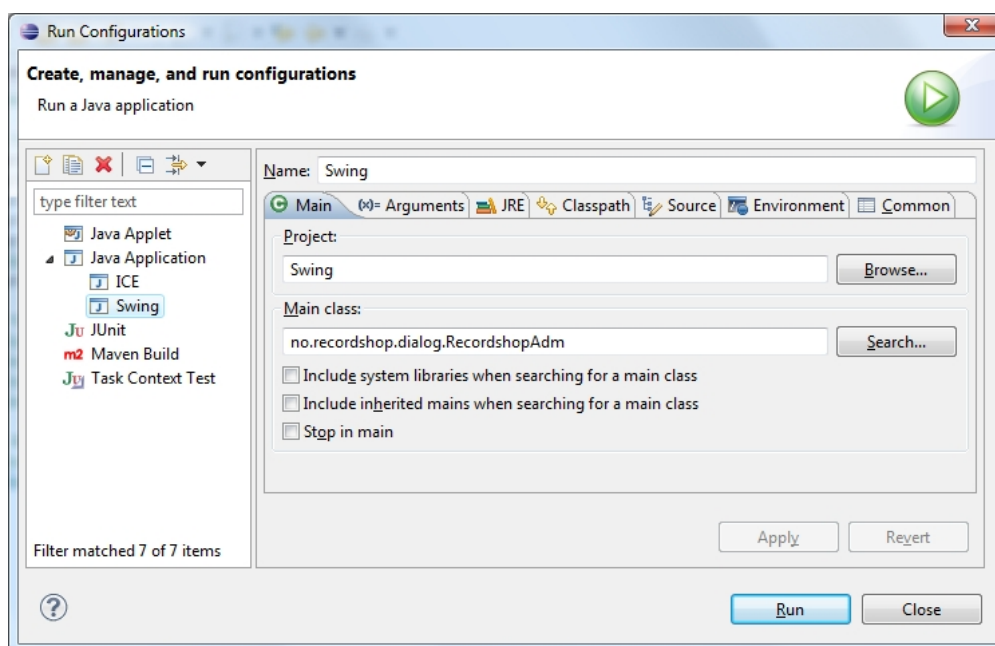
The *Hibernate Generator* is used to generate data access code. The system will generate a mapping file named *<Classname>.hbm.xml* for each domain class specified in the database mapping file. Also created is a *G9Dataaccess.xml* file which contains references to all the domain class mapping files as well as the *hibernate.properties* file which contains JDBC and EJB connection properties.

## 6.2.3 Compiling and running your Swing application

The sample project creates the database populated with data for you and it is stored in the db directory named *recordshop*. You can create/recreate it, see instructions on [Creating Derby Database](#).

You are now ready to compile and run your Swing application.

1. Switch to the Java perspective by clicking on the Java perspective button at the top right. (Or select menu option *Window>Open perspective>Other>Java (default)*)
2. Click on the *qs-swing* java project and press the F5 button to refresh
3. Check the menu option *Project>Build Automatically*. If it is checked, then the code should have automatically compiled. If it is not checked, the select the menu option *Project > Build All*.
4. Select the *Run > Run configurations...* menu option. This opens up the Run Configurations Dialog
5. Expand the *Java Application* node and click the *Swing* node. This has already been created for you.



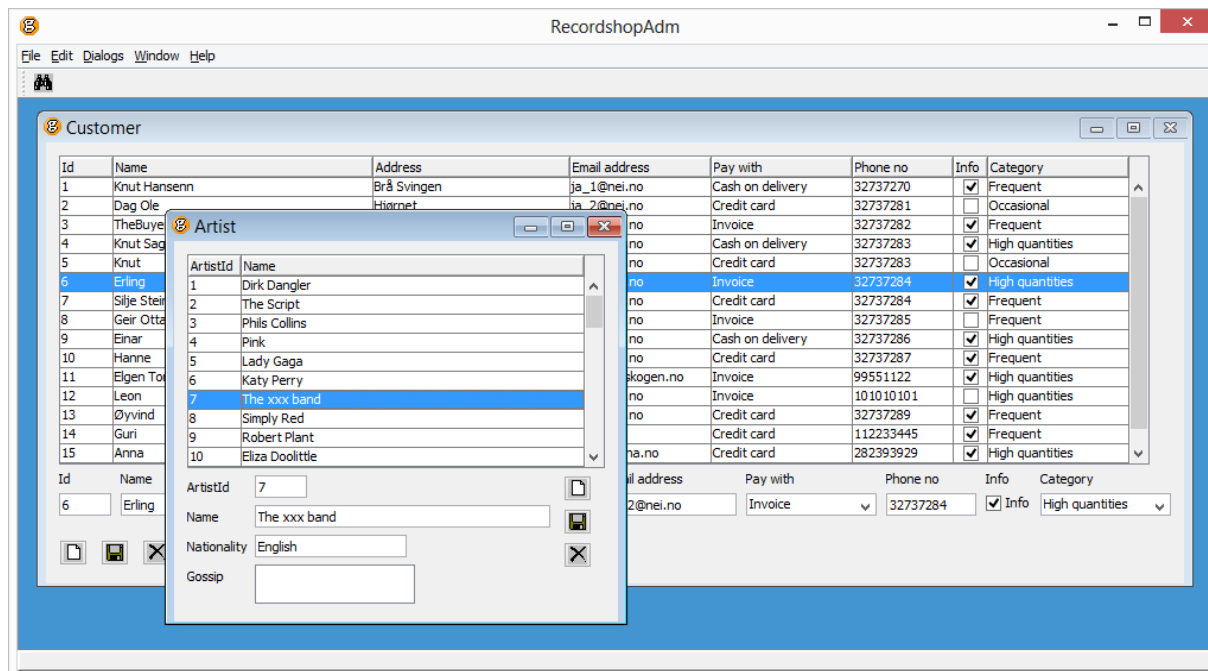
6. Click on the *Classpath* tab and you will notice 3 additional user entries were added. The *qs-swing* project, and the folders *resources* and *images*

7. Press the *Run* button to launch the application:



8. Select the *Dialogs>Customer* and *Dialogs>Artist* menu option

9. Press the Search button in the toolbar:



10. Click the X in the top right corner of the application, or click *File>Exit* in the menu, to close the application.



## 6.3 Modeling your Dialogs

In this section, you will model a user-interface dialog based on an object selection. Even though the dialogs may already be defined in the application modeling editor, you have the option of further refining the dialog or creating a new dialog.

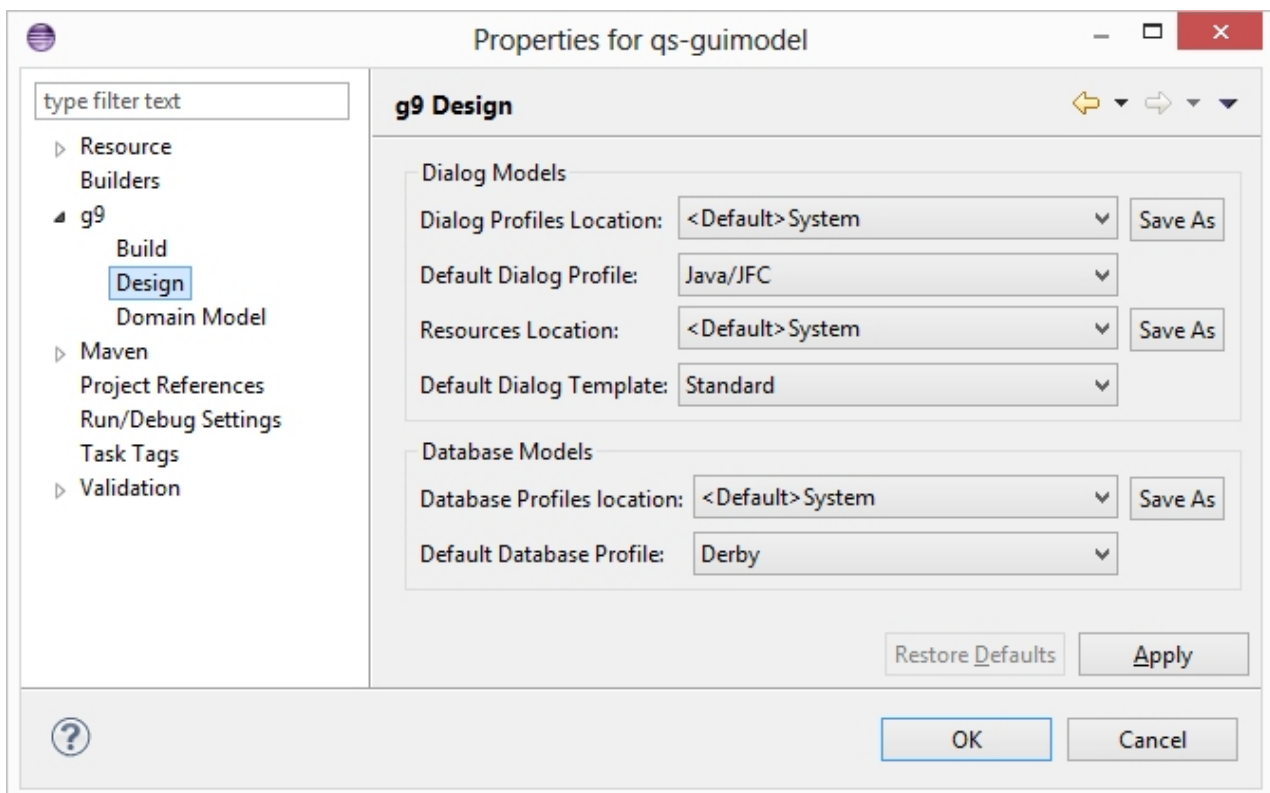
Dialogs are grouped into applications. In the *Quickstart Swing* project, there is one application defined: the *RecordshopAdm* application found in the *Model View < Applications* folder and it contains all the Recordshop back-office dialogs whose intended target is a Java Swing client.

You will perform the following tasks:

### 6.3.1 Modify the Resources File

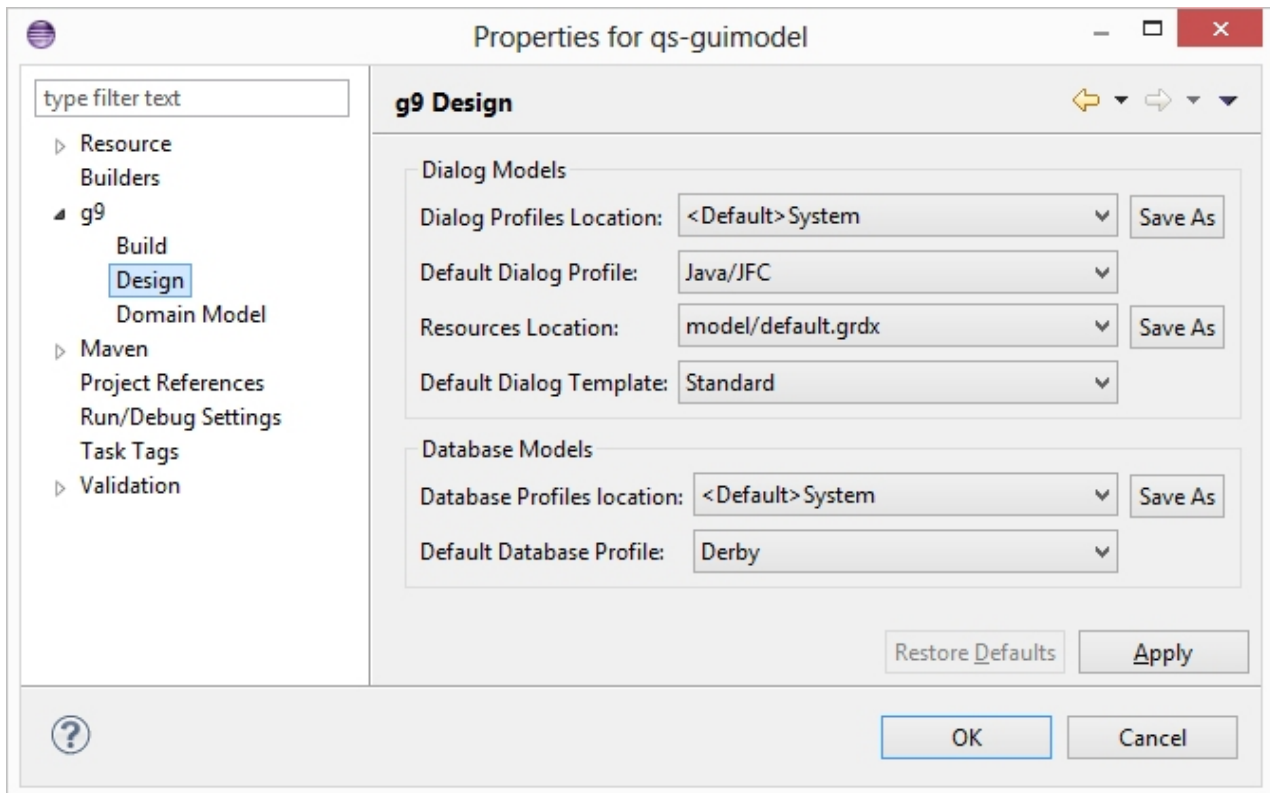
The term *Resource* is used to describe various resources that apply to elements in a dialog. These include *colors*, *fonts*, *styles*, *layouts*, *skins*, *images*, *dialog templates*, *print settings* and *component types*.

g9 maintains an internal list of resources in the system. However, if you need to make a modification or add a new element, you can create your own resource file (.grdx extension). In the *qs-guimodel* project, a resource file is defined by the System Resources. In order to create a local copy, right click the *qs-guimodel* project and select *Properties*. In the Properties dialog, select *g9 > Design*. From *Resources location*, press *Save As* and then *OK*. The Resources file will be stored as *model/default.grdx* in the *qs-guimodel* project.

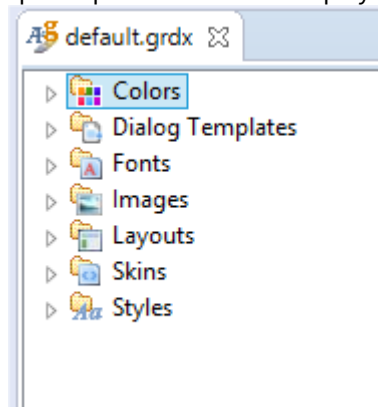


The new Resources is now available and selectable in the *Resources location* combobox. Select the *model/default.grdx* file in the combobox:

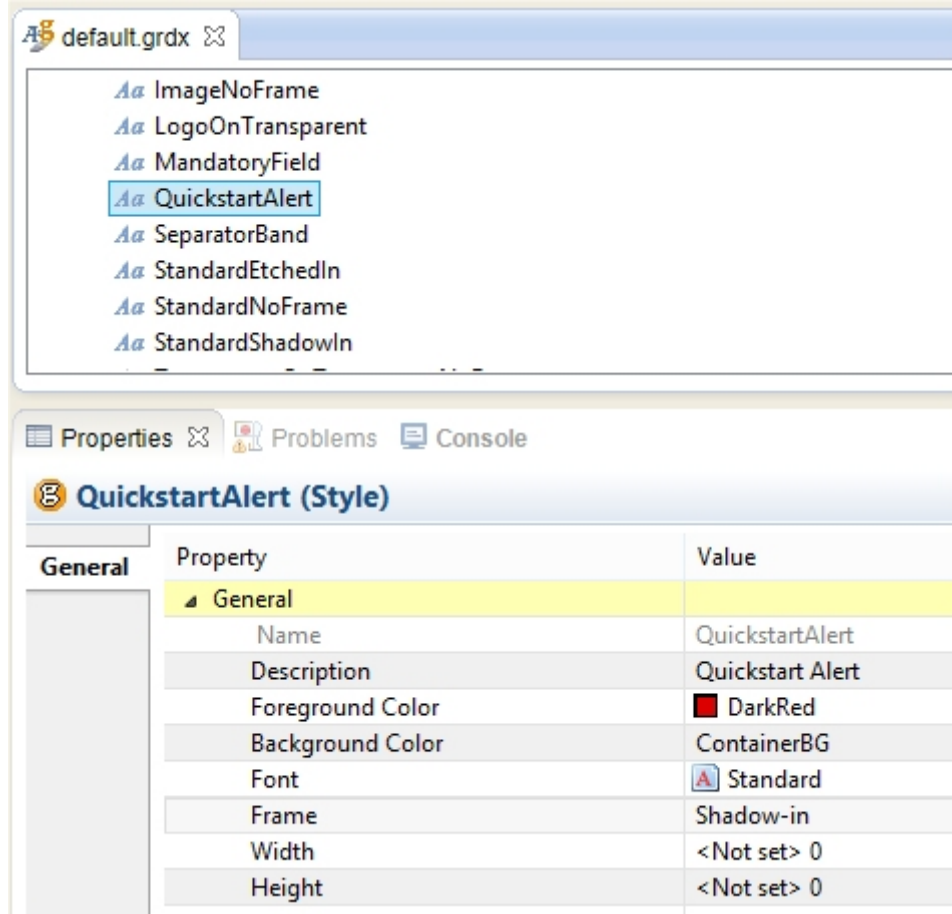




1. In the *Project Explorer*, expand the *qs-guimodel>Model View* node and double-click the **Resources** node. This opens up an editor which displays a tree view of the various resources.



- Expand the *Styles* node. Right-click on the *Styles* node and select *Add>Style*. In the *Add Style* dialog that appears, enter name **QuickstartAlert** and click OK. The QuickstartAlert style is added as shown:



- In the *Properties* view, enter the following values:

<b>Description</b>	Quickstart Alert
<b>Foreground Color</b>	DarkRed
<b>Background Color</b>	ContainerBG
<b>Font</b>	Standard
<b>Frame</b>	Shadow-in

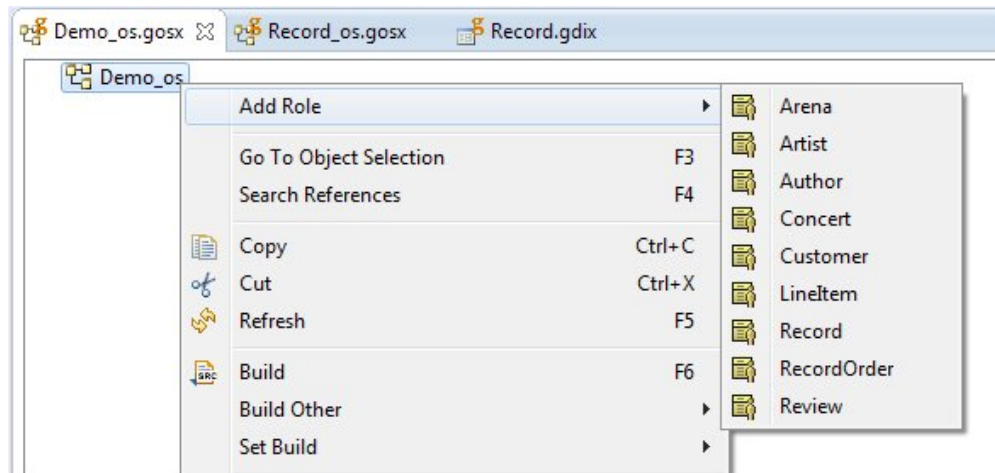
- Close the resource editor by pressing **Ctrl+W**. The system will display a dialog informing you that the file was modified and asking you whether you want to save the changes. Click on **Yes**.

### 6.3.2 Create an Object Selection

An object selection is a subset of a class model, specifying classes that comprise the information model used by a dialog or a service. Normally, you have one object selection for each dialog specified in the application model. In the *qs-guimodel* project, various object selections have already been created for you. However, to illustrate how to create an object selection, the record object selection is empty.

To create an object selection:

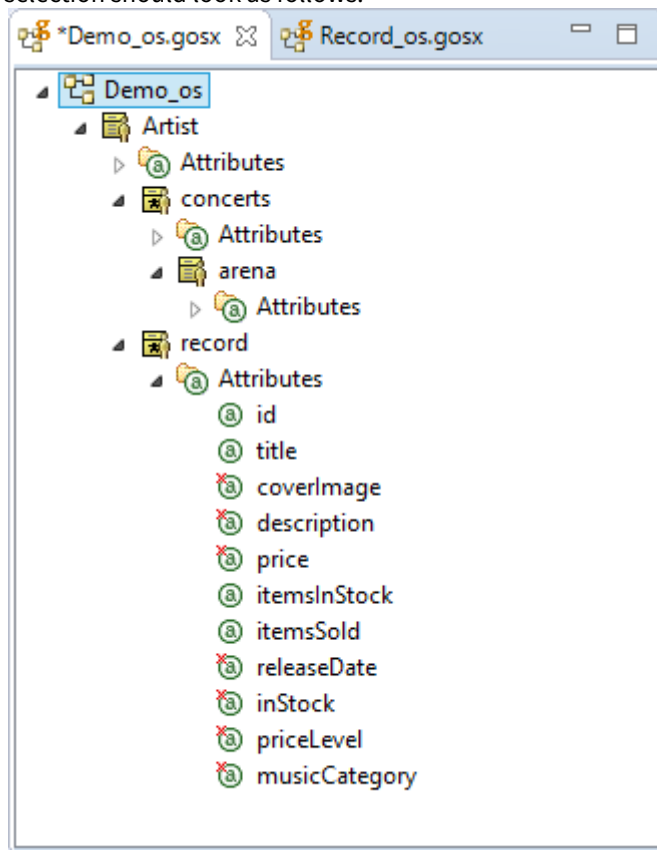
1. Expand the `qs-guimodel>Model View>Object Selections` node in the Project Explorer. Double-click the **Demo\_os** node. This opens an editor called *Demo\_os.gosx*.
2. Right click on the *Demo\_os* node in the editor and select the **Add Role>Artist** menu option.



An *Add Role* dialog appears. Accept the name **Artist** and press OK.

3. Right click on the newly added *Artist* node in the editor and select the **Add Role>Concert** menu option. Accept the name to **concerts** and press OK.
4. Right click on the *concerts* node in the editor and select the **Add Role>Arena** menu option. Accept the name **arena** and press OK.
5. Right click on the *Artist* node in the editor and select the **Add Role>Record** menu option. Accept the name record and press OK.
6. Expand the *record* node, right-click the *coverImage* attribute and select **Exclude attribute** to exclude it from the dialog. Note that a red x marker overlays the attribute icon. This indicates that the attribute is excluded. Also exclude the attributes: *description*, *price*, *releaseDate*, *inStock*, *priceLevel*, *musicCategory*. Your object

selection should look as follows:

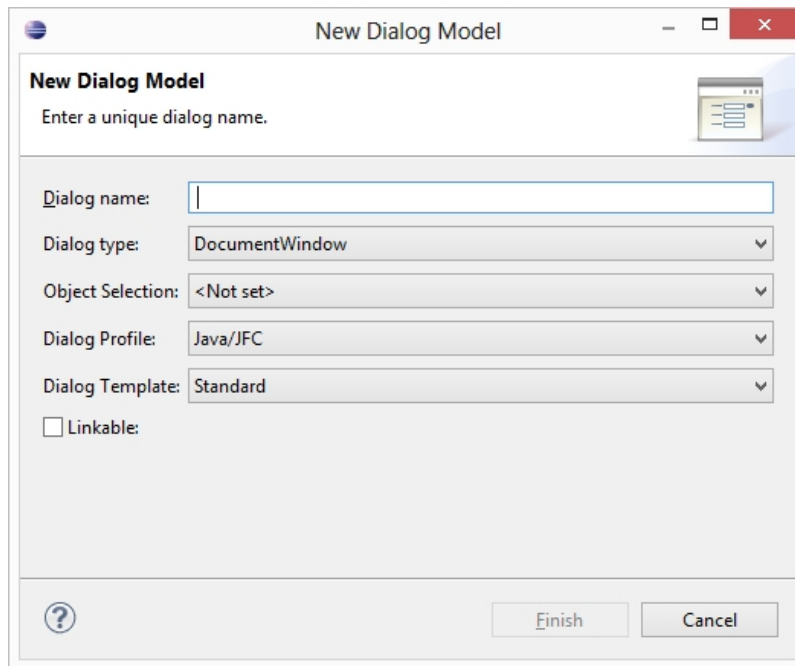


7. Look at the Properties of the roles and their attributes to see the various properties that can be changed to affect the dialog creation.
8. Note the asterisk next to the editor label (*\*Demo\_os.gosx*). This indicates that the file has been modified. Click on the Save button in the main toolbar to save the Object Selection file and the asterisk disappear.

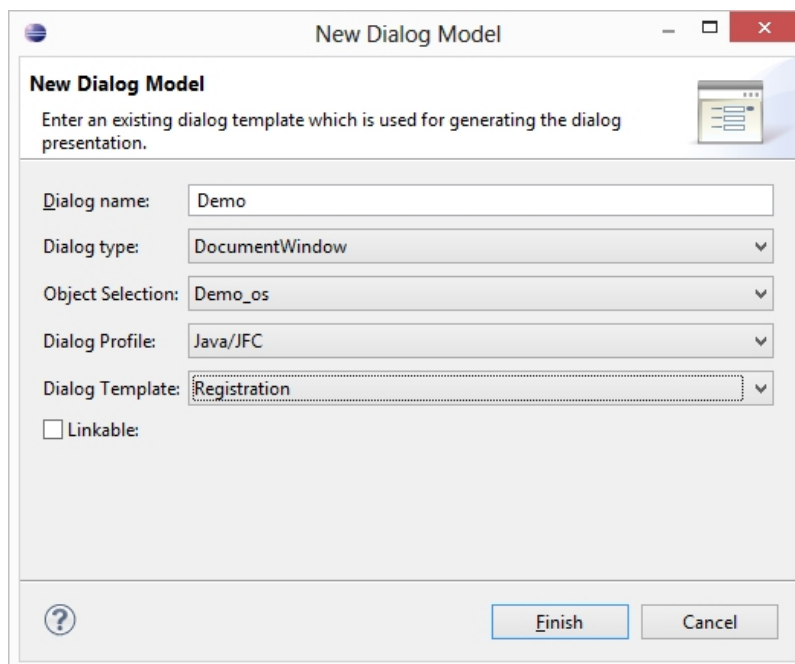
### 6.3.3 Generate the Dialog Model

The preferred way to generate a dialog is to do most of the application modeling in the application model editor. Open the *RecordshopAdm* application in the Application Model Editor:

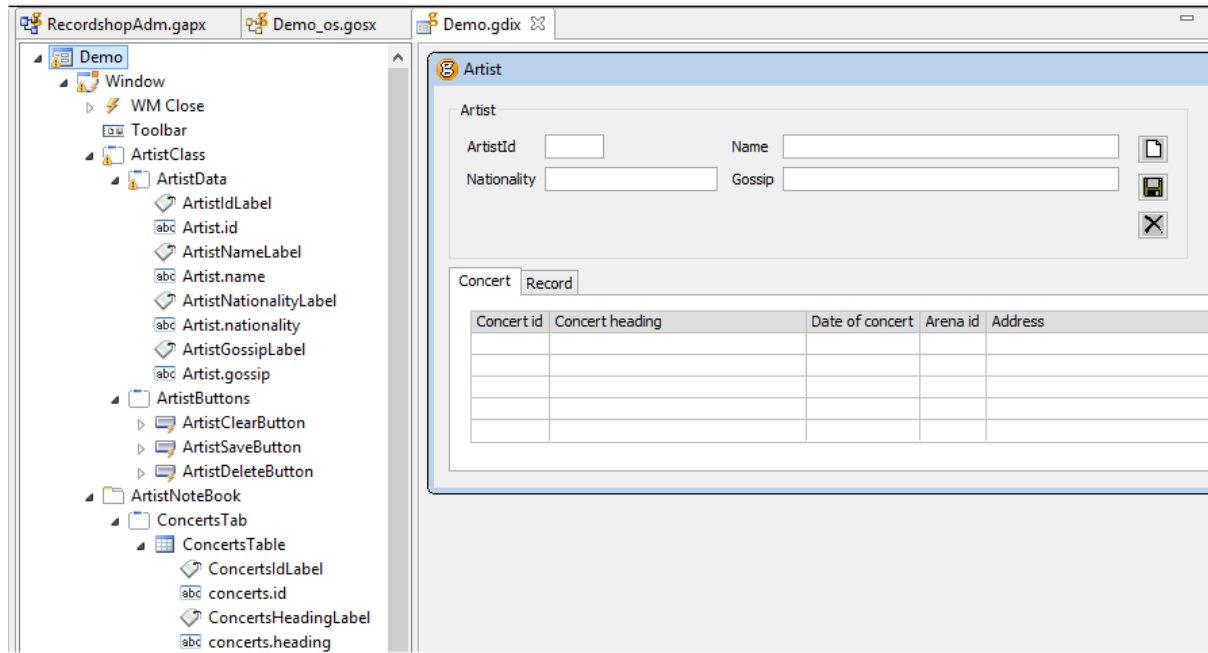
1. Expand the *qs-guimodel>Model View>Applications* nodes and double-click on the *RecordshopAdm* node. The Application Model Editor is displayed:
2. Select the *Dialog models* nodes, right-click and select *New Dialog model...*:



3. Add Dialog name *Demo*, select Object Selection *Demo\_os* and change the Dialog Template from *Standard* to *Registration*:



4. Accept the default values for the other fields and click the Finish button to generate the dialog. An editor is opened containing a tree view representation of the dialog on the left. To the right of the tree view, a canvas containing a visual representation of the dialog is displayed.



5. To expand the canvas area, move the cursor over the right corner of the editor until the cursor changes to a resize icon, then click and drag the cursor until the entire dialog is visible.

As you click on different nodes in the tree view, the corresponding control in the visual designer is highlighted with a red dashed line. Also, the *Properties* view displays all properties for the highlighted control.

### 6.3.4 Edit the Dialog Model

The dialog design generated in the previous section is the result of applying a *dialog template* to an object selection. The outcome of this automatic process is a dialog with one component for each attribute of the object selection. A *dialog template* is a resource that decides how attributes are represented in a dialog, as well as the layout. Existing templates may be changed, and new ones may be added.

However, to have more control over the look of the dialog, the WYSIWYG User Interface designer can be used to not only visualize the dialog, but also aid in its creation and modification.



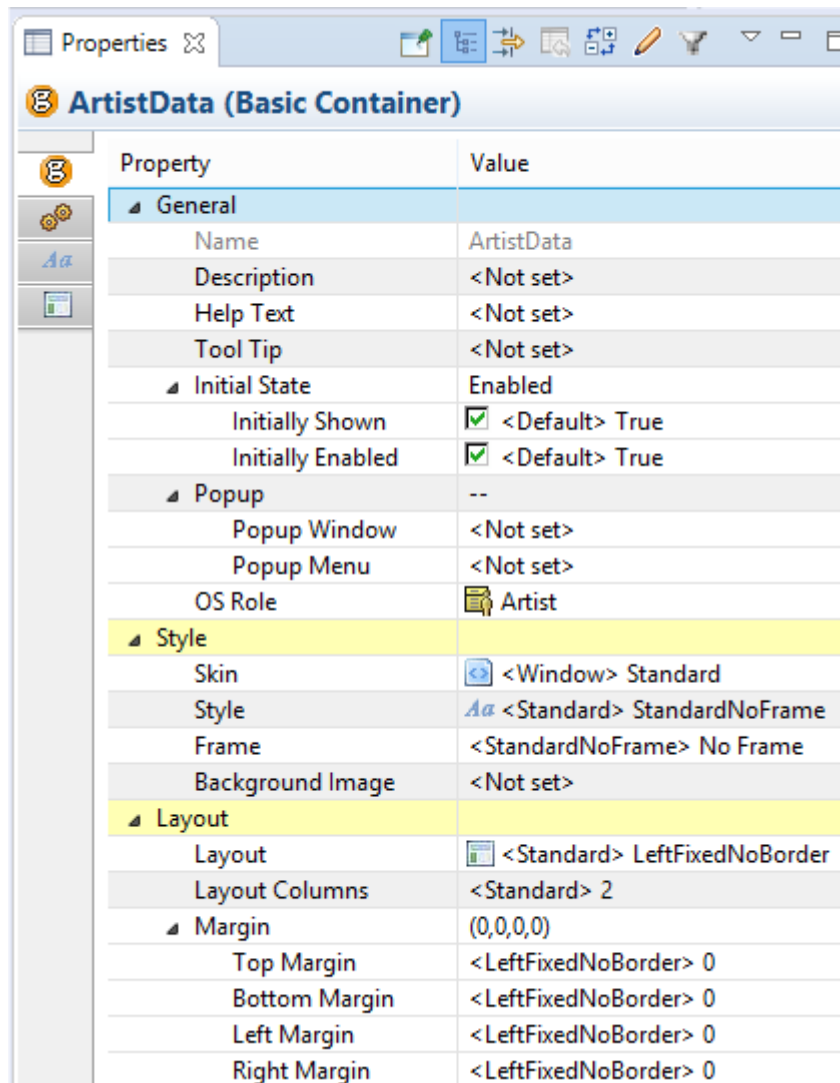
The visual designer uses native widgets (SWT) and may not look exactly like the final client target

### Working with the Properties view

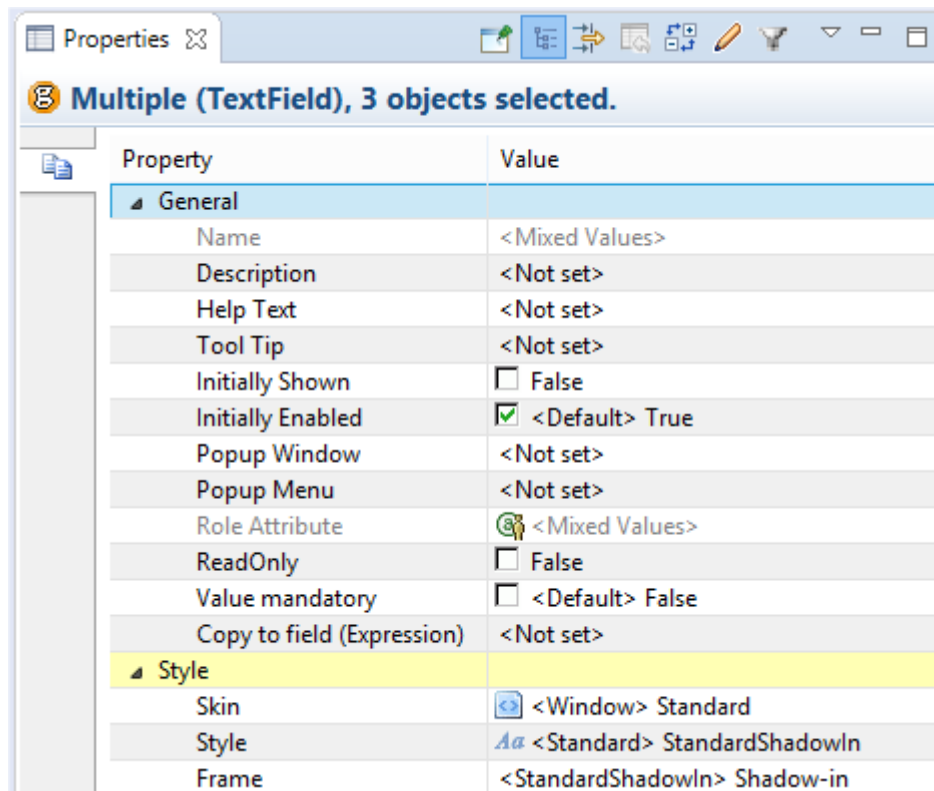
The *Properties* view is used to change property values that affect the look and function of the widgets in the dialog. The *Property* view also has tabbed categories that group similar properties together.

To edit the dialog, proceed as follows:

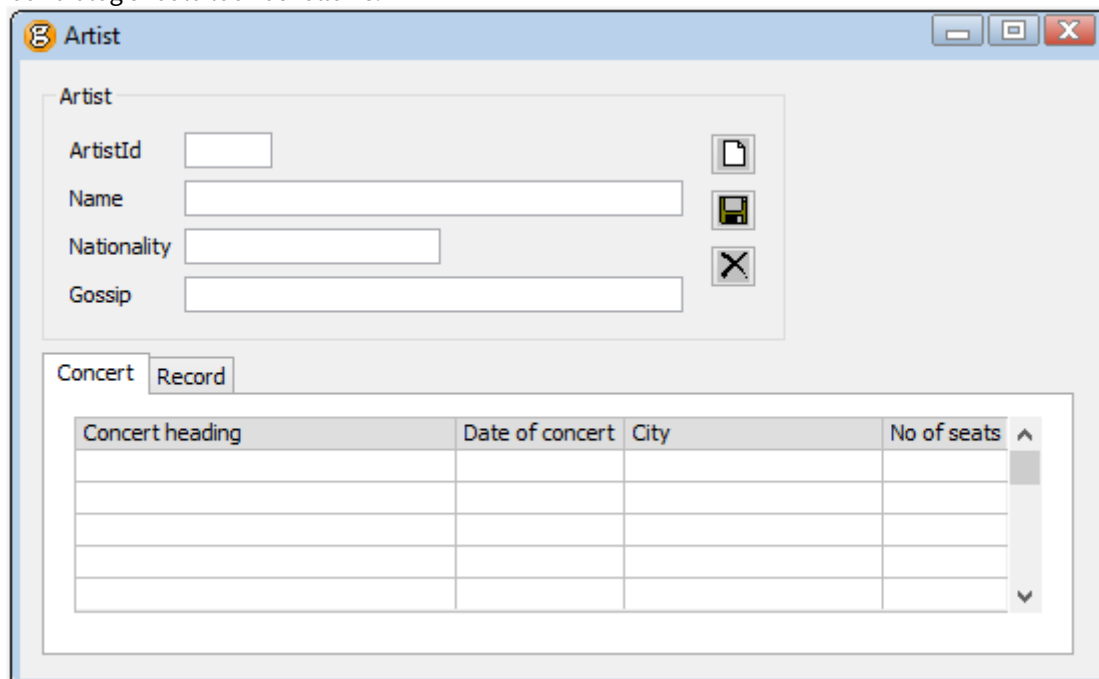
1. Click the *ArtistData* container node in the tree view dialog model. Notice the red box drawn in the visual design. The *Properties* view displays as follows



- Click on the *Layout Columns* value (<Standard> 2), set the value 1 and press Enter.
- Click on the *Gossip* text field in the visual designer. The *Artist.gossip* field in the tree view should be selected. Now Change the *Style.Style* property from *StandardShadowIn* to **QuickstartAlert**. This will not change the visualization as it will only display the gossip values in red.
- Identify the *ConcertsTable* container in the tree view and multi select *concerts.id*, *arena.id* and *arena.address*. In the Properties View, set *Initially Shown* false.



5. Select the *ConcertsTable* container, set the *Widget.Paging mechanism* to *Scrollbar*. Do the same for the *RecordTable* container element.
6. Your dialog should look as follows:

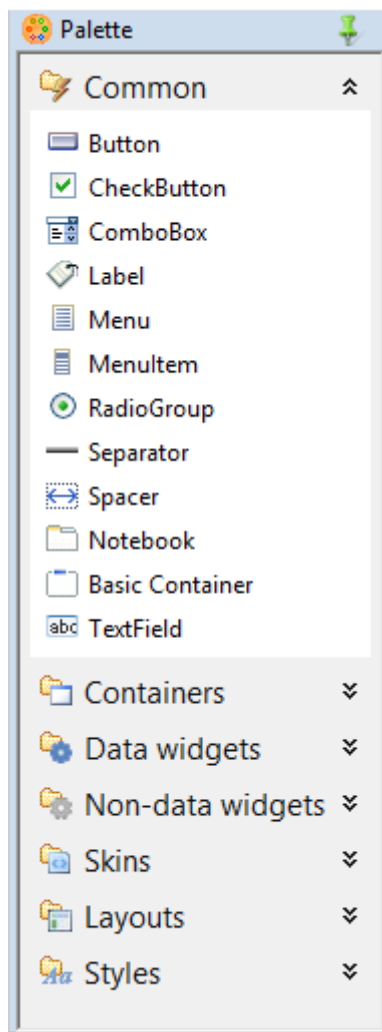


7. Save the dialog by clicking on the save icon in the main toolbar.
8. Close the editor containing the dialog by clicking on the X beside the *Demo.gdix* tab label or press **Ctrl + W**.



To demonstrate the palette, we will reorganize the window to optimize the screen real estate by adding a new container containing two columns and moving various fields into this new container.

1. Expand the *qs-guimodel>Model View>Dialog models* nodes and double-click on the *Demo* node.  
(Note: You can also double-click the *qs-guimodel>model>dialogmodels>Demo.gdix* node. This file contains the XML code that describes the dialog.)
2. Expand the *Common* accordion bar in the palette by clicking on the arrow to the right of the *Common* label. It should look as follows:



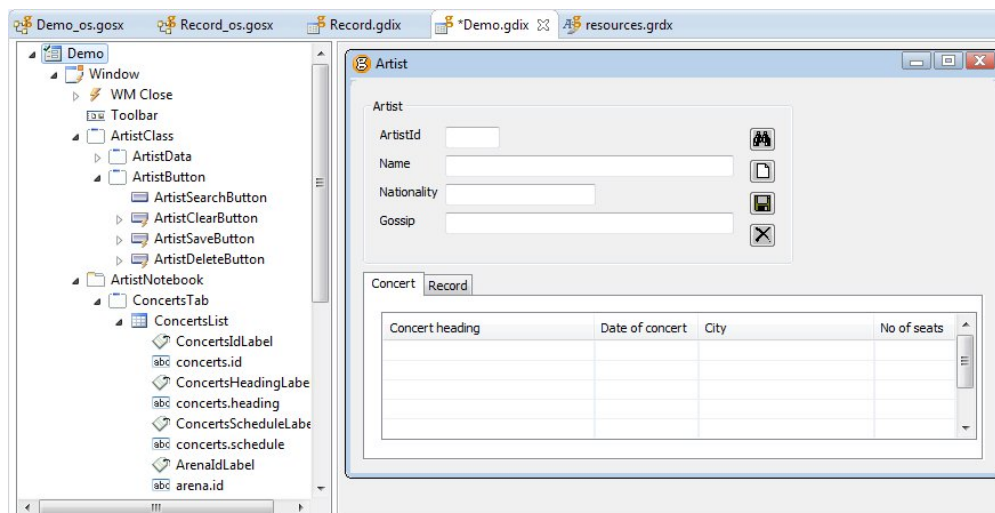
3. Click on *Button* item in the *Common* group in the Palette so that it is highlighted in blue.
4. Move the cursor over the visual designer until it is hovering above the Clear Button. (This should highlight the ArtistClearButton) Now click the left-mouse button and give the button a name (ArtistSearchButton). This adds a new *Button* above the *ArtistClearButton* node in the tree view.
5. Click on the *ArtistSearchButton* node in the tree view and make the following changes in the Properties View:

Widget.Title	Search
Widget.Content.Button contents	Use Image
Widget.Image.Image	FindAll

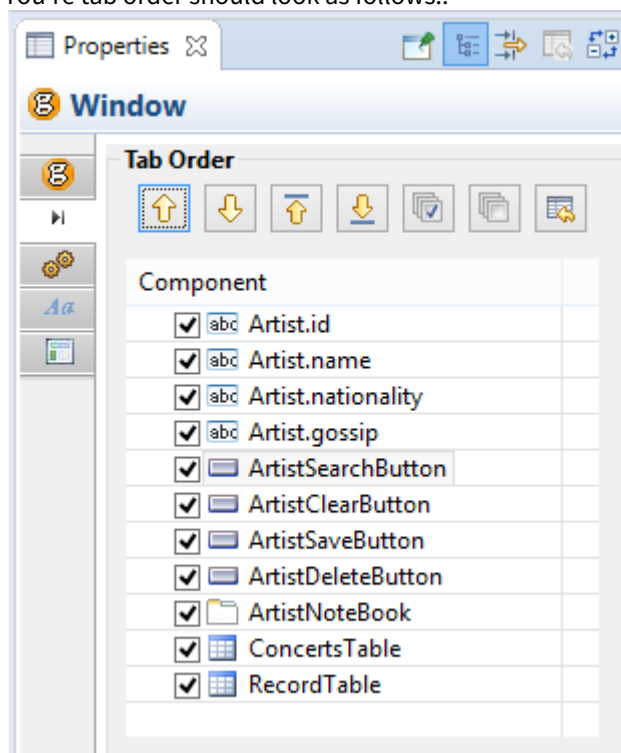
## Working with the Palette

The palette is contained in the Dialog Editor and contains multiple drawers (expand bars), each containing multiple dialog controls. It can be used to add widgets to the visual canvas or the tree view by either doing a drag-and-drop operation, or by simply clicking on the palette and then clicking on the canvas or tree view. It can also assign layouts to containers and styles to individual components.

6. Your dialog should look as follows:



7. The adding of the new button causes the tab order to change. To fix the tab order, Click on the *Window* node in the treeview and then click on the *Tab Order* tab in the *Properties* view.
8. Select the *ArtistSearchButton* node and press the *Up* button located above several times, until the button is located above the *ArtistClearButton*.
9. Your tab order should look as follows:.



10. Save the dialog by clicking on the save icon in the main toolbar.

### 6.3.5 Add an Event

The dialog includes three pre-defined buttons which provides functionality. A *Save* button for saving dialog data to the database, a *Clear* button for clearing the fields in the dialog and a *Delete* button for *deleting* the data. The initial presence and function of these buttons is dictated by the Dialog template we used to create the dialog. As part of default generation is a *Find* function, which is defined on the edit field representing a primary key for the root role used. In his case it is the *Artist.id* field with a *Value Changed* event. The event fires the *findArtist* method. The effect of this behavior is that when a user types a value into the *Artist.id* field, the system searches the database for the specified artist and the dialog gets populated with information.

We now wish to add new functionality to the dialog: When a user clicks the *ArtistSearchButton*, the system searches the database for the specified artist given by the *Artist.id* field and the dialog gets populated with Artist information.

#### Defining the Event and Method

A list of pre-determined user-interface or system events is available depending on the type of selected object. You will now assign a *Clicked* event to the *ArtistSearchButton*.

1. Select the *ArtistSearchButton* in the dialog designer
2. Right click and select **Add Event > Clicked**
3. Right click on the Clicked node that is added below the *ArtistSearchButton* node and select **Add Method > findArtist**
4. A method is an abstract level of functionality that can perform one or more actions. An action is a pre-defined functionality that can perform various tasks such as navigation, data handling and more. A method can contain one or more actions and the code generator creates the named methods. The **findArtist** method was generated as part of the initial dialog generation.
5. The Methods tab, located at the bottom of the editor shows the defined methods. Ensure the *findArtist* method contains the Find action with parameter Artist.

The screenshot shows the g9 IDE interface. The top pane displays the \*Demo.gdix file with a tree view of methods. The bottom pane shows the Properties window for the 'Find > Artist (Action)'.

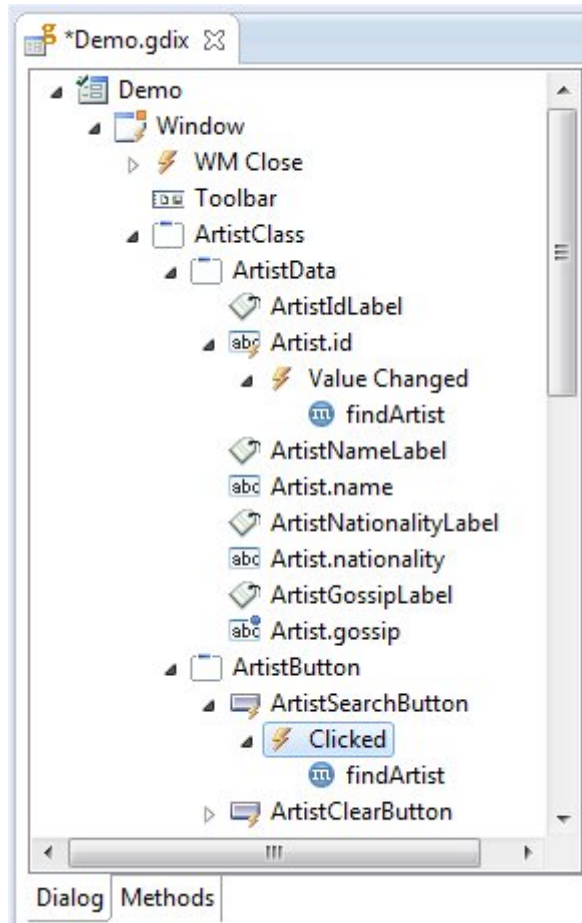
**Methods List:**

- Methods
  - clearArtist
    - Clear > Artist
  - closeDemo
    - Close > Demo
  - deleteArtist
    - Delete > Artist
  - findArtist
    - Find > Artist
  - saveArtist
    - Save > Artist

**Properties Window: Find > Artist (Action)**

Property	Value
<b>General</b>	
Name	Find
Description	<Not set>
Target	Artist
ServerEvent	<input type="checkbox"/> <Default> False

6. The tree view should look as follows for the two events defined:



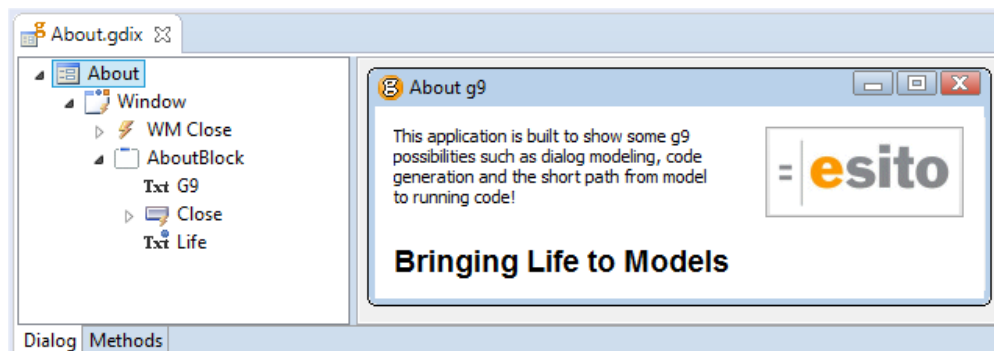
7. Save the dialog by typing **Ctrl+S**

### Editing the applications Demo menu item

The sample application contains a MDI window (RecordshopApp) with the application menus. To add and activate the Demo dialog, open the dialog *RecordshopApp*, go to the *Methods*, select *openDemo* method and select the Open action. In the Properties **General** section, select the Target combobox and set **Demo** as parameter.

### 6.3.6 Edit a Dialog Box Model

1. Open the About dialog by double clicking the *qs-guimodel/Model View/Dialog models/About* node.

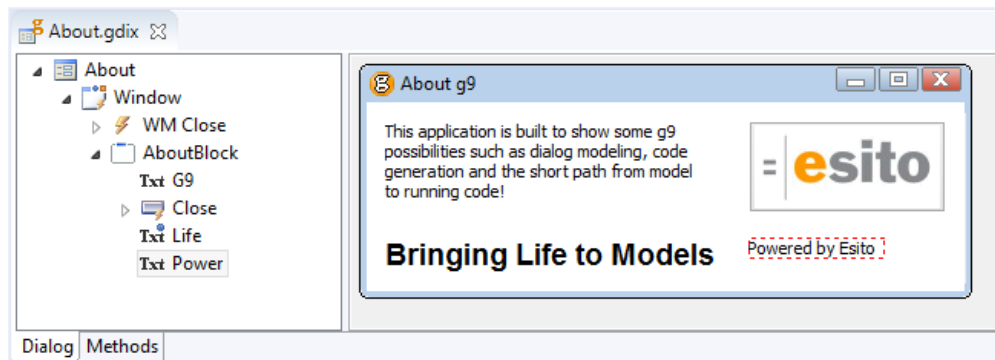


2. Expand the *Non-data widgets* expand bar in the palette and click on the **Text** item so that it is highlighted.

3. Move the cursor so that it hovers over the right side of the **Bringing Life to Models** text field, then click and enter the name Power. This adds the text field to the right of that field, just below the *esito* logo.
4. Click on the newly added field and change the properties as follows:

<i>Widget.Text</i>	<b>Powered by Esito</b>
<i>Style.Style</i>	<b>WebSmallText</b>

5. Your dialog should look as follows:



6. Save the dialog by clicking on the save icon in the main toolbar.

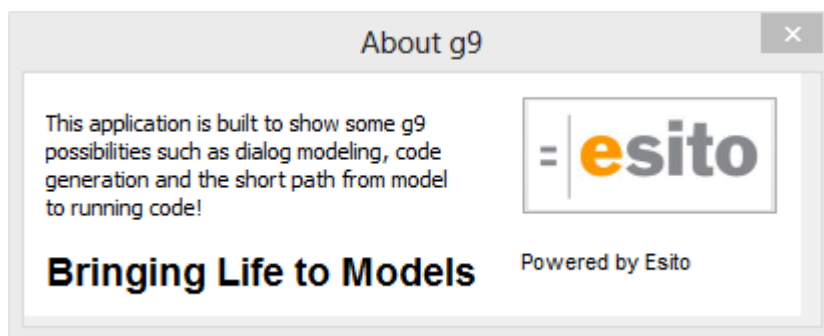
### 6.3.7 Compile and rerun

You are now ready to compile and rerun your edited Swing application.

1. The default Build Configuration contains all necessary Generator Configurations. Select the g9 project root *qs-guimodel* and push F6 (accelerator for the Build menu). This will generate all code in this project.
2. Select the *Run > Run configurations...* menu option, expand the *Java Application* node and click the *Swing* node.
3. Select the *Help>Demo* menu option
4. Enter the value *12* into the *ArtistId* field and then tab out of the field. Due to the event that you added, the Artist record with all its connected Concerts and Records are retrieved from the database.

Concert heading	Date of concert	City	No of seats
Heavy Dimmur	23. 10. 2012	London	2300
Heavy Dimmur	24. 10. 2012	Notodden	1100
Heavy Dimmur	25. 10. 2012	Oslo	10000
Heavy Dimmur	26. 10. 2012	New York	100
Heavy Dimmur	27. 10. 2012	Oslo	50

5. Press the Clear button to clear all the fields and enter 47 in the *ArtistId* field and then tab out.
6. Select the *Help>About* menu option.



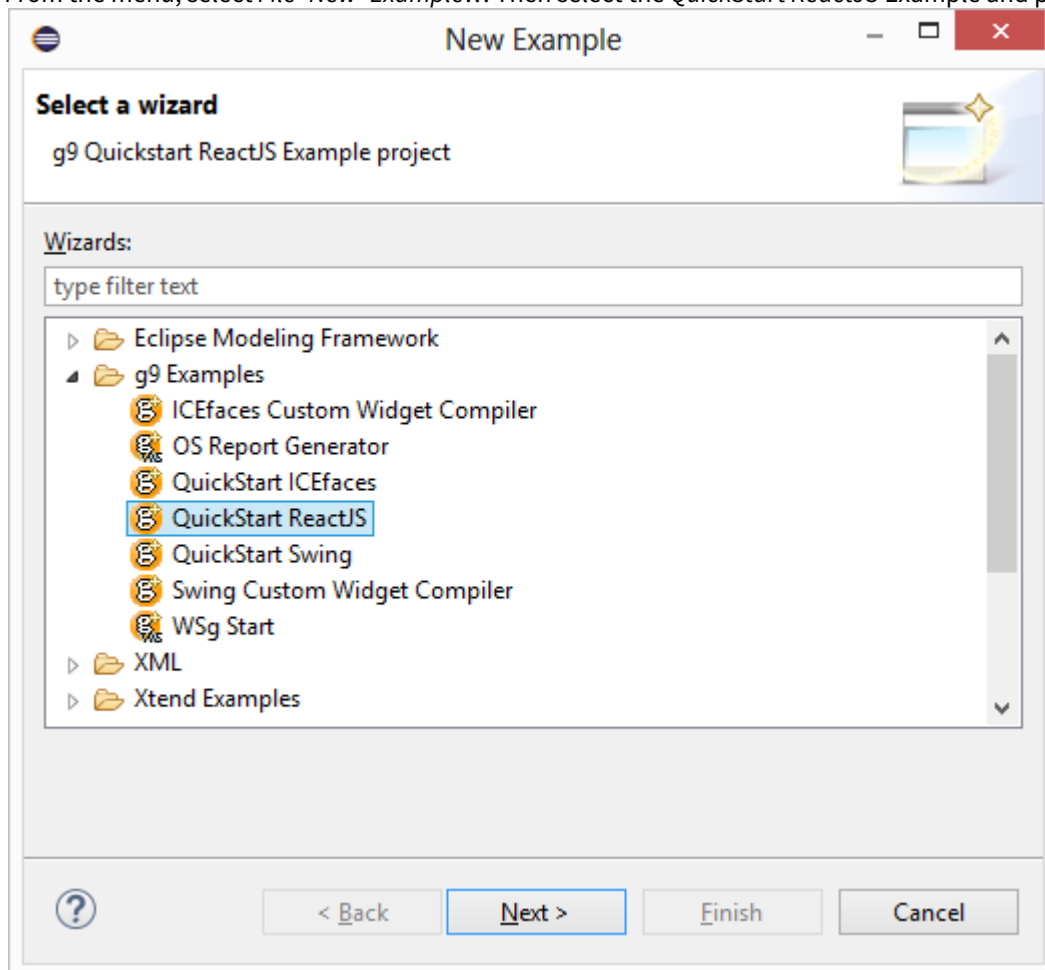
## 7 Creating the Quickstart ReactJS Project

A g9 model project is where the g9 objects are stored including the domain class model, objects selections and dialog models. To create the Quickstart ReactJS project, you can import an existing project into the workspace.

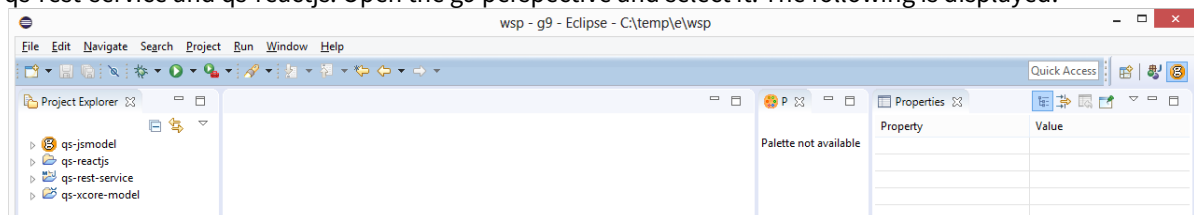
In order to quickly get you started, the *Quickstart ReactJS* project contains an application based on an existing domain model designed using *Xcore*. This project contains predefined database model, object selections and dialog models, as well as all the necessary generated Java/JavaScript code for running parts of the application in a JavaScript React web client.

To create the Quickstart ReactJS projects:

1. From the menu, select *File>New>Example...* Then select the *QuickStart ReactJS* Example and press *Next*.



2. Press the **Finish** button. Several projects are opened and shown in the explorer: *qs-xcore-model*, *qs-jsmodel*, *qs-rest-service* and *qs-reactjs*. Open the g9 perspective and select it. The following is displayed:



The *Quickstart* project *qs-jsmodel* contains the model which includes the Domain Classes, the Object selections and Dialog models. The other projects are the definition of the Xcore domain model (*qs-xcore-model*), a Java project that has been created for the generated service code (*qs-rest-service*) and a JavaScript React project for the generated client code (*qs-reactjs*).



3. The system should begin building the project automatically and display the progress in the status bar. You might notice error markers (red x) next to the projects. These should disappear once the system finishes building the projects and Maven resolves the dependency issues.
4. If there are still error markers, it is possible the build did not occur automatically or correct. Close and Open all projects. If there are still error markers, you can select the menu option *Project > Build* or *Project > Clean*. If there are still errors, it is possible Maven did not resolve the dependencies. You can select all the projects then right-click on the project and select *Maven > Update Dependencies*.

## 7.1 Javascript ReactJS Code Generation

In this section you will generate client code for a web client that is based on React, which is an open source JavaScript framework. The sample *Quickstart* application uses Spring Boot (Tomcat web server) for the server part, Express for client part and comes with the generated code for the various web objects. This sample will still take you through the steps for generating this code.

### 7.1.1 Prerequisites for ReactJS

Necessary pre-installation steps is described in the [Installation](#) part.

The application code is generated to two target projects. The Java code is generated to *qs-rest-service*, and the JavaScript code is generated to *qs-reactjs*.

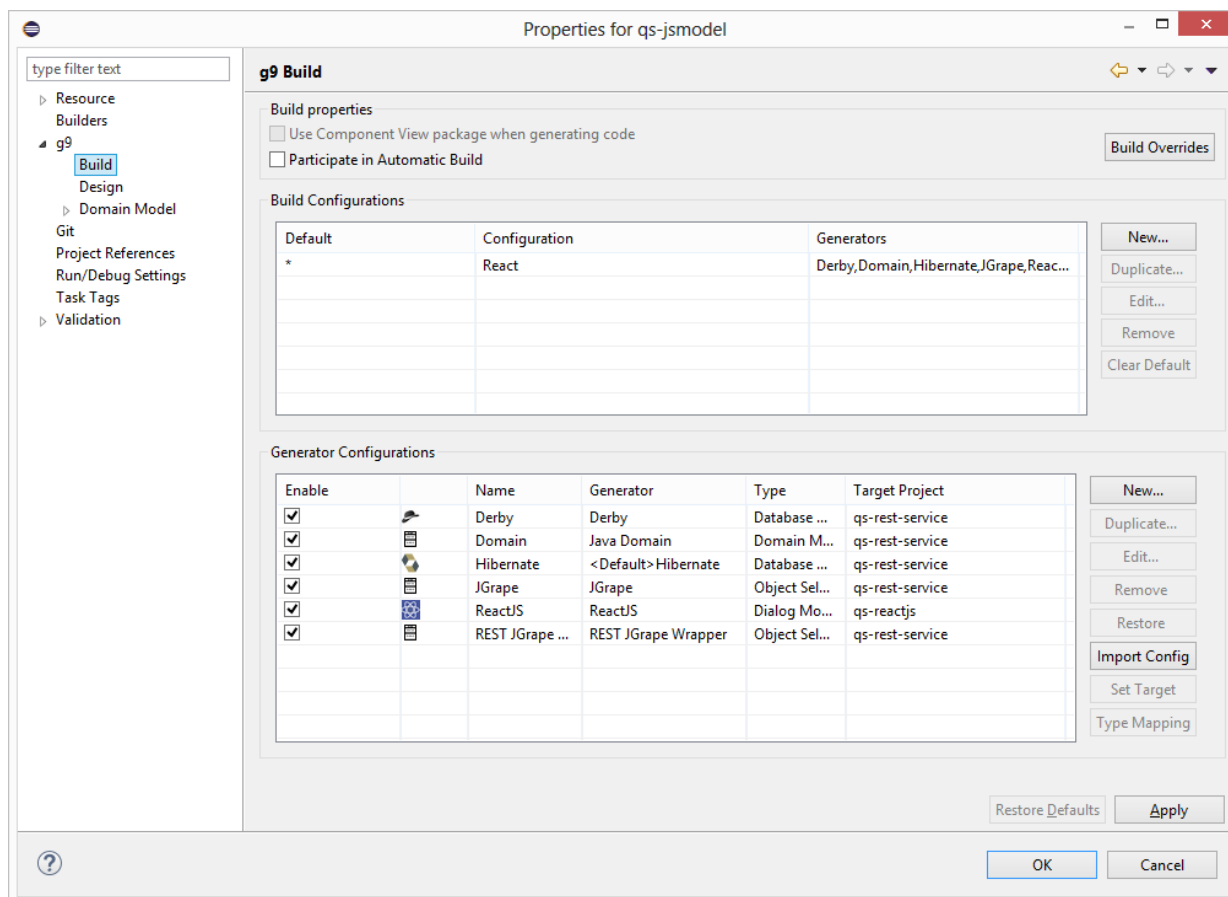
The *qs-rest-service* project is based on *Spring Boot*, and all dependencies are resolved by Maven.

The *qs-reactjs* project uses *Node.js* and *npm* to resolve dependencies and run the project. Download (from <https://nodejs.org/>) and install *Node.js* if not already done. The project is tested with the LTS version: node.js (12.16.3) and npm (6.14.4).

### 7.1.2 React Build Properties and Code Generation

g9 is structured to allow the generation of domain classes, client code, service code, database schema and hibernate mappings to occur alongside the Eclipse project build. You can configure the system to automatically generate code when saving content, or you can generate code manually. The Build section in the g9 project properties is where you can configure the code generation. Select the g9 perspective when working with g9 model projects.

- Click the *qs-jsmodel model* project in the *Project Explorer* and then right-click and select Properties. When the Property Dialog appears, expand the *g9* node and then click on the *Build* node. The following dialog is displayed:



## Generator Configurations

You can create multiple *Generator Configurations* for the different components of the application. A *Generator Configuration* is a user-defined term that consists of a supported code generator, a set of parameter settings for that specific generator, and a specified Target Project for separating code among multiple projects.

In addition to the *Generator Configurations*, you can create multiple *Build Configurations* which consists of a series of Generator Configurations. These can be associated to g9 objects or be performed on Build/Build Other commands. A default Build is marked with an asterisk and will be performed on a Build command.

If you wanted, you could select the option '*Participate in Automatic Build*' and the code generation would be performed on save of g9 models.

### Generate all in one Build

The default Build Configuration contains all necessary Generator Configurations. Select the g9 project root and push F6 (accelerator for the Build menu). This will build all code in this project.

### Separation of generated and manually maintained code

The generators supports splitting of generated and manually maintained code into two separate source hierarchies. This behavior is triggered by setting the "Source directory" generator parameter to a different value than the "Target directory" parameter. With this behavior, if a file is present in the "Source directory" it will not be generated to the "Target directory". Default values are src-gen/main and src/main.

### Console View

The Console View might show terminated views. To toggle between different Console Views, select *Display Selected Console* in the View toolbar.

### Domain Class Generator

The *Domain* generator generates Java code for the domain model which includes java classes, interfaces and enumerations. The *Quickstart ReactJS* project is configured to generate the domain classes into the *qs-rest-service* project.

1. Double click the **Domain** Generator Configuration in the bottom list to see the various standard and customized parameters and their values. Note the various parameters. Close the dialog.
2. Right click on the *qs-jsmodel/Model View/Domain Model* node in the *Project Explorer* and select the menu option **Build**.
3. Click on the Console View tab. It should display a list of all the generated Java classes.

### Dialog Code Generator

The *ReactJS Dialog* generator generates code for the client part of an application. The code generated is based on the dialogs modeled and their object selections. There are one *Dialog Generator Configuration* for the *qs-jsmodel* project. The *ReactJS* configuration uses the ReactJS generator for generating a React based client and is configured to generate the code into the *qs-reactjs* JavaScript project.

1. Double-click the **ReactJS** configuration to display the parameters. Close the dialog and then close the *Properties for qs-jsmodel*.
2. In the *Project Explorer*, expand the *Applications* node and right click on the **Recordshop** node and select **Build**. This starts the code generation process for the application main program and some helper classes.
3. Right click the Recordshop node again and select **Build Application > ReactJS**. This starts the code generation process for the complete client part of the application including application main and all dialogs.
4. Click on the Console View tab. It should display information regarding the code generation. If there is an error, it will be displayed here.

### Service Generator

The *JGrape* generator generates code for the service part of the application. This includes the code for all necessary CRUD actions. The *qs-jsmodel* project is configured to write all service code for the ReactJS clients into the *qs-rest-service* java project.

1. Open the *qs-jsmodel* properties dialog again and double-click the **JGrape** configuration to display the parameters. Note the location of the *Service Package*. Close the dialog and also the properties dialog.
2. Right click the Object Selections node and select **Build**. This starts the code generation process.
3. Click on the Console View tab. It should display the various code elements that were generated.

### REST Service Wrapper Generator

The *REST JGrape Wrapper* generator generates a REST layer on top of the JGrape service code for an Object Selection. The *qs-jsmodel* project is configured to write all REST wrapper code for the ReactJS clients into the *qs-rest-service* java project.

1. Open the *qs-jsmodel* properties dialog again and double-click the **REST JGrape Wrapper** configuration to display the parameters. Close the dialog and also the properties dialog.
2. Right click the Object Selections node and select **Build**. This starts the code generation process.
3. Click on the Console View tab. It should display the various code elements that were generated.

### Database Schema Generator

The Database generator is used to generate database schema scripts to help in creating databases. It works from the database model that has already been provided for the Derby database.

- Open the `qs-jsmodel` properties dialog again and double-click the **Derby** configuration to display the parameters. Note the target project is set to `qs-rest-service` and the target directory is `db`. Close the dialog

### Hibernate Mapping Generator

The *Hibernate* generator is used to generate data access code. The system will generate a mapping file named `<Classname>.hbm.xml` for each domain class specified in the database mapping file. Also created is a `G9Dataaccess.xml` file which contains references to all the domain class mapping files as well as the `hibernate.properties` file which contains JDBC and EJB connection properties.

1. Double-click the **Hibernate** configuration to display the parameters. Note the target project is set to `qs-rest-service` and the target directory is `src-gen/main`. Close the dialog.
2. In the *Project Explorer*, expand the *Model View/Database Models* node and right-click on the `recordshop` node and select the menu **Build**. This starts the code generation process of the database schema files and the hibernate mapping files.
3. Click on the Console View tab to display the various code elements that were generated.

### Model specific Builds

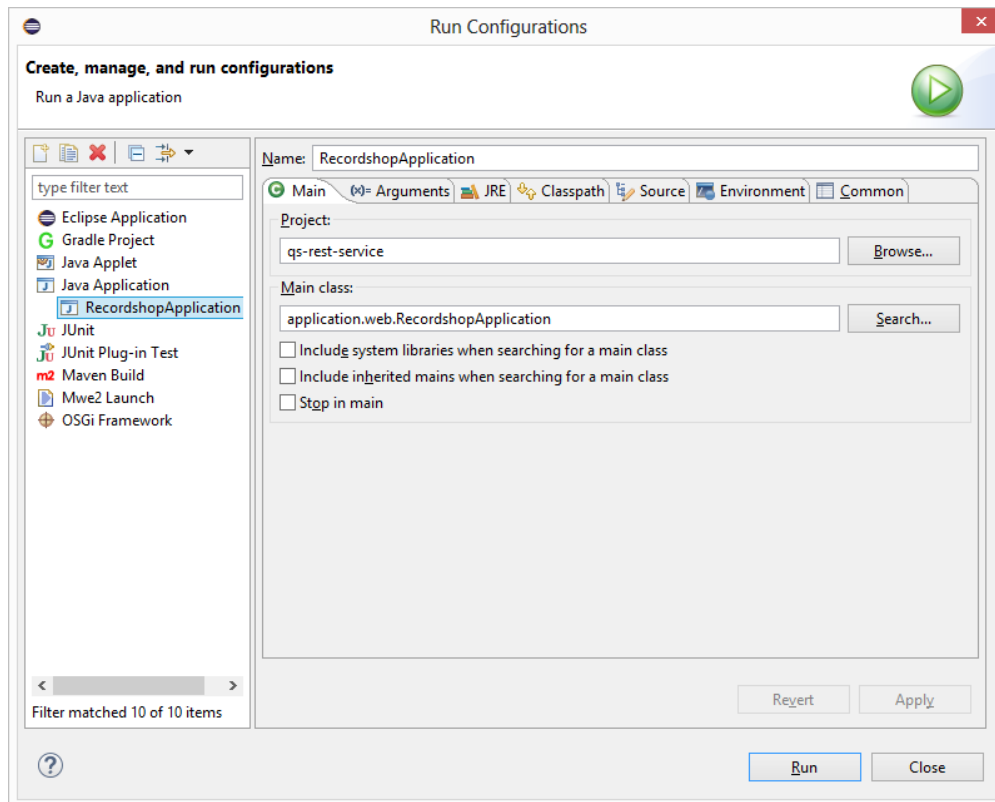
If you want to see what the specific generators do, you may follow the instructions below. However, all necessary code are generated and ready to run.

## 7.1.3 Compiling and running the React application

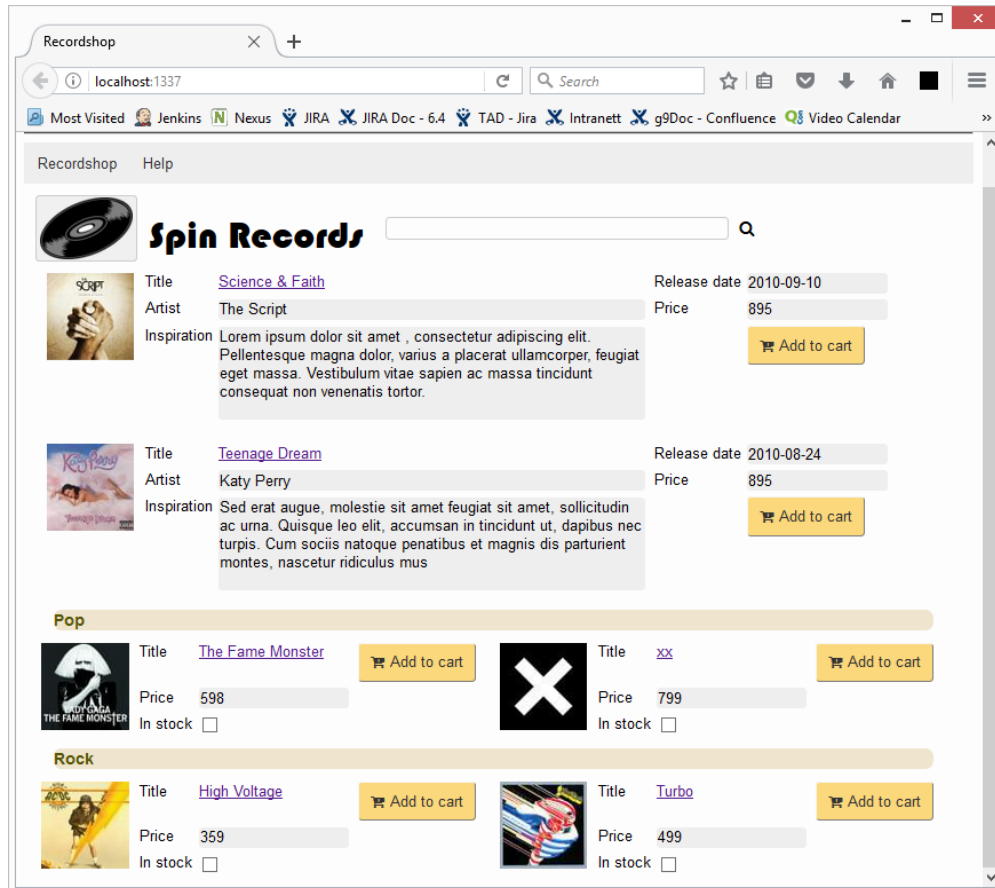
The sample project creates the database populated with data for you and it is stored in the `db` directory named `recordshop`. You can create/recreate it, see instructions on [Creating Derby Database](#).

You are now ready to compile and run your React application.

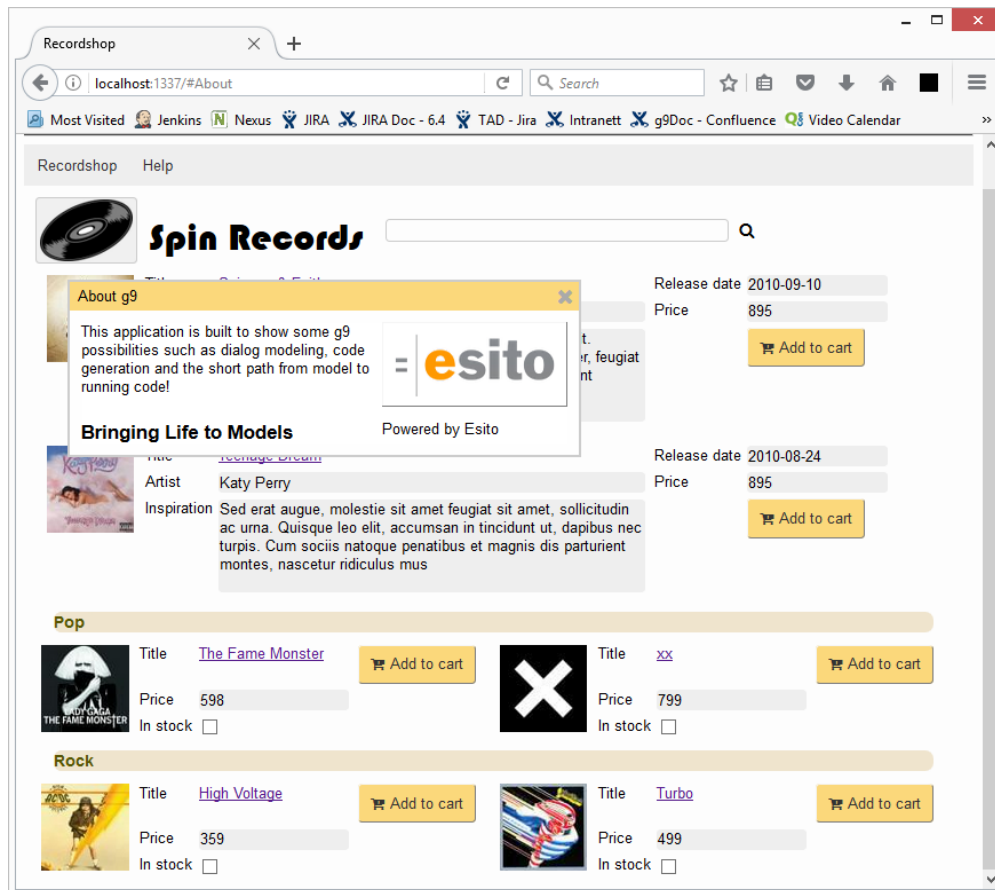
1. Click on the `qs-rest-service` Java project and press the F5 button to refresh
2. Check the menu option *Project>Build Automatically*. If it is checked, then the code should have automatically compiled. If it is not checked, select the menu option *Project > Build All*.
3. Select the *Run > Run configurations...* menu option. This opens up the Run Configurations Dialog
4. Expand the *Java Application* node and click the *RecordshopApplication* node. This has already been created for you.



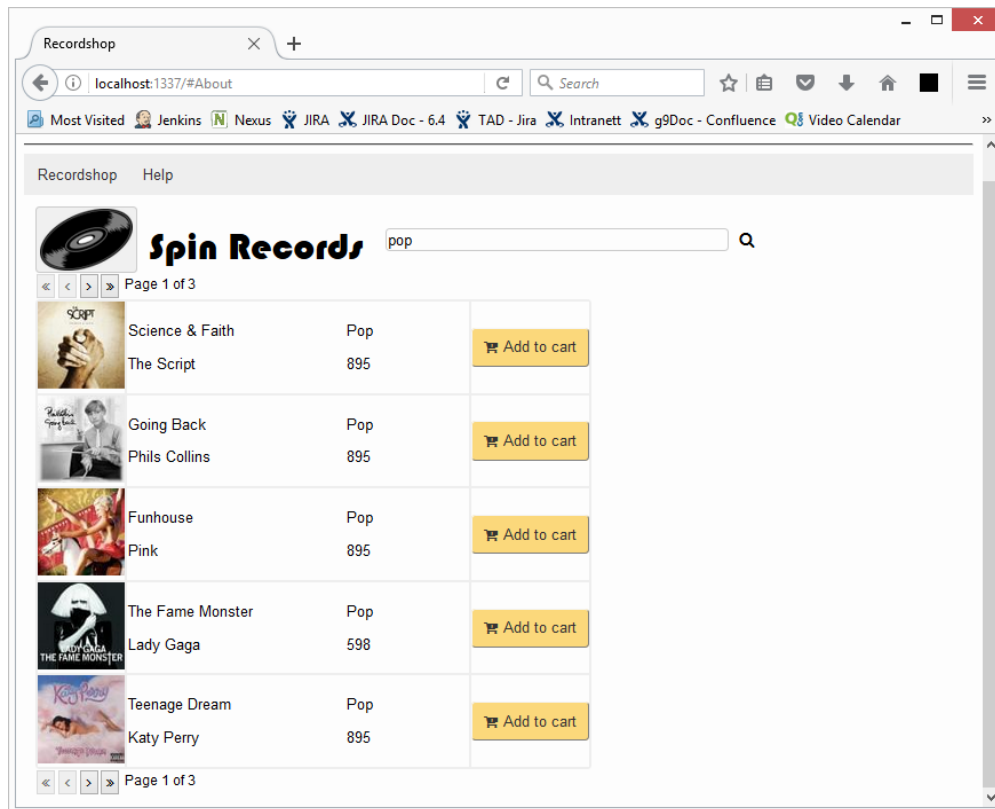
5. Click the Run button to start the server application.
6. Check the Console. The output should indicate whether the web server has started.
7. Open a Node.js command shell and navigate to the root of the qs-reactjs project.
8. Run `npm install` once (may take some minutes the first time) and then `npm start`. A browser window will appear with the first page of the application:



9. Select the menu Help > About from the menu bar at the top. The modified dialog is displayed:



10. Close the About dialog by clicking on the x in the top right or the logo button.
11. Enter the category **pop** into the search field at the top and then click the search button to the right. The following dialog is displayed:

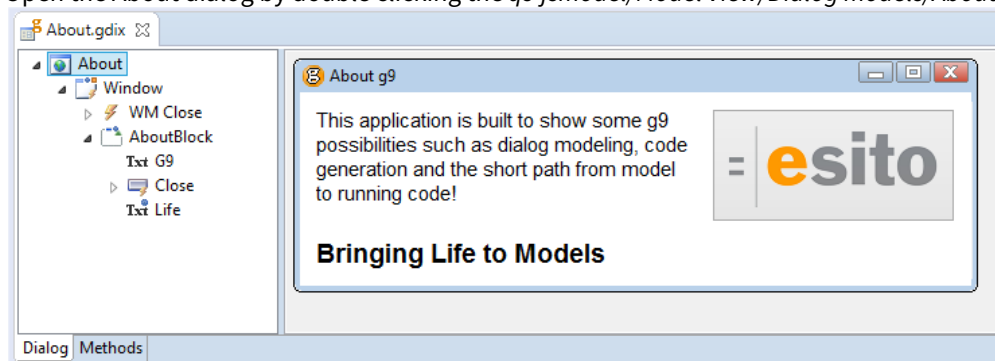


12. Close the Server by clicking the red 'Terminate' button in the Console view.

### 7.1.4 Edit the Dialog Box Model

In this section, you will modify the user-interface dialog *About g9*, regenerate the code and rerun the application. First, read more about g9 model artifacts in [Modeling your Dialogs](#).

1. Open the About dialog by double clicking the *qs-jsmodel/Model View/Dialog models/About* node.



2. Expand the *Non-data widgets* expand bar in the palette and click on the **Text** item so that it is highlighted.
3. Move the cursor so that it hovers over the right side of the **Bringing Life to Models** text field, then click and enter the name *Power*. This adds the text field to the right of that field, just below the *esito* logo.
4. Click on the newly added field and change the properties as follows:

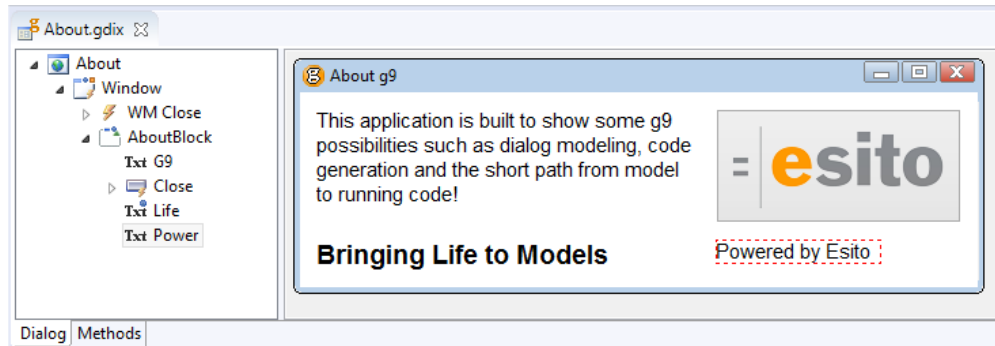
<b>Widget.Text</b>	Powered by Esito
--------------------	------------------



Style.Style

WebSmallText

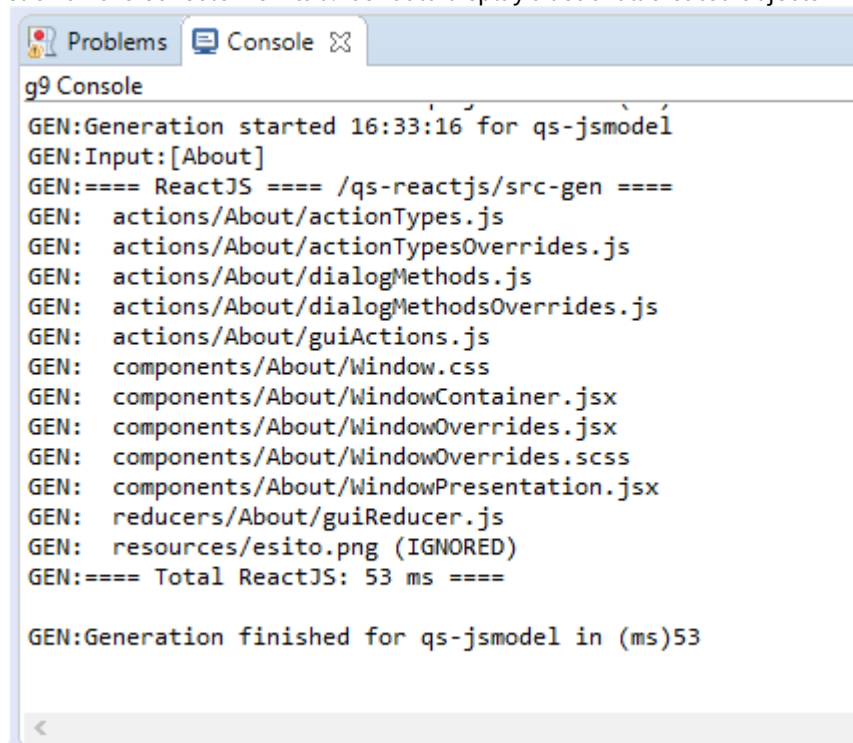
5. Your dialog should look as follows:



6. Save the dialog by clicking on the save icon in the main toolbar.

Generate the client code for the modified Help About dialog:

1. Click the *qs-jsmodel* project in the *Project Explorer* and then right-click and select *Properties*. When the *Property Dialog* appears, expand the *g9* node and then click on the *Build* node.
2. Double-click the **ReactJS** configuration to display the parameters. Close the dialog and then close the *Properties for qs-jsmodel* dialog.
3. In the *Project Explorer*, expand the *qs-jsmodel/Dialog Models* node and right click on the *About* node and select **Build**.
4. Click on the *Console View* tab. It should display a list of all created objects



5. If your React client is running, you will see the changes without restarting the application. Otherwise run the application to see the result.

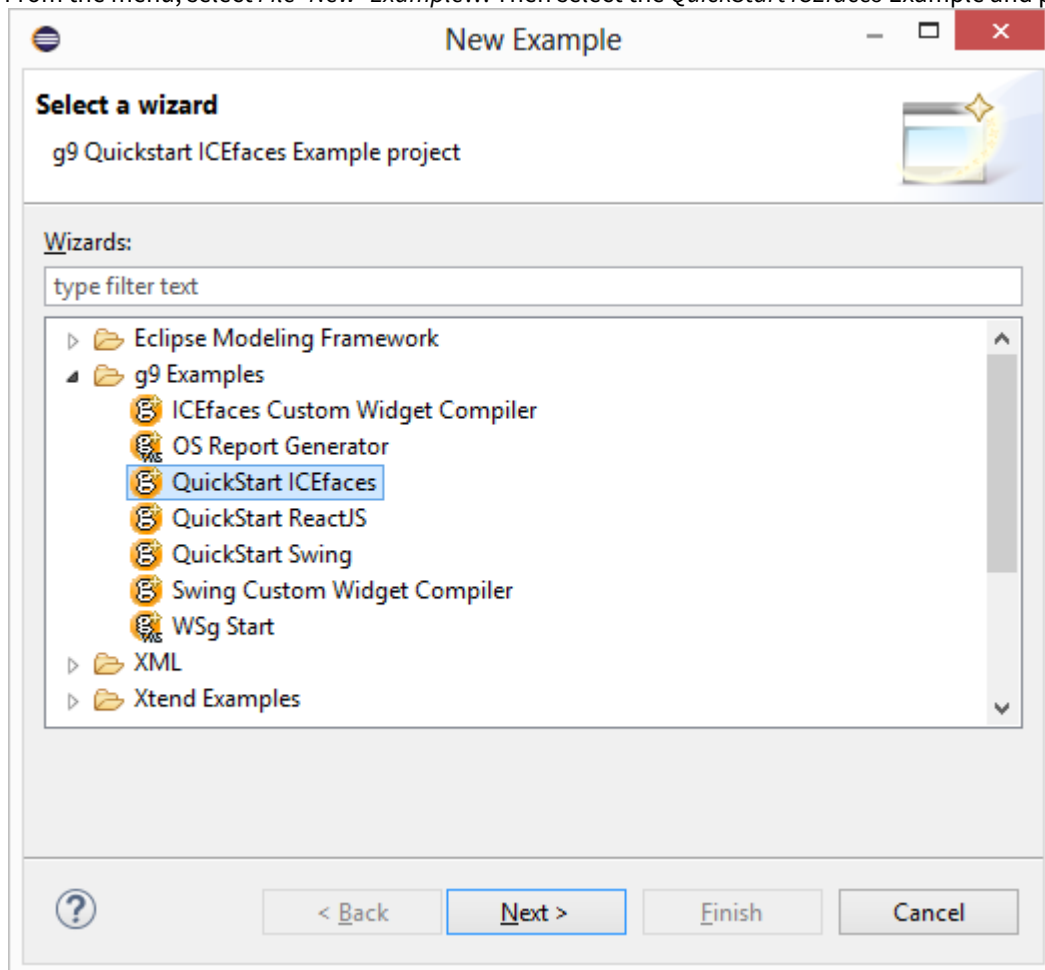
## 8 Creating the Quickstart ICEfaces Project

A g9 model project is where the g9 objects are stored including the domain class model, objects selections and dialog models. To create the Quickstart ICE project, you can import an existing project into the workspace.

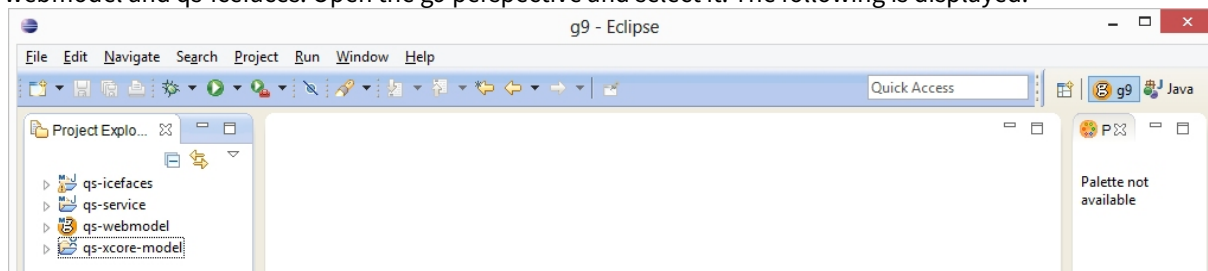
In order to quickly get you started, the *Quickstart ICEfaces* project contains an application based on an existing domain model designed using *Xcore*. This project contains predefined database model, object selections and dialog models, as well as all the necessary generated Java code for running parts of the application in a Java ICEfaces Web client.


To create the Quickstart ICEfaces projects:

1. From the menu, select *File>New>Example...* Then select the *QuickStart ICEfaces* Example and press *Next*.



2. Press the **Finish** button. Several projects are opened and shown in the explorer: *qs-xcore-model*, *qs-webmodel* and *qs-icefaces*. Open the g9 perspective and select it. The following is displayed:



 Currently, there are more projects displayed in the dialog (qs-service). Ignore it, it is for future improvements.

The *Quickstart* project *qs-webmodel* contains the model which includes the Domain Classes, the Object selections and Dialog models. The other projects are the definition of the Xcore domain model (*qs-xcore-model*) and a Java project that have been created for the generated code (*qs-icefaces*).

3. The system should begin building the project automatically and display the progress in the status bar. You might notice error markers (red x) next to the projects. These should disappear once the system finishes building the projects and Maven resolves the dependency issues.
4. If there are still error markers, it is possible the build did not occur automatically or correct. Close and Open all projects. If there are still error markers, you can select the menu option *Project > Build* or *Project > Clean*. If there are still errors, it is possible Maven did not resolve the dependencies. You can select all the projects then right-click on the project and select *Maven > Update Dependencies*.

## 8.1 JSF ICEfaces Code Generation

In this section you will generate client code for a web client that is based on ICEfaces, which is an open source framework based on the Java Server Faces 2 standard. The sample *Quickstart* application uses a Jetty web server and comes with the generated code for the various web objects. This sample will still take you through the steps for generating this code.

### 8.1.1 Prerequisites for ICEfaces

Necessary pre-installation steps is described in the [Installation](#) part.

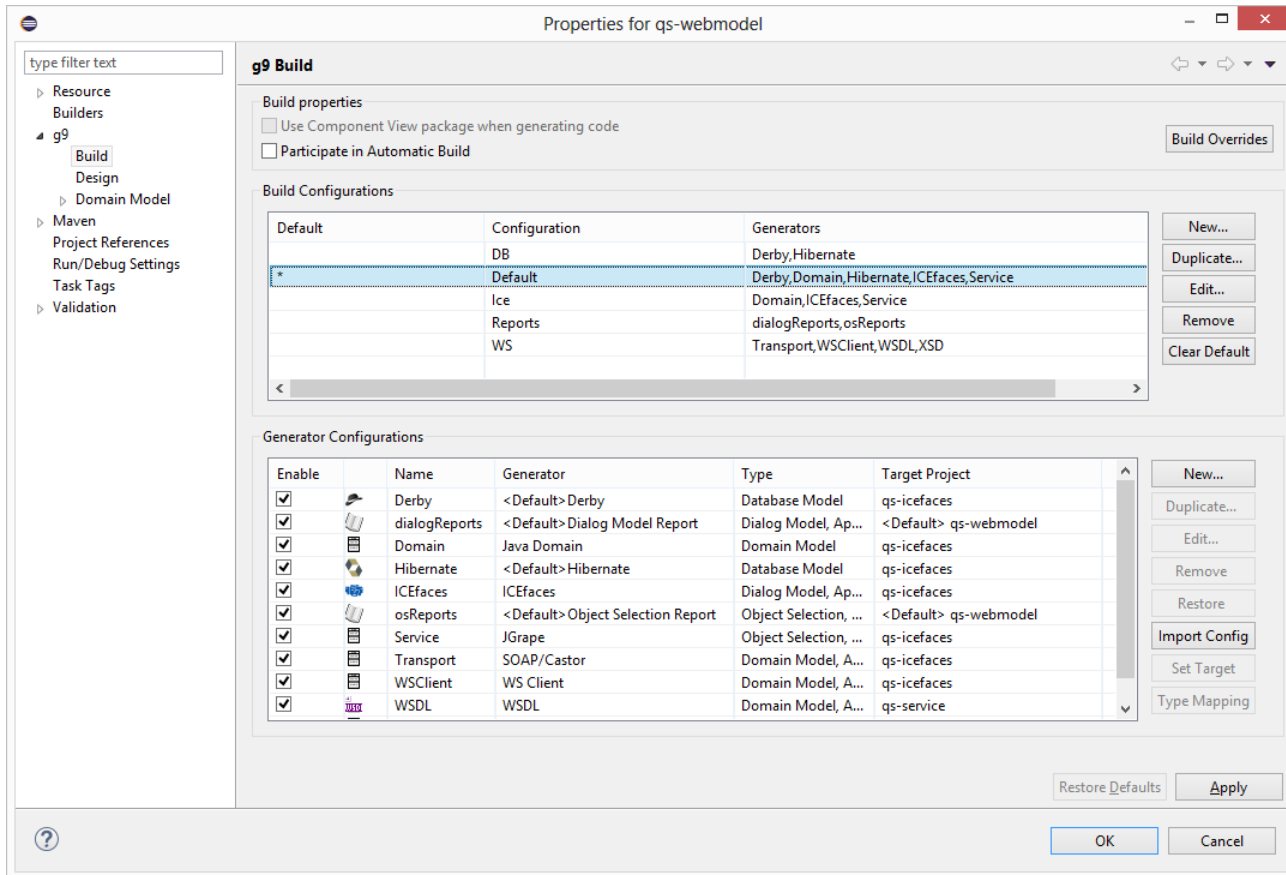
The application code that is generated is dependent on various libraries. These libraries should be automatically installed if you have successfully installed Maven and the Maven Eclipse plug-in. The required libraries are:

- *hibernate core*, along with all dependencies available from <http://hibernate.org> (Requires version 5.1.17.Final or later).
- *spring.jar*, available from <http://www.springframework.org>. (Requires version 4.3.25.RELEASE or later). Some Spring versions depend on *apache commons logging* 1.2 or later, available from <http://commons.apache.org/logging>.
- *log4j*, available from <http://logging.apache.org/log4j/2.x/>. (Requires version 2.11.2 or later)
- *derby.jar*, available from <http://db.apache.org/derby>. (Requires version 10.14.2.0 or later)
- *ICEfaces* version 4.1 or later, along with all dependencies (including JSF 2.2.10). Available from <http://www.icefaces.org>.
- *Jetty* version 9.4.19.v20190610 or later, along with all dependencies. Available from <http://www.eclipse.org/jetty>

### 8.1.2 ICEfaces Build Properties and Code Generation

g9 is structured to allow the generation of domain classes, client code, service code, database schema and hibernate mapping to occur alongside the Eclipse project build. You can configure the system to automatically generate code when saving content, or you can generate code manually. The Build section in the g9 project properties is where you can configure the code generation.

- Click the *qs-webmodel* project in the *Project Explorer* and then right-click and select Properties. When the Property Dialog appears, expand the *g9* node and then click on the *Build* node. The following dialog is displayed:



## Generator Configurations

You can create multiple *Generator Configurations* for the different components of the application. A *Generator Configuration* is a user-defined term that consists of a supported code generator, a set of parameter settings for that specific generator, and a specified Target Project for separating code among multiple Java projects.

In addition to the *Generator Configurations*, you can create multiple *Build Configurations* which consists of a series of Generator Configurations. These can be associated to g9 objects or be performed on Build/Build Other commands. A default Build is marked with an asterisk and will be performed on a Build command.

If you wanted, you could select the option '*Participate in Automatic Build*' and the code generation would be performed on save of g9 models.

### Generate all in one Build

The default Build Configuration contains all necessary Generator Configurations. Select the g9 project root and push F6 (accelerator for the Build menu). This will build all code in this project.

### Separation of generated and manually maintained code

The generators supports splitting of generated and manually maintained code into two separate source hierarchies. This behavior is triggered by setting the "Source directory" generator parameter to a different value than the "Target directory" parameter. With this behavior, if a file is present in the "Source directory" it will not be generated to the "Target directory". Default values are src-gen/main and src/main.

## Model specific Builds

If you want to see what the specific generators do, you may follow the instructions below. However, all necessary code are generated and ready to run.

### Console View

The Console View might show terminated views. To toggle between different Console Views, select *Display Selected Console* in the View toolbar.

### Domain Class Generator

The *Domain Generator* generates Java code for the domain model which includes java classes, interfaces and enumerations. The *Quickstart ICEfaces* project is configured to generate the domain classes into the *qs-icefaces* project.

1. Double click the **Domain** Generator Configuration in the bottom list to see the various standard and customized parameters and their values. Note the various parameters. Close the dialog.
2. Right click on the *qs-webmodel/Model View/Domain Model* node in the *Project Explorer* and select the menu option **Build**.
3. Click on the Console View tab. It should display a list of all the generated Java classes.

### Dialog Code Generator

The *Dialog Generator* generates code for the client part of an application. The code generated is based on the dialogs modeled and their object selections. There are one *Dialog Generator Configuration* for the *qs-webmodel* project. The *ICEfaces* configuration uses the ICEfaces generator for generating Java Server Faces based client and is configured to generate the classes into the *qs-icefaces* java project.

1. Double-click the **ICEfaces** configuration to display the parameters. Close the dialog and then close the *Properties for qs-webmodel*.
2. In the *Project Explorer*, expand the *Applications* node and right click on the **Recordshop** node and select **Build**. This starts the code generation process for the application main program and some helper classes.
3. Right click the Recordshop node again and select **Build Application > Ice**. This starts the code generation process for the complete client part of the application including application main and all dialogs.
4. Click on the Console View tab. It should display information regarding the code generation. If there is an error, it will be displayed here.

### Service Generator

The *JGrape Generator* generates code for the service part of the application. This includes the code for all necessary CRUD actions. The *qs-webmodel* project is configured to write all service code for the JSF clients into the *qs-icefaces* java project.

1. Open the *qs-webmodel* properties dialog again and double-click the **Service** configuration to display the parameters. Note the location of the *ServicePackage*. Close the dialog and also the properties dialog.
2. In the *Project Explorer*, right click the *Object Selections* node and select the menu **Build**. This starts the code generation process for the services for the ICEfaces client.
3. Click on the Console View tab. It should display the various code elements that were generated.

### Database Schema Generator

The Database Generator is used to generate database schema scripts to help in creating databases. It works from the database model that has already been provided for the Derby database.

- Open the *qs-webmodel* properties dialog and double-click the **Derby** configuration to display the parameters. Note the target project is set to *qs-icefaces* and the target directory is *db*. Close the dialog.

## Hibernate Mapping Generator

The *Hibernate Generator* is used to generate data access code. The system will generate a mapping file named `<Classname>.hbm.xml` for each domain class specified in the database model file. Also created is a `G9Dataaccess.xml` file which contains references to all the domain class mapping files as well as the `hibernate.properties` file which contains JDBC and EJB connection properties.

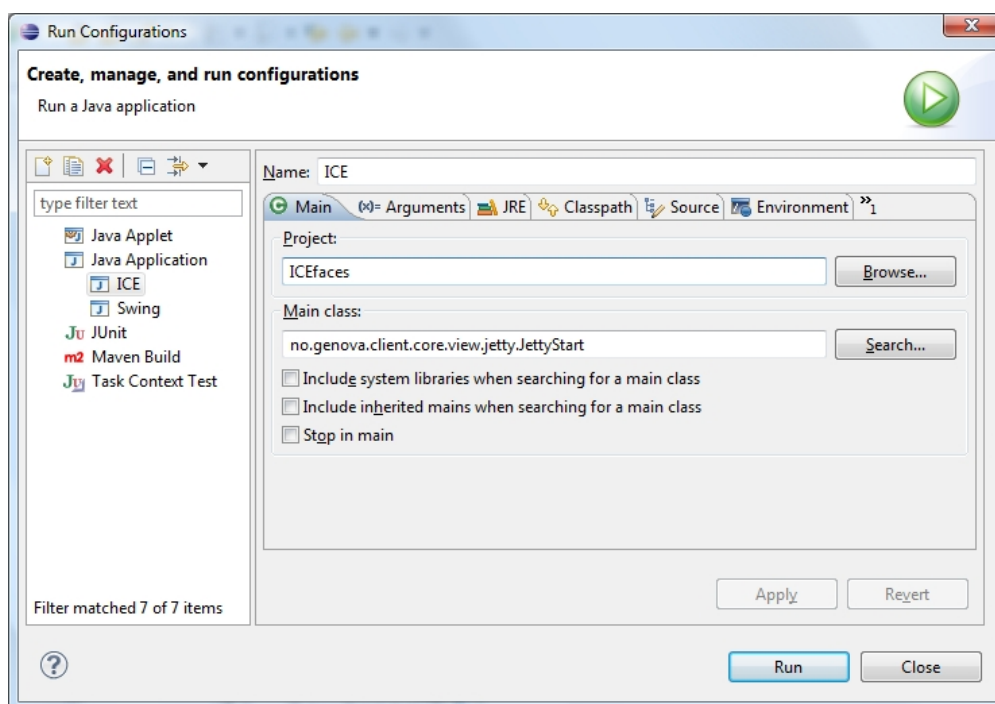
1. Double-click the **Hibernate** configuration to display the parameters. Note the target project is set to `qs-icefaces` and the target directory is `src-gen/main`. Close the dialog.
2. In the *Project Explorer*, expand the *Model View/Database Models* node and right-click on the *recordshop* node and select the menu **Build**. This starts the code generation process of the database schema files and the hibernate mapping files.
3. Click on the Console View tab to display the various code elements that were generated.

## 8.1.3 Compiling and running the ICEfaces application

The sample project creates the database populated with data for you and it is stored in the db directory named *recordshop*. You can create/recreate it, see instructions on [Creating Derby Database](#).

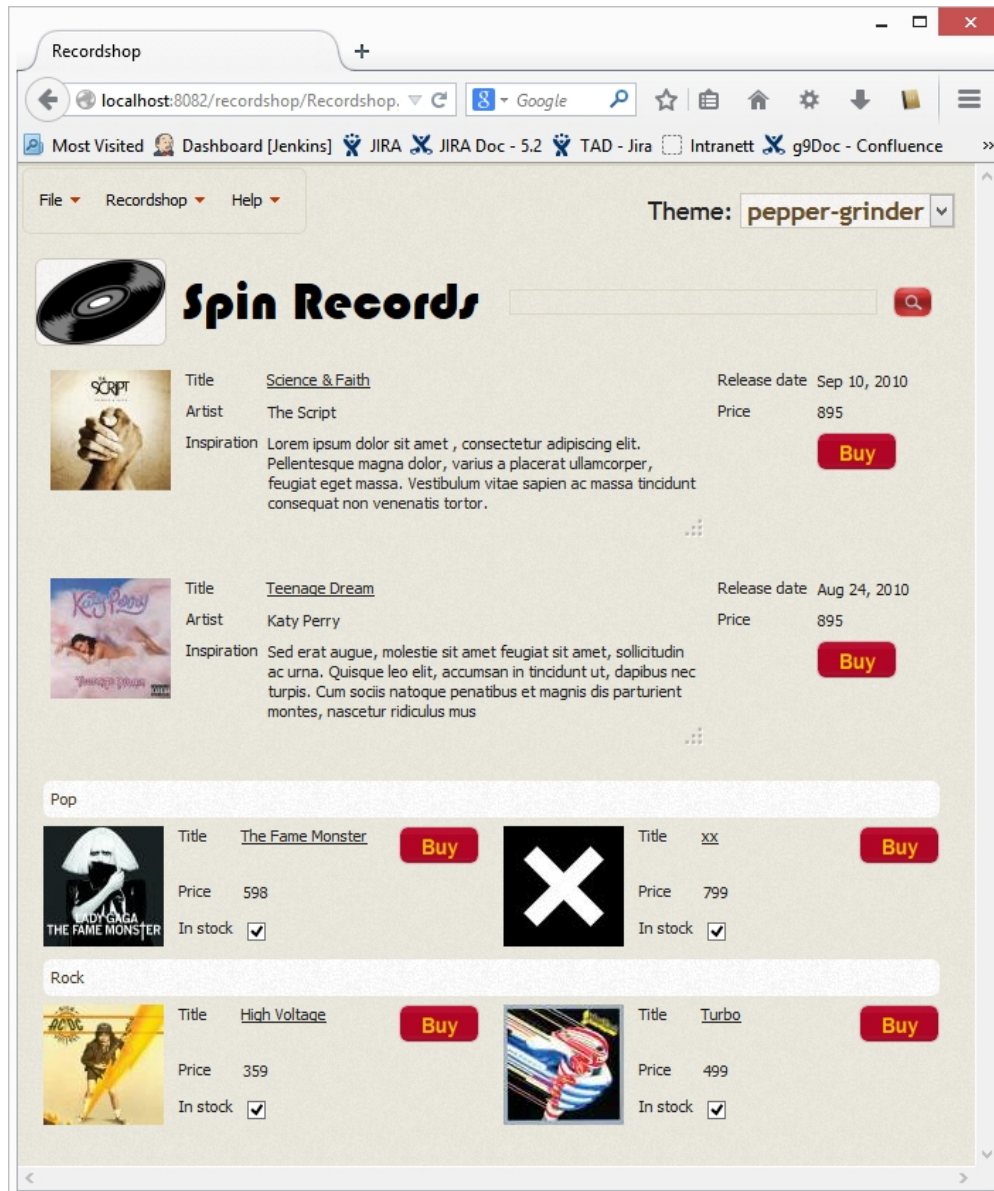
You are now ready to compile and run your ICEfaces application.

1. Switch to the Java perspective by clicking on the Java perspective button at the top right. (Or select menu option *Window>Open perspective>Other>Java (default)*)
2. Click on the *qs-icefaces* java project and press the F5 button to refresh
3. Check the menu option *Project>Build Automatically*. If it is checked, then the code should have automatically compiled. If it is not checked, select the menu option *Project > Build All*.
4. Select the *Run > Run configurations...* menu option. This opens up the Run Configurations Dialog
5. Expand the *Java Application* node and click the *ICE* node. This has already been created for you.

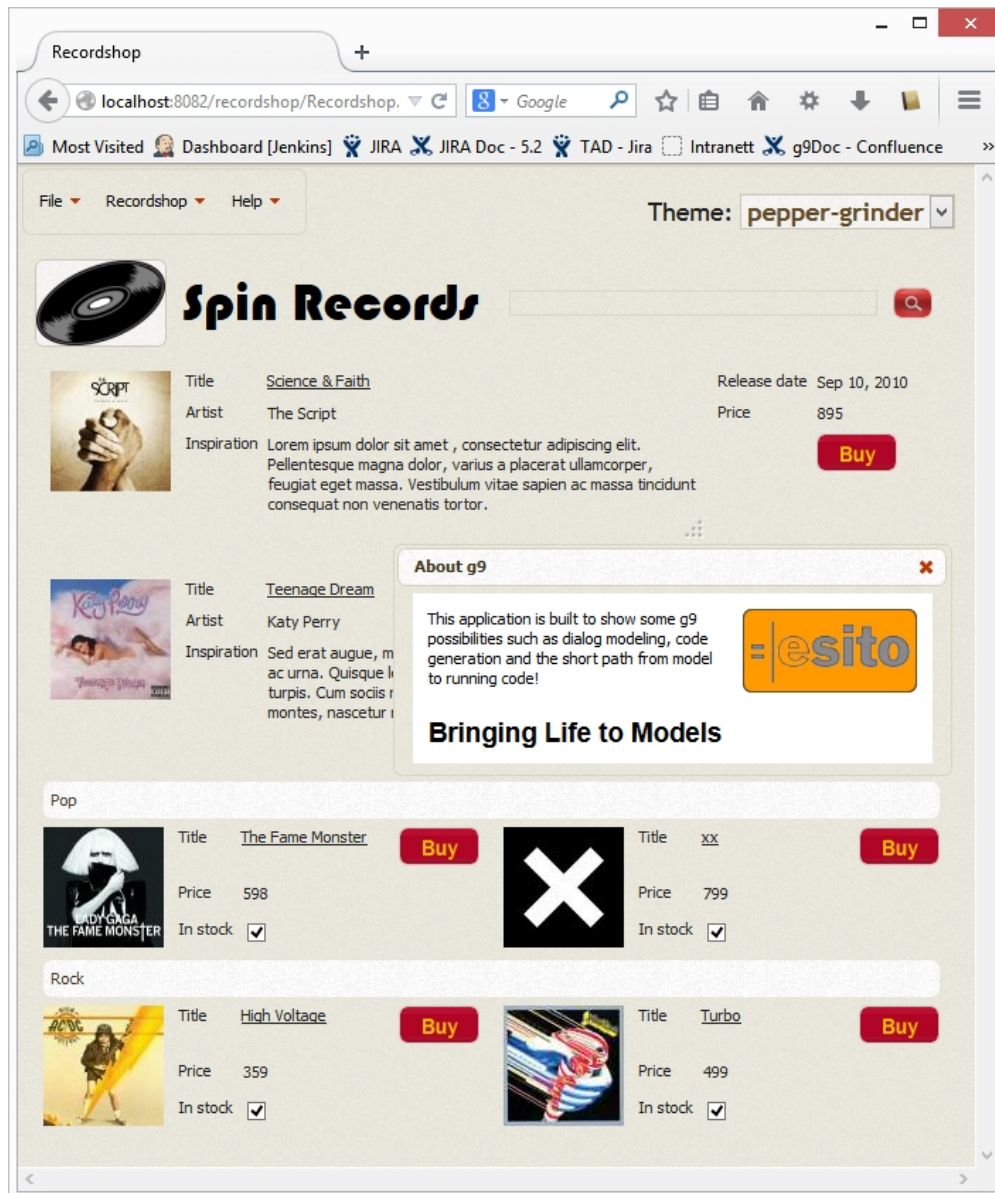




- Click the Run button to start the Jetty web server.
- Check the Console. The last line should indicate whether the web server has started
- Open the URL <http://localhost:8082/recordshop> in a web browser. The following is displayed:

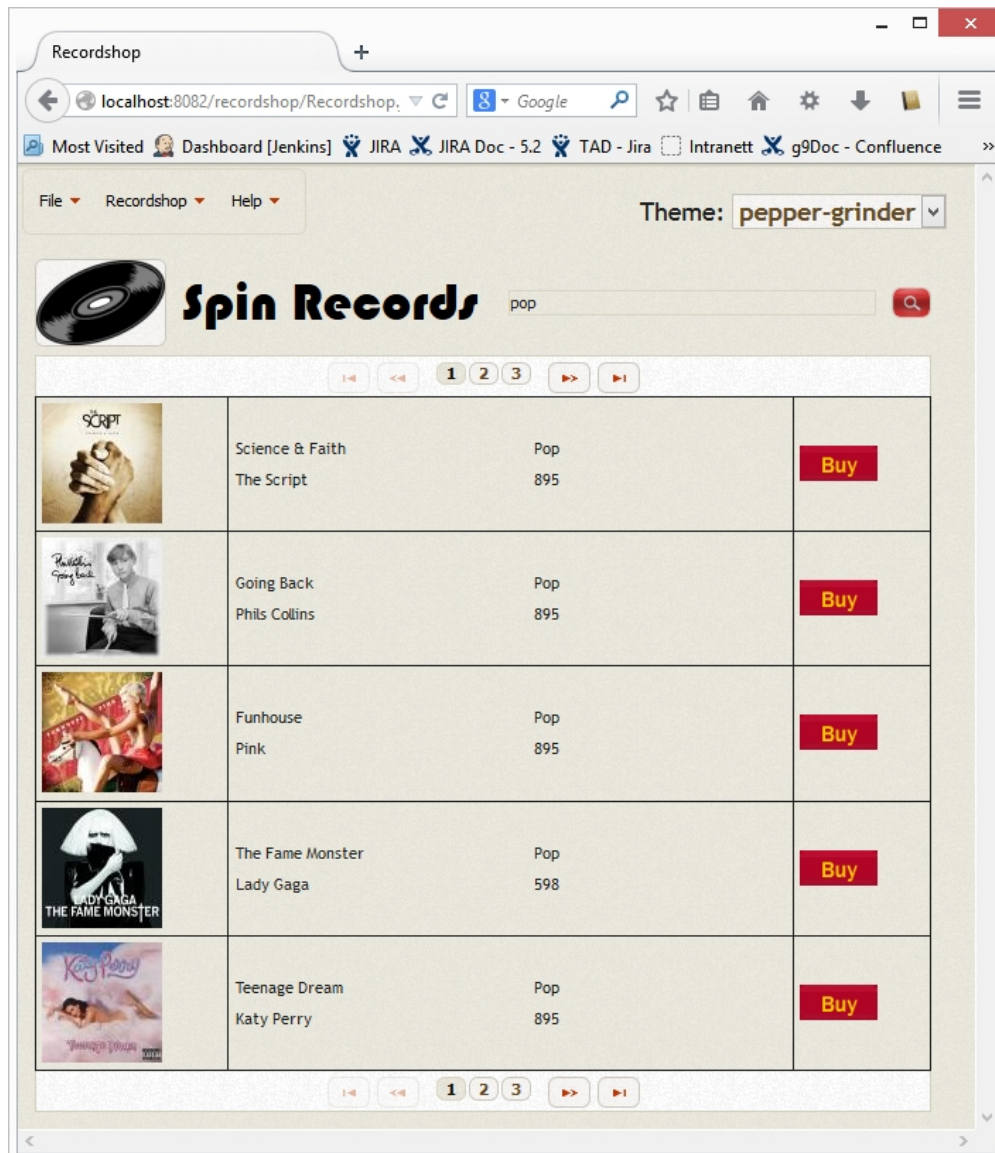


- Select the menu Help > About from the menu bar at the top. The modified dialog is displayed:



10. Close the About dialog by clicking on the x in the top right or the logo button.
11. Enter the category **pop** into the search field at the top and then click the search button to the right. The following dialog is displayed:



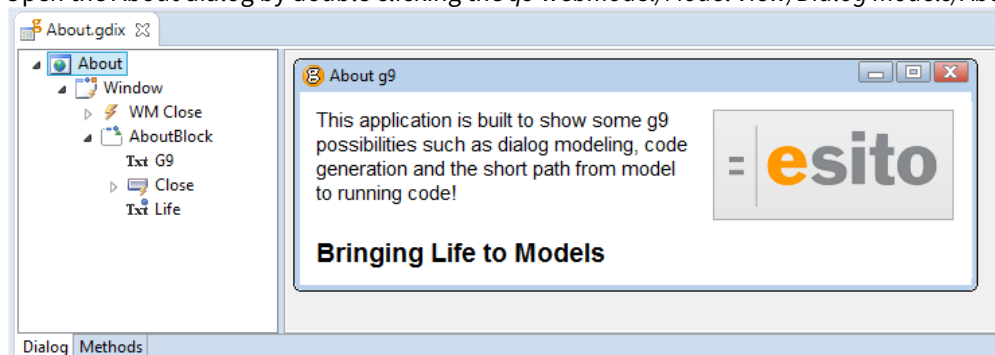


12. Close the Jetty Server by clicking the red 'Terminate' button in the Console view.

### 8.1.4 Edit the About Dialog Box Model

In this section, you will modify the user-interface dialog *About g9*, regenerate the code and rerun the application. First, read more about g9 model artifacts in [Modeling your Dialogs](#).

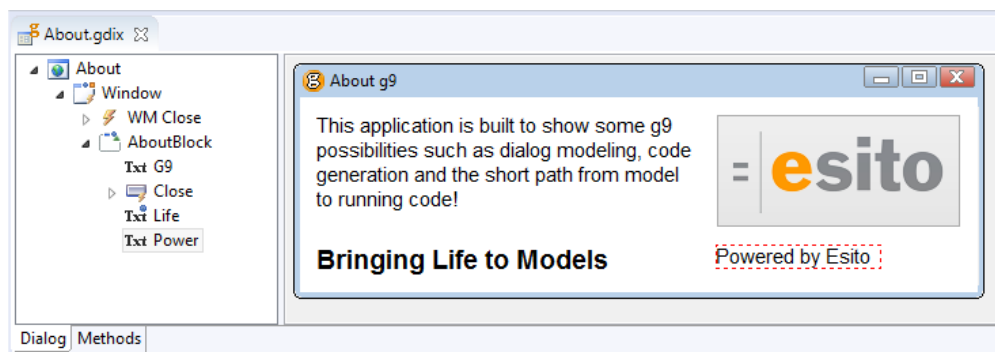
1. Open the About dialog by double clicking the *qs-webmodel/Model View/Dialog models/About* node.



2. Expand the *Non-data widgets* expand bar in the palette and click on the **Text** item so that it is highlighted.
3. Move the cursor so that it hovers over the right side of the **Bringing Life to Models** text field, then click and enter the name *Power*. This adds the text field to the right of that field, just below the *esito* logo.
4. Click on the newly added field and change the properties as follows:

<i>Widget.Text</i>	<b><i>Powered by Esito</i></b>
<i>Style.Style</i>	<b><i>WebSmallText</i></b>

5. Your dialog should look as follows:

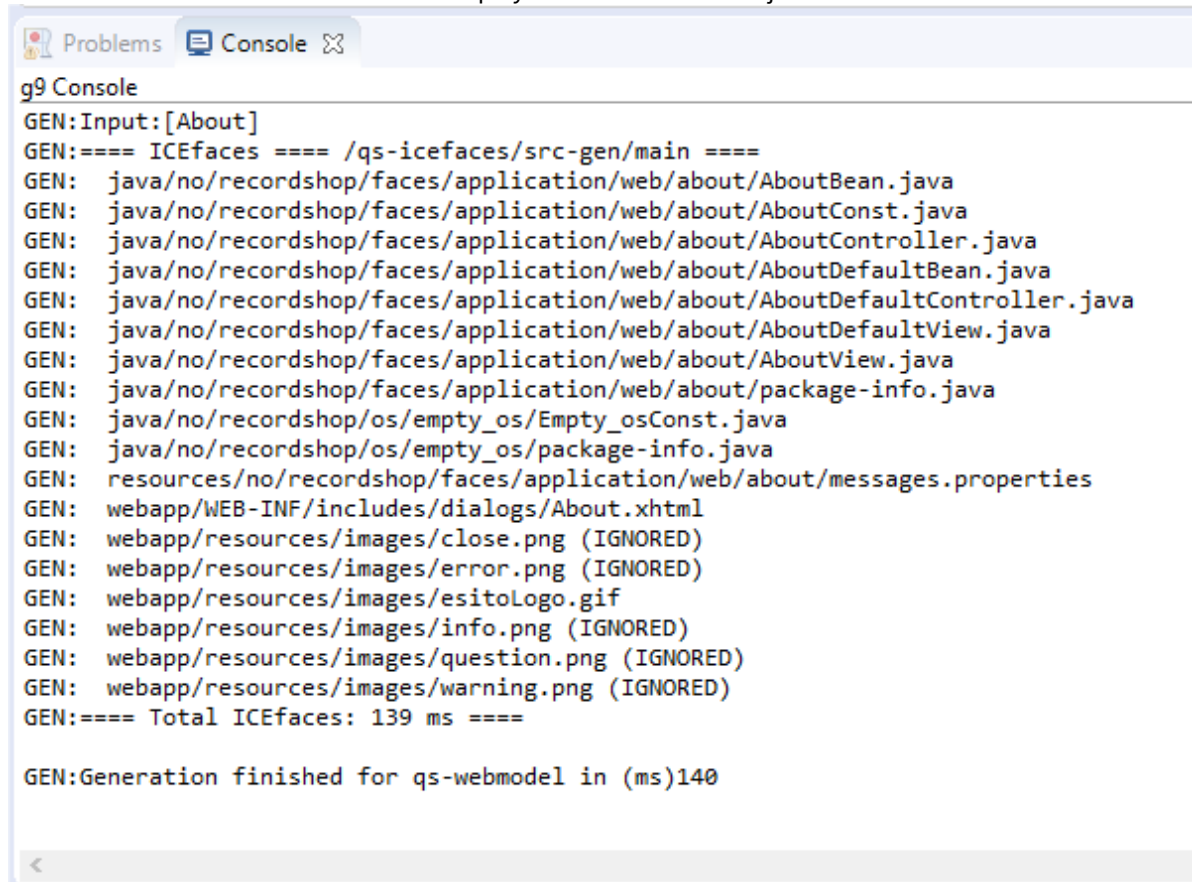


6. Save the dialog by clicking on the save icon in the main toolbar.

Generate the client code for the modified Help About dialog:

1. Click the *qs-webmodel* project in the *Project Explorer* and then right-click and select *Properties*. When the *Property Dialog* appears, expand the *g9* node and then click on the *Build* node.
2. Double-click the **ICEfaces** configuration to display the parameters. Close the dialog and then close the *Properties for qs-webmodel* dialog.
3. In the *Project Explorer*, expand the *qs-webmodel/Dialog Models* node and right click on the *About* node and select **Build**.

- Click on the Console View tab. It should display a list of all created objects



The screenshot shows the Eclipse IDE's Console View tab. The title bar indicates 'g9 Console'. The output text is as follows:

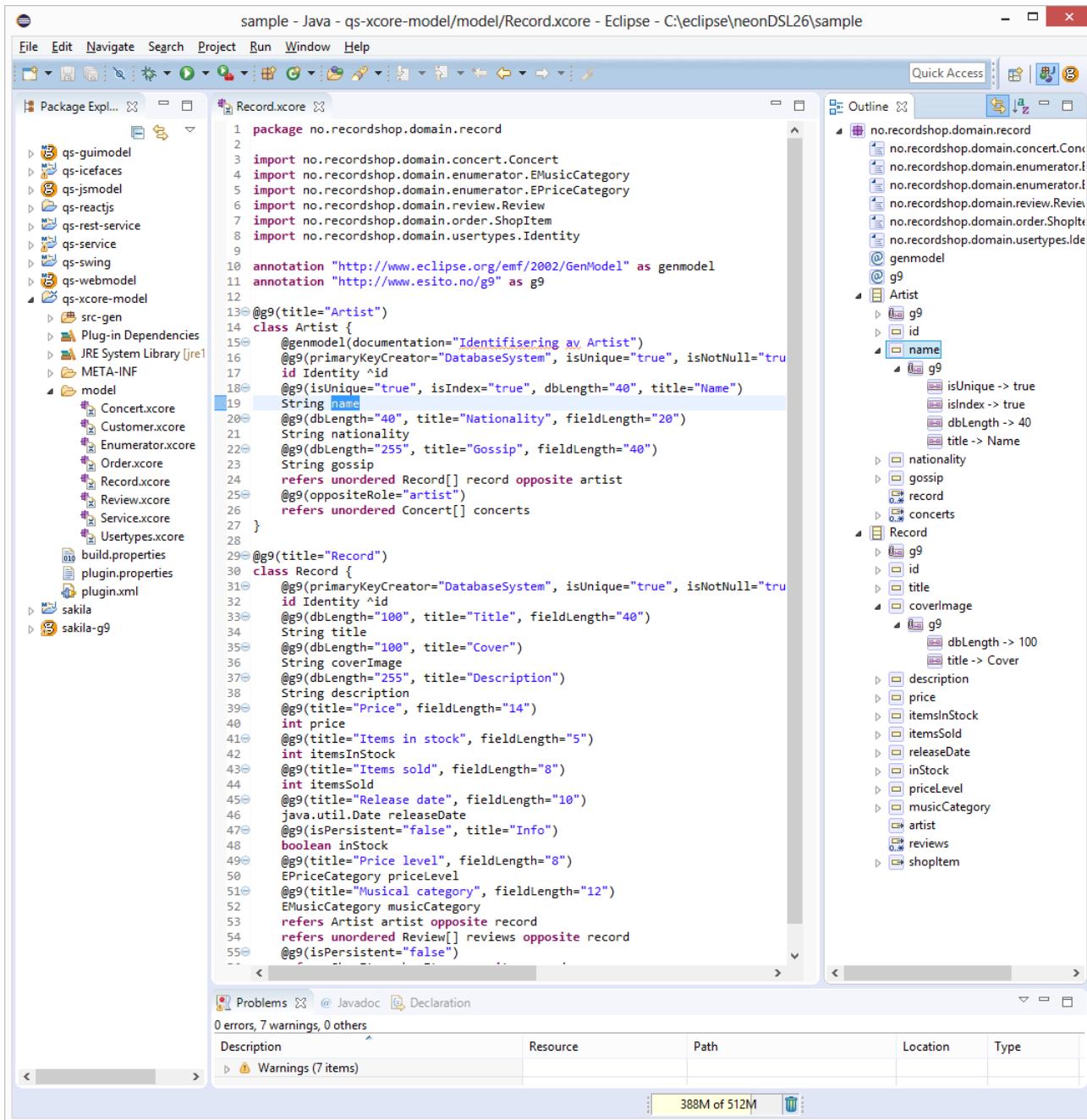
```
GEN:Input:[About]
GEN:==== ICEfaces ==== /qs-icefaces/src-gen/main ====
GEN:  java/no/recordshop/faces/application/web/about/AboutBean.java
GEN:  java/no/recordshop/faces/application/web/about/AboutConst.java
GEN:  java/no/recordshop/faces/application/web/about/AboutController.java
GEN:  java/no/recordshop/faces/application/web/about/AboutDefaultBean.java
GEN:  java/no/recordshop/faces/application/web/about/AboutDefaultController.java
GEN:  java/no/recordshop/faces/application/web/about/AboutDefaultView.java
GEN:  java/no/recordshop/faces/application/web/about/AboutView.java
GEN:  java/no/recordshop/faces/application/web/about/package-info.java
GEN:  java/no/recordshop/os/empty_os/Empty_osConst.java
GEN:  java/no/recordshop/os/empty_os/package-info.java
GEN:  resources/no/recordshop/faces/application/web/about/messages.properties
GEN:  webapp/WEB-INF/includes/dialogs/About.xhtml
GEN:  webapp/resources/images/close.png (IGNORED)
GEN:  webapp/resources/images/error.png (IGNORED)
GEN:  webapp/resources/images/esitoLogo.gif
GEN:  webapp/resources/images/info.png (IGNORED)
GEN:  webapp/resources/images/question.png (IGNORED)
GEN:  webapp/resources/images/warning.png (IGNORED)
GEN:==== Total ICEfaces: 139 ms ====

GEN:Generation finished for qs-webmodel in (ms)140
```

- Restart the application to see the result.

## 9 Working with the Xcore domain model

The domain model used in these projects is written using Xcore. The model source is stored in the *qs-xcore-model* project under the directory *model*. The screenshot below shows the definition of Artist and Record classes.



### 9.1 Synchronizing changes

If you changes classes, attributes or annotations in the model files, you must synchronize the changes into the g9 model project. See Setting and updating the Domain model.

## 10 Creating Derby Database

If you have changed the model and need to drop the database and produce a new one, follow the description below.

### 10.1 Installing Apache Derby Database

The Apache Derby database is a SQL compliant Java based relational database that is embedded in the *Quickstart* sample application. Although g9 supports multiple databases, the Derby database was chosen for its ease of use and small footprint.

To install Derby:

1. Go to [http://db.apache.org/derby/derby\\_downloads.html](http://db.apache.org/derby/derby_downloads.html)
2. Select the link for the latest official release (version 10.14.2.0 or later)
3. Select the bin distribution ([db-derby-10.14.2.0-bin.zip](#)) and download it.
4. Extract the zip file into a local directory (i.e. **c:\java**). The Derby database should reside in your local directory (i.e **c:\java\db-derby-10.14.2.0-bin**)

### 10.2 Creating the schema and Derby database

The Quickstart sample applications specifies *Derby* as the target database system. The g9 code generators create script files for the specified target database. In this case, a script file is generated for the creation of the Derby database, another script file is generated for the schema definition and another for the index creations.

To create the database, you must have installed the Derby database and you must have database administrator privileges.

In addition to the generated script files, various manual script files have been created to facilitate the creation of the schema and to populate the database with sample data.

The sample projects create the databases populated with data, so this step should not be necessary. If the database exists, it is stored in the db directory and named *recordshop*.

To create the database, schema and populate the database with sample data:

1. Select the Eclipse Java project, either *qs-swing*, *qs-icefaces* or *qs-rest-service*, depending on the context. Denoted here as *qs-<project>*, expand the *qs-<project>\db* node in the Project Explorer and right-click on the ***create-recordshop.cmd*** file and select *Open With>Text Editor*
2. Ensure that the `DERBY_HOME` variable in the script is set to the correct directory where Derby was previously installed (i.e. **c:\java\db-derby-10.14.2.0-bin**). If not, change the directory and save the file.
3. Right-click the ***CreateDerby.launch*** file and select *Run As > CreateDerby*. The launch file starts the ***create-recordshop.cmd*** script, which deletes the database and creates it again.
4. Click on the ***Console*** tab to display the output of the schema creation and data population.
5. To verify that the database was created:
  - Swing: Check that a *recordshop* folder is created in the *qs-swing/db* directory.
  - ICEfaces: Check that a *recordshop* folder is created in the *qs-icefaces/db* directory.
  - ReactJS: Check that a *recordshop* folder is created in the *qs-rest-service/db* directory.