



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Composer Help

Project Types and Directories

Project Types and Directories

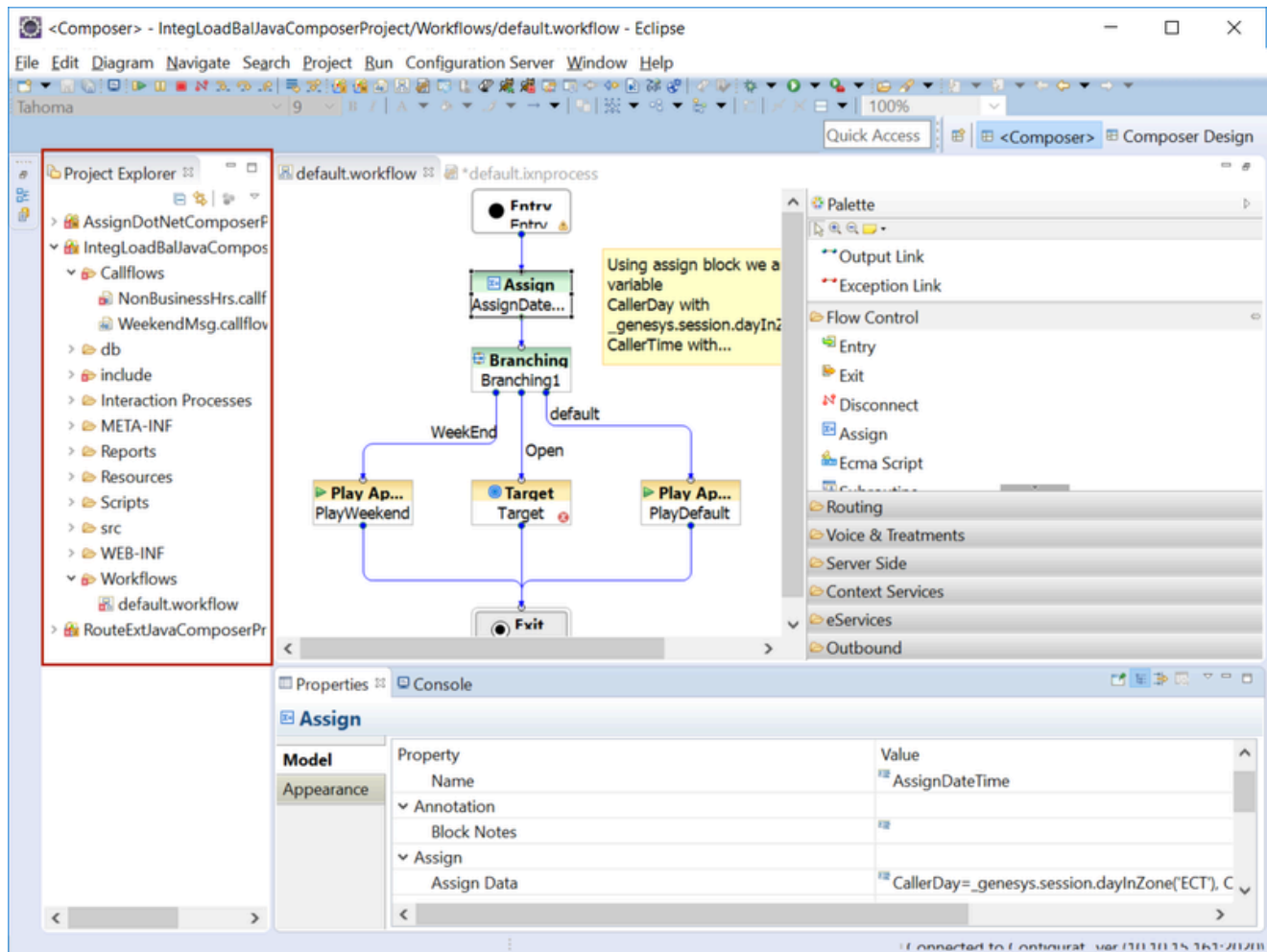
Contents

- **1 Project Types and Directories**
 - 1.1 Java and .NET Projects
 - 1.2 Voice Application Project Types
 - 1.3 Routing Application Project Types
 - 1.4 Project Structure/Directories
 - 1.5 Sub-Directories
 - 1.6 Folders Created When Upgrading Projects and Diagrams
 - 1.7 Adding Files to an Existing Project
 - 1.8 Project Permissions
 - 1.9 Using Composer Shared Subroutines
 - 1.10 Project Properties

A Composer *Project* is associated with either:

- A voice application for **Genesys Voice Platform** or
- A routing application for the **Orchestration Platform**.

In general, a Project consists of a predefined, structured set of files and folders that contain all resources for the application. See the **Project Explorer** below.



For information on Projects referencing other Projects, see the figure in topic **Project Properties dialog box**. Expand Project References.

Java and .NET Projects

There are two types of Composer Projects:

- **Java Composer Projects** -- Use JSP and Java to implement custom business logic. These Projects can be **deployed** on web applications servers as described in the **Composer 8.1 Deployment Guide**. See **Creating a New Project**.
- **.NET Composer Projects** -- Use ASP.NET and C# to implement server-side blocks and custom business logic. The Project can only be **deployed** to Microsoft IIS. See **.NET Troubleshooting** for steps for working with Composer .NET Projects when a machine does not have WSE 3.0.

Starting a New .NET Project

To start a new .NET Project:

1. Click the Create a NET Composer Project button in the menu bar. You can also click the button above the Project Explorer and select **Composer > Composer > Projects > .NET Composer Project**.
2. In the Project dialog box, type a name for your Project.
3. If you want to save the Composer Project in your default workspace, select the **Use default location** check box. If not, clear the check box, click Browse, and navigate to the location where you wish to store the Composer Project.
4. Select the Project type.
5. Click **Next**.
6. If you want to use templates, expand the appropriate Project type category and select a template for your application.

.NET Project Warning

.NET Projects may show this warning in the Console View: `include\getWebServiceData.aspx(482): warning CS0618: 'Microsoft.Web.Services3.SoapContext.Security' is obsolete: 'SoapContext.Security is obsolete. Consider deriving from SendSecurityFilter or ReceiveSecurityFilter and creating a custom policy assertion that generates these filters. This warning can be ignored and no workarounds are needed. It will not show up as an error or warning in the Problems View.`

.NET Project Logging

Starting with 8.1.410.14, .NET Project logging configuration is handled with file `web.config` located in the Project root directory. The level of logging can be changed to values:

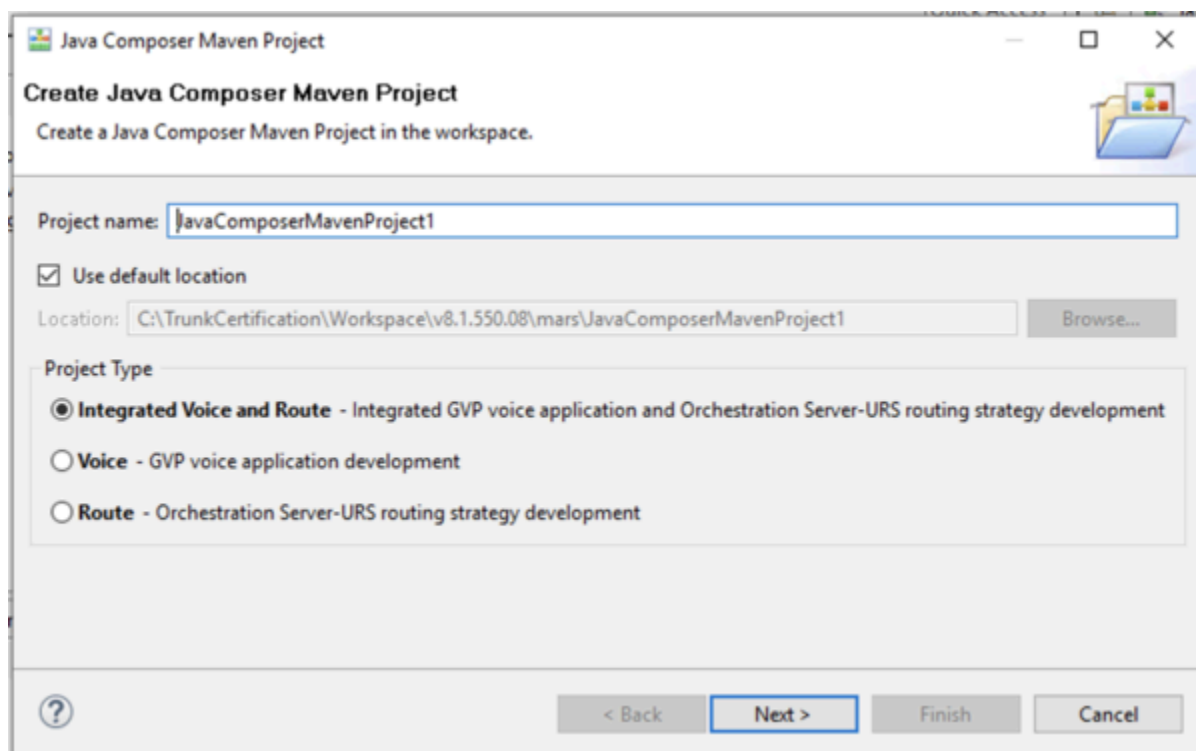
OFF	The highest possible rank and is intended to turn off logging.
FATAL	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARN	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	Detailed information on the flow through the system. Expect these to be written to logs only.

TRACE More detailed information. Expect these to be written to logs only.

Maven Support for Building Java Projects

Beginning with release 8.1.550.08, Composer supports Maven for building Java projects. Maven is a software project management and comprehension tool by Apache. It is a build automation tool used primarily for Java projects. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. A Java Composer Maven project's structure and contents are declared in an XML file, `pom.xml`.

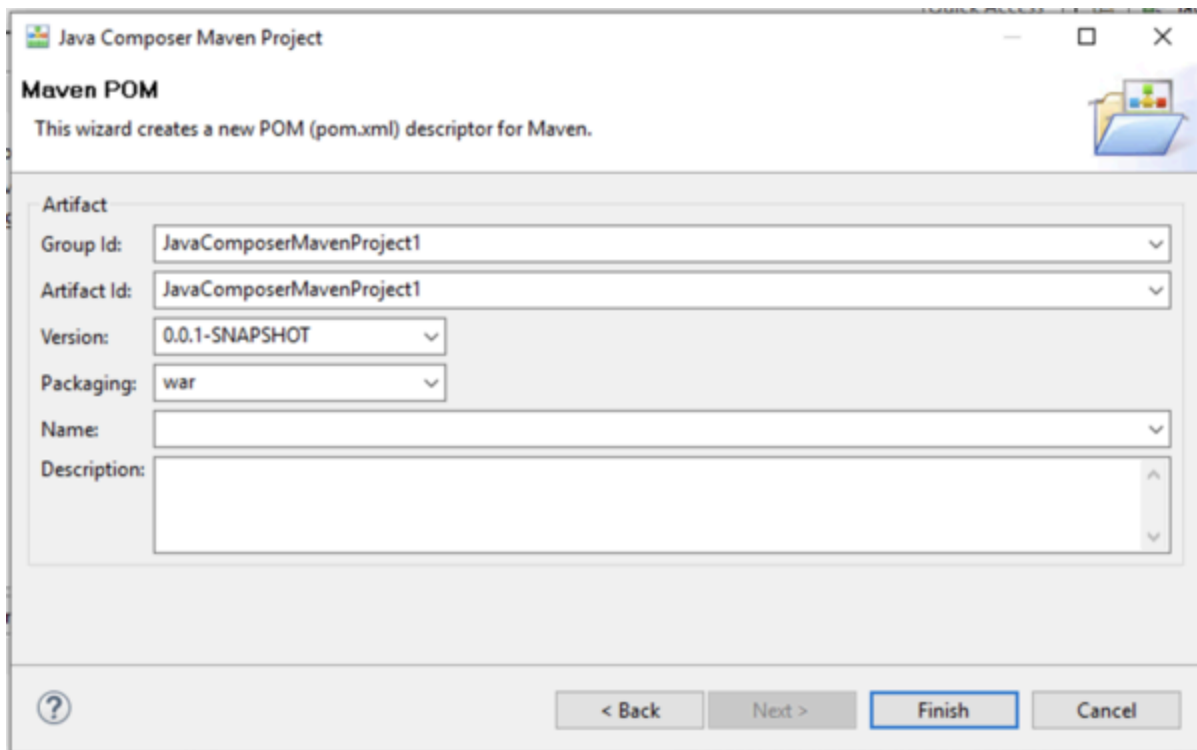
A new project type, **Java Composer Maven Project**, is introduced in Composer's project creation wizard for Java projects. A Maven project will inherit all the features and functionalities of a regular Java Composer project in addition to the Maven specific features.



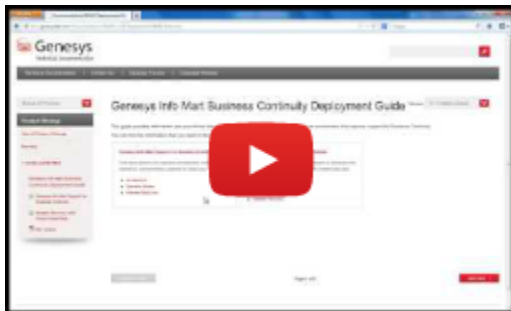
A new button is introduced in the Main toolbar to create a new Java Composer Maven project from the toolbar.



A new page, **Maven POM**, is introduced as part of the project creation wizard to specify Maven related values such as GroupID and ArtifactID.



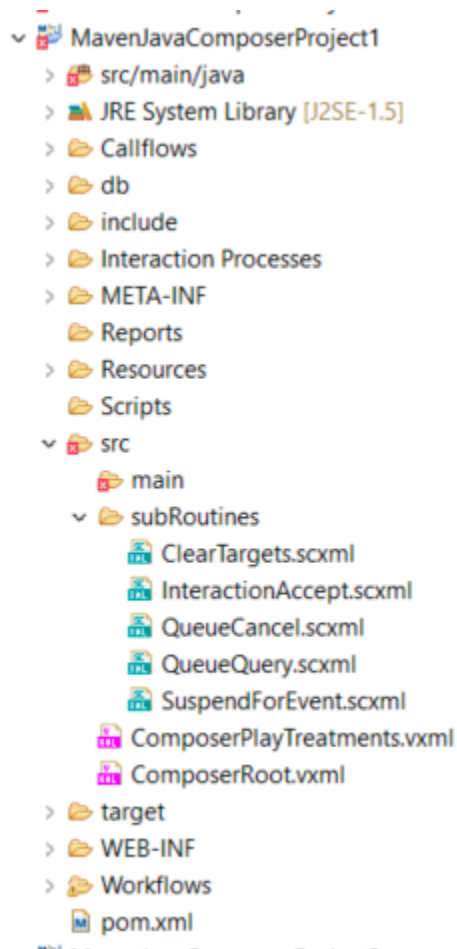
Here is a short video on creating a Maven project in Composer:



The **Maven Project Builder** is added to a Composer project on creating a new Java Composer Maven project.



The following a sample Java Composer Maven project's folder structure.



Maven Build Configuration and Goals

The Maven build follows a specific life cycle to deploy and distribute target projects. Each life cycle consists of a sequence of phases. The **default** build life cycle consists of 23 phases as it is the main

build lifecycle.

On the other hand, the **clean** life cycle consists of 3 phases, while the **site** lifecycle is made up of 4 phases.

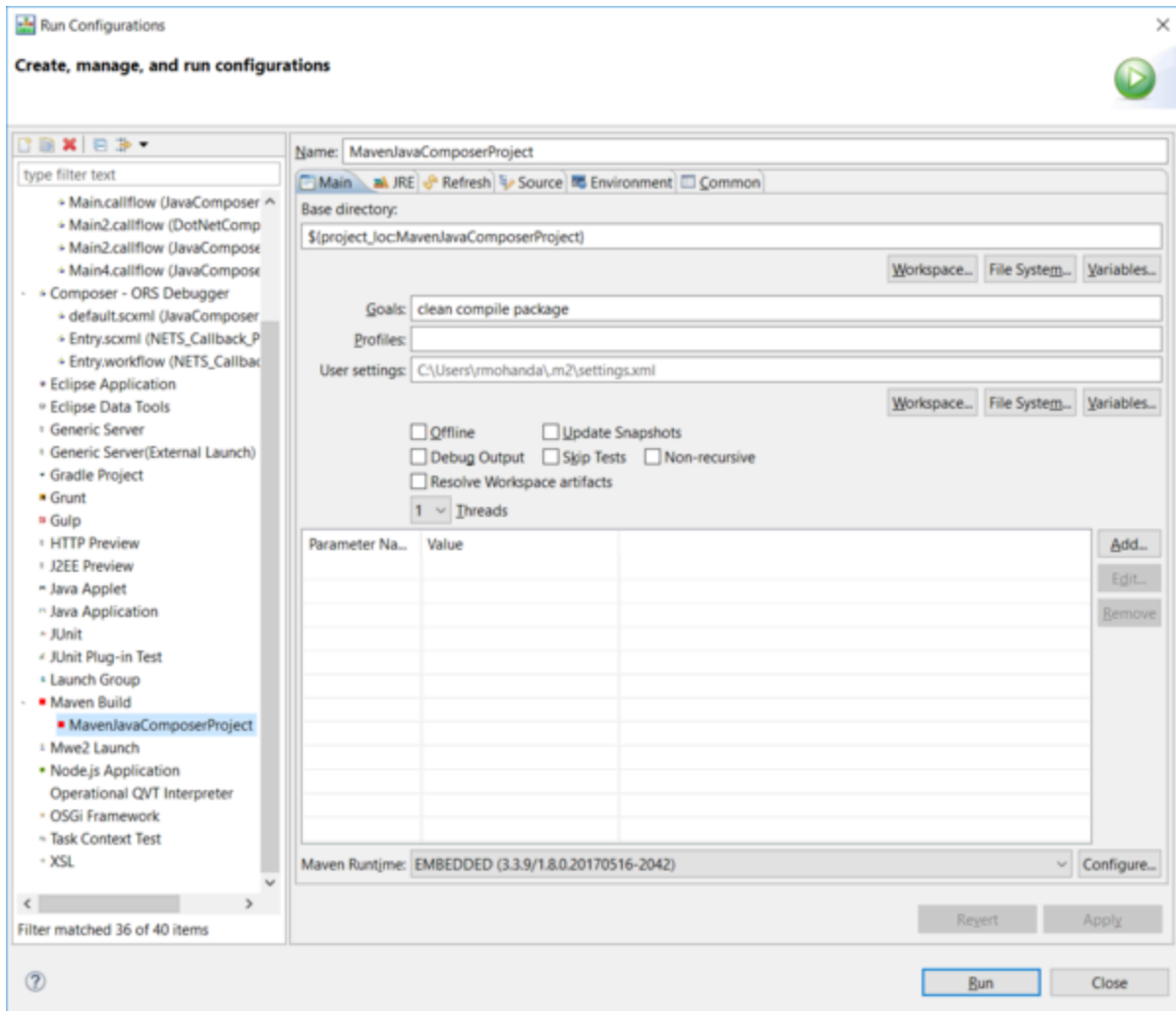
Each phase is a sequence of goals, and each goal is responsible for a specific task.

When you run a phase, all goals bound to the phase are executed in order.

The following is a list of some of the phases and default goals bound to them:

- `compiler:compile` – the compile goal from the compiler plugin is bound to the **compile** phase
- `compiler:testCompile` is bound to the **test-compile** phase
- `surefire:test` is bound to the **test** phase
- `install:install` is bound to the **install** phase
- `jar:jar` and `war:war` are bound to the **package** phase

When using Maven to build a Java Composer project, you must specify the goal in order to be able to build the project. For example, in the screenshot below, a **clean** compile package has been specified as the goal:



The following is a sample screenshot of the Maven runtime console view:

```
Console
<terminated> MavenJavaComposerProject [Maven Build] C:\Program Files (x86)\Java\jre1.8.0_161\bin\javaw.exe (18-Dec-2019, 12:23:58 AM)
[INFO] -----
[INFO] Building JavaComposerProject3 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ JavaComposerProject3 ---
[INFO] Deleting C:\testComposerProject11\MavenJavaComposerProject\target
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ JavaComposerProject3 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\testComposerProject11\MavenJavaComposerProject\src\main\resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ JavaComposerProject3 ---
[INFO] No sources to compile
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ JavaComposerProject3 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\testComposerProject11\MavenJavaComposerProject\src\main\resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ JavaComposerProject3 ---
[INFO] No sources to compile
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ JavaComposerProject3 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\testComposerProject11\MavenJavaComposerProject\src\test\resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ JavaComposerProject3 ---
[INFO] No sources to compile
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ JavaComposerProject3 ---
[INFO] No tests to run.
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ JavaComposerProject3 ---
[WARNING] JAR will be empty - no content was marked for inclusion!
[INFO] Building jar: C:\testComposerProject11\MavenJavaComposerProject\target\JavaComposerProject3-0.0.1-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.473 s
```

Important

A Maven project augments the capability of Composer to build (or compile) the Java and Composer sources (VXML & SCXML) seamlessly. With this support, users can directly consume or access Composer's backend APIs (`composerBackend.jar`) in their Java counterparts (`src/main/java`).

WebSphere Application Server Files

The custom **Build.xml** and Maven **Pom.xml** files will not create the **ibm-web-bnd.xmi** and **jboss-web.xml** WebSphere Application Server files, respectively. Users must create these files manually and place it under the project's WEB-INF directory. Once created, the files are updated automatically.

Sample **ibm-web-bnd.xmi** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<webappbnd:WebAppBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:webappbnd="webappbnd.xmi" xmi:id="WebAppBinding_1276009185886"
virtualHostName="default_host">
```

```
<webapp href="WEB-INF/web.xml#WebApp_ID"/>
<resRefBindings xmi:id="ResourceRefBinding_1276009394684" jndiName="jdbc/pooledDS">
<bindingResourceRef href="WEB-INF/web.xml#ResourceRef_1276009394684"/>
</resRefBindings>
</webappbnd:WebAppBinding>
```

Sample *jboss-web.xml* file:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
  <resource-ref>
    <res-ref-name>jdbc/oraclePooled</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <jndi-name>java:comp/env/jdbc/oraclePooled</jndi-name>
  </resource-ref>
</jboss-web>
```

Important

The generic WAR Export wizard generates these files only during WAR export. In other instances, users must create these files manually.

Voice Application Project Types

Voice applications are VoiceXML applications with full support for the Genesys Voice Platform. A Voice application can be deployed on a web application server that meets the minimum prerequisites described in the [Composer 8.1 Deployment Guide](#). Also see [Creating Voice Applications for GVP](#).

Routing Application Project Types

Routing applications are SCXML applications with full support for the modules described in the [Orchestration Developer's Guide](#). A Routing application can be deployed on a web application server meeting the minimum prerequisites described in the [Composer 8.1 Deployment Guide](#). Also see [Creating a New Routing Project](#).

Project Structure/Directories

A Composer Project (Java or .NET) will contain some or all of these subfolders depending on the type of Project:

- App_Code -- .NET Composer Projects only. This folder will be empty by default as Composer bundles all the C# classes in to the ComposerBackend.dll file. Custom C# classes will also go into this folder.
- bin -- Any libraries used in a .NET Composer Project go here.

- Callflows -- Folder for storing all the callflow diagrams (.callflow files)
- db -- **Database connection.properties** and .sql files are stored here.
- include -- Composer-provided standard include files used by **Backend** logic blocks.
- debugging-results -- The **Run As** option on a VXML file creates a debugging-results folder when no debugging-results folder exists in the Project.

Custom JavaScript files (.js) can be included in a routing application by placing the file(s) in the include/user folder. Re-generating code for all IPD diagrams in the project is required after placing the files. The JavaScript functions in the specified .js file can then be used from any Workflow block that supports writing expressions e.g. the Assign, Branching and ECMAScript blocks.

- META-INF -- Created when you create a new Java Composer Project. It is needed for Java and is included when a .war file is exported from Composer. Do not make changes to this directory.
- WEB-INF/lib -- Java Composer Projects only. Folder for external dependency libraries such as JAR files. Note: The Tomcat application server should be restarted after changing any JAR files in this folder.
- Interaction Processes -- Folder for storing all the **interaction process diagrams** (.ixnprocess files).
- Resources -- Folder for the audio and grammar resources. Resources/grammars -- Folder for **Grammar Builder** (.gbuilder files) and GrXML files.
 - Resources/grammars/<language code> -- Place language-specific grammars here (such as en-US or es-MX folders).
 - Resources/prompts -- Folder for prompts files.
 - Resources/prompts/<language code> -- Place language-specific prompts here. If the application language is changed mid-call using a Set Language block, prompts audio resource paths in these language folders will be translated to the current language at run time.
- Scripts -- Folder for user-written ECMAScript. Custom JavaScript files (.js) can be included in a voice application by placing the file(s) in the Scripts folder.
- src-gen -- Folder for the code generation VXML/SCXML files.
- upgradeReports -- When **migrating IRD strategies into Composer**, folder for migration reports. Also used for reports as result of **upgrading Projects and diagrams**.
- src -- Folder for custom code such as backend logic pages written by the user. The **ComposerPlayTreatments** VXML file is located here.
- Workflows -- Folder for storing all the workflow diagrams (.workflow files).

Static VXML/SCXML code is generated with the name of the Composer diagram file. The code will be saved in the src-gen folder under the current active Project. The two types of Projects have different Project natures. Based on these Project natures, different builders, editors and preferences are associated with the Projects. For example, .NET Composer Projects and Java Composer Projects have different preferences for deployment since they are deployed to different web/application servers.

Project Folders and Resources

Common Project folders and resources (Resources, src, include, db, and so on) do not change between .NET and Java Projects. Project Export and deployment procedures related resources will differ between .NET and Java Projects (bin folder for .NET and WEB-INF, META-INF folders for Java).

For more information, see [Migrating a Composer Application From Lab to Production](#).

Sub-Directories

Composer 8.1.440.18 adds support for sub-directories:

Feature	Scope	Comments
Generate All	Project	Instead of selecting direct files from the diagram folders, Generate All now includes all the folders and files recursively.
Locales Project property	Project	Used to update Project locales. Now includes all the folders and files recursively.
Command Line Code Generation	Project	Used to generate code for all the diagrams. Now includes all the folders and files recursively.
Command Line Upgrade	Project	Used to upgrade callflow, workflow, and IPD diagram files. Now includes all the folders and files recursively.
Command Line Publish	Project	Used to publish all IPD files. Now includes all the folders and files recursively.
Java Project Export	Project	Used to generate code while exporting WAR files. Now includes all the folders and files recursively.
Debugger	Diagram	Includes all the folders and files recursively for code generation during debugging.
Project Upgrade	Project	Used to upgrade callflow, workflow, and IPD diagram files. Now includes all the folders and files recursively.
SubModule handling	Diagram	Used to find the sub-module diagram when using auto-synchronization of parameters.
New callflow/workflow file creation	Project	Allows you to create new callflow/workflow files under the selected sub-directories (right-click and New > Other > Composer > Diagrams). Previously, it was created only under the callflows or workflows directory.

Folders Created When Upgrading Projects and Diagrams

The following additional folders may also be created in the Project Explorer:

- When upgrading to 8.1, a Project upgrade creates the folder `./WEB-INF/lib`, copies files from `./lib` to `./WEB-INF/lib`, then removes the `./lib` folder from the Java Composer Project.
- `archive` -- For placing zipped original contents of the Composer Project (created during an upgrade).
- `upgradeReports` -- For upgrade reports (created during an upgrade).

Adding Files to an Existing Project

Composer recommends adding files (i.e., a prompts audio file) to an existing Project within Composer using the following methods:

- Use the **File > Import** capability.
- Add directly from Windows Explorer and then refresh the resource list by pressing F5 in Composer's Project Explorer.
- Drag and drop files onto Composer's Project Explorer.

For out of sync files, please see troubleshooting topic [Workspace Files Not in Sync](#).

Project Permissions

Composer Project upgrade and code generation processes need current\launching user WRITE permission to the Composer Project Directories and Files. If you move Projects between Windows and OS X, these considerations may apply:

WRITE permission:

In Windows 7 OS, Projects created using Mac OS needs Effective Permission to be set. To do that:

1. Open Windows Explorer and browse to Composer Project directory.
2. Right Click the Project folder and select the **Properties** option to open the properties dialog.
3. Select the **Security** tab and click the **Advanced** button.
4. In the Advanced properties dialog, select the **Effective Permissions** tab.
5. In the Effective Permissions tab, select the current User / Groups to grant Full Permission.

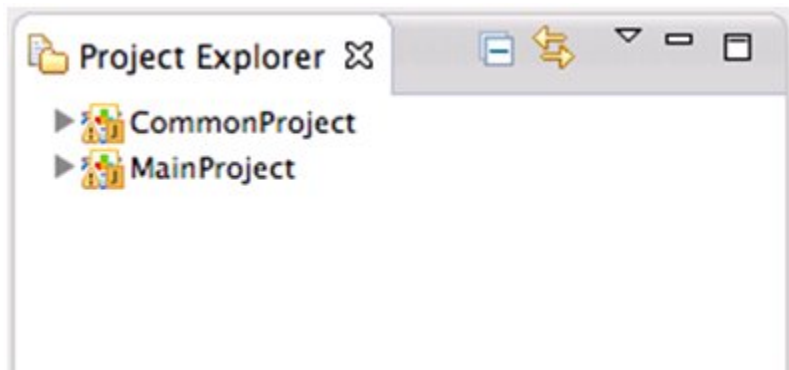
Also uncheck the **Read-only** and **Hidden** properties in the **General** tab for the Project and sub directories. Note: While importing Composer Projects, if the **Copy Projects into Workspace** option is selected, the above mentioned permissions needs to be set for the copied Project directory separately.

Using Composer Shared Subroutines

Typically subroutines are a part of the Project in which other diagrams call them. This makes the project self-contained that can be deployed as a unit with minimum dependencies on other Projects. However, in some cases subroutines may be used by multiple Projects but are required to be present in only one location in the workspace. This need for residing in a single Project within the workspace is usually governed by the need to deploy to all subroutines to a single location from where these subroutines may be referenced by multiple applications - similar to how a service is exposed. It is recommended that subroutines be a part of the Project they are consumed in and to enable this "sharing" via an SCM system (e.g., symbolic links in ClearCase; other system will support this capability differently). If that is not an option, subroutines in Composer can be placed into a "common" Project, so that multiple other Projects can access and reuse them. NOTE: In order to support the URL substitution from the "\$\$" tokens, this feature requires Orchestration Server version 8.1.300.27 and above.

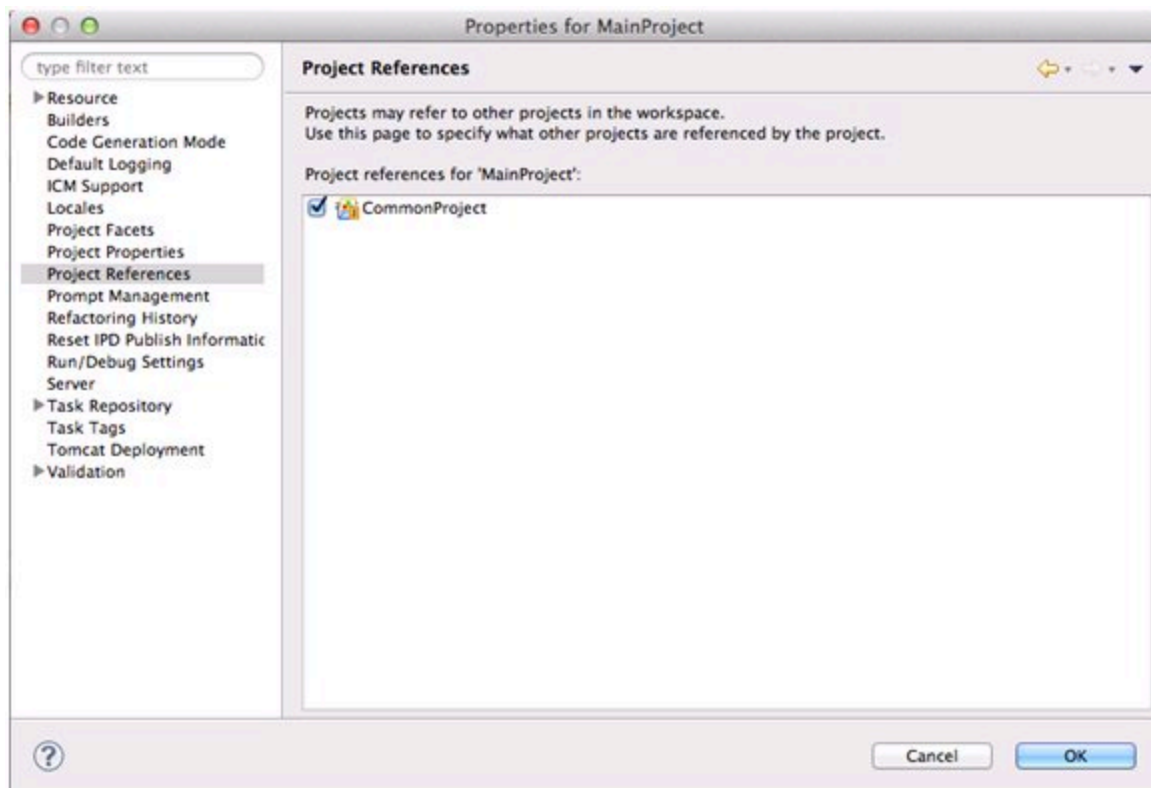
In our example, we will create two Projects:

- **CommonProject** - the Project containing subroutines
- **MainProject** - the Project containing the main diagrams, which will use the subroutines in CommonProject

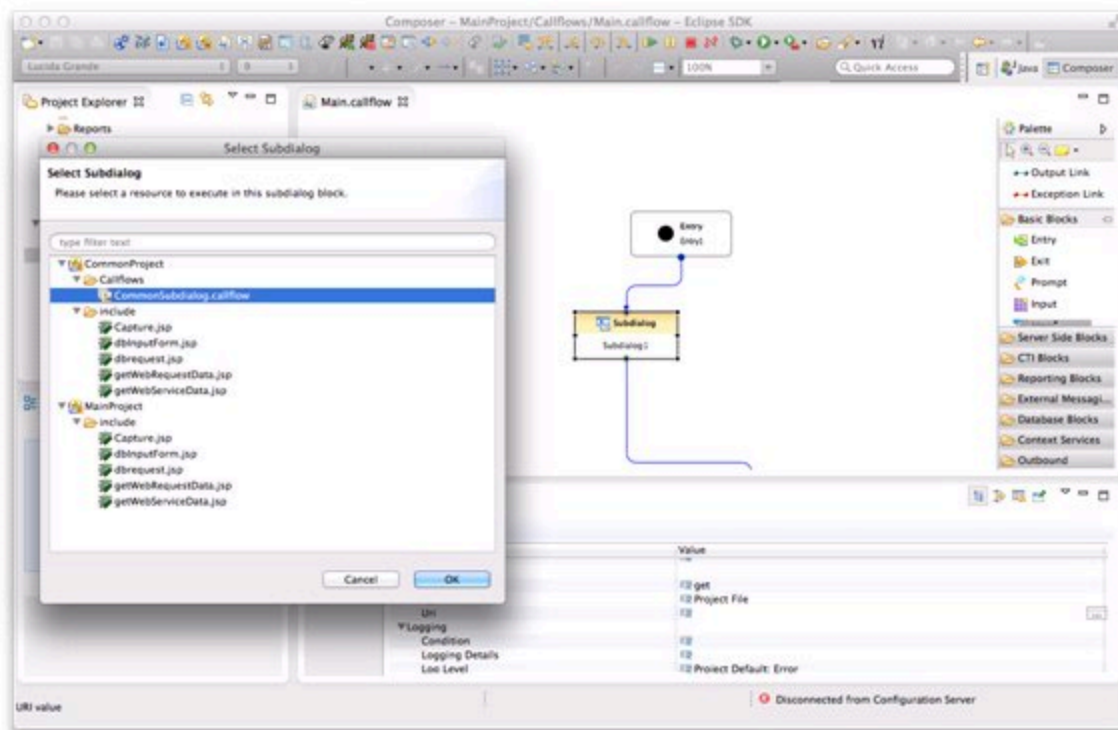


After subroutines have been created in CommonProject, MainProject must be set to reference CommonProject. This means that MainProject can use the subroutines files in CommonProject.

To do this, open the Project properties page of MainProject by right-clicking and selecting **Properties**. Select the **Project References** page on the left-hand side, and enable the checkbox for **CommonProject**:



In a callflow in MainProject, you can create a **Subdialog block** which uses a Subcallflow diagram in CommonProject:

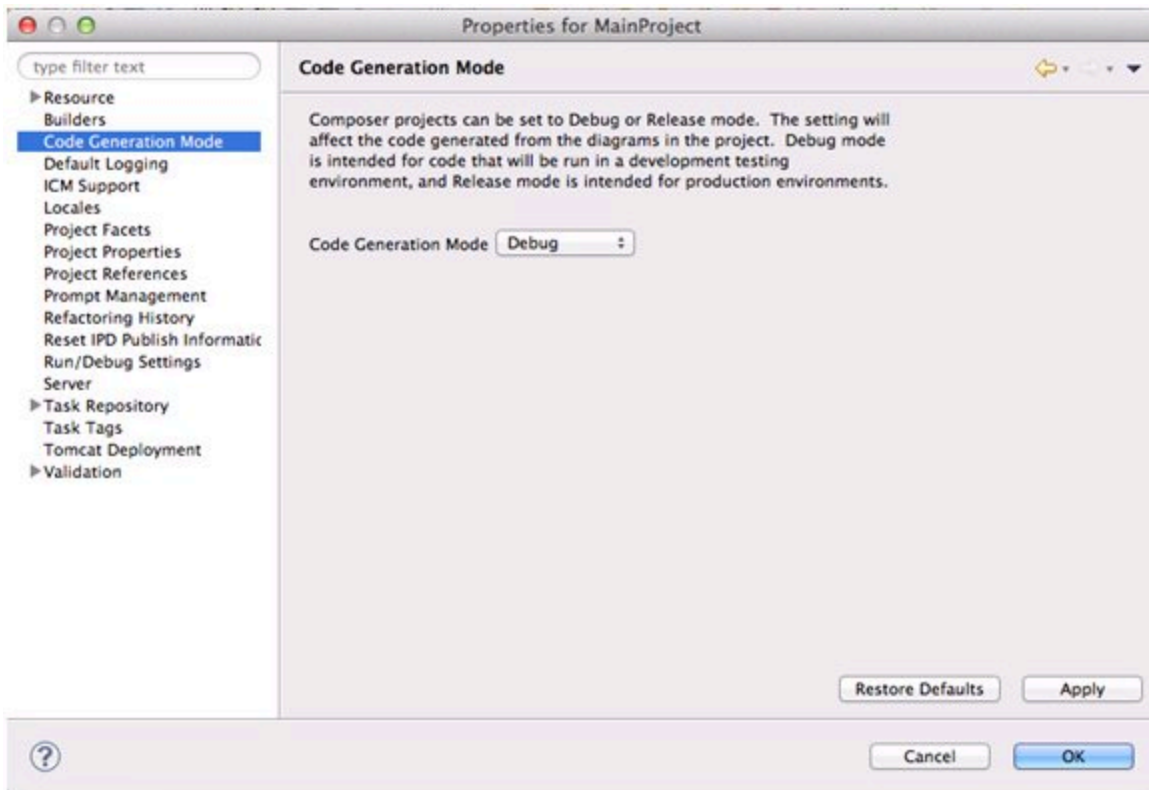


Debug and Release Modes

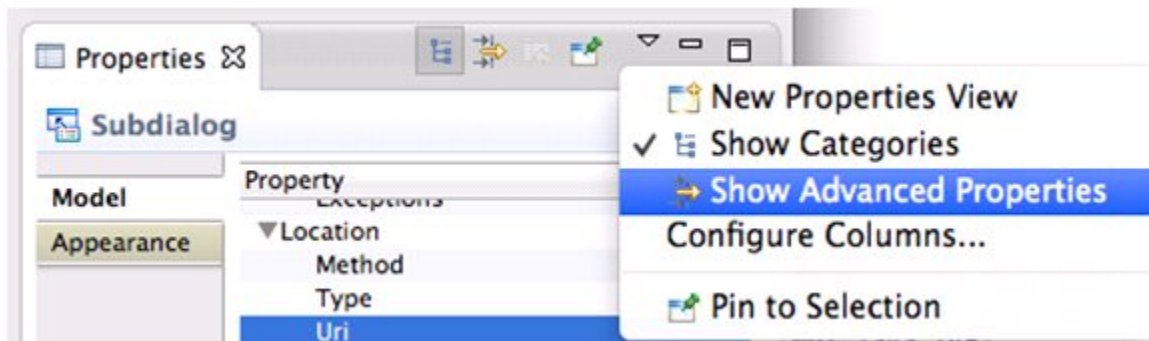
When using shared subroutines, it may be helpful to separate the development process from the final deployed application. During the development process, it is assumed that `CommonProject` resides in the same Workspace as `MainProject`. However, in a production environment, a more complex service may be needed to host subroutines.

Composer supports the concept of *Debug* and *Release* mode code generation. Using this mode flag, the same Project can generate different code suitable for specific tasks.

Debug and Release mode can be set by Project properties dialog:



To apply **Release Mode** to shared subroutines development, open the Properties view of the Subdialog block in the the callflow for MainProject. Enable the **Show Advanced Properties** option:



This will reveal a **Release Mode URI** property:

▼ Location	
Method	get
Release Mode Uri	http://\$\$SUBROUTINE_SERVICES\$/subroutines/\$\$SUBROUTINE_VERSION\$\$
Type	Project File
Uri	

Note that any token delimited by “\$\$” in this property can be substituted at runtime.

Once the Application is ready to deploy, set the **Code Generate Mode** of the Project to **Release**.

This will generate code that uses the **Release Mode URI** property value.

Project Properties

Selecting Properties from the Project menu opens a dialog box showing the **properties** of the selected Project or of the Project that contains the selected resource.