Composer Help

Composer Projects and Directories

5/13/2025

Composer Projects and Directories

Contents

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.

Voice Application Project Types

Voice applications use two types of Composer Projects:

- **Java Composer Projects** -- Use JSP and Java to implement server-side blocks and custom business logic. These Projects can be deployed to Tomcat, JBoss, or IBM WebSphere servers or other web applications servers that meet the requirements described in the Composer 8.1 Deployment Guide.
- **.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.

Also see Creating Voice Applications for GVP. **Note:** .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 `"include\getWebServiceData.aspx(482): warning CS0618: 'Microsoft.Web.Services3.SoapContext.Security' is obsolete: 'SoapContext.Security is obsolete.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.

Routing Application Project Types

Routing applications are created as **Java Composer Projects**. They are SCXML applications with full support for the Genesys Functional Modules. A Routing 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 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.

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.
- **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.

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.

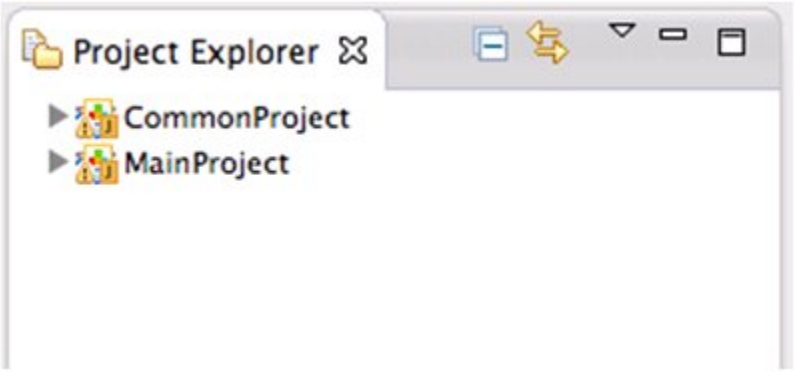
Using Composer Shared Subroutines

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.

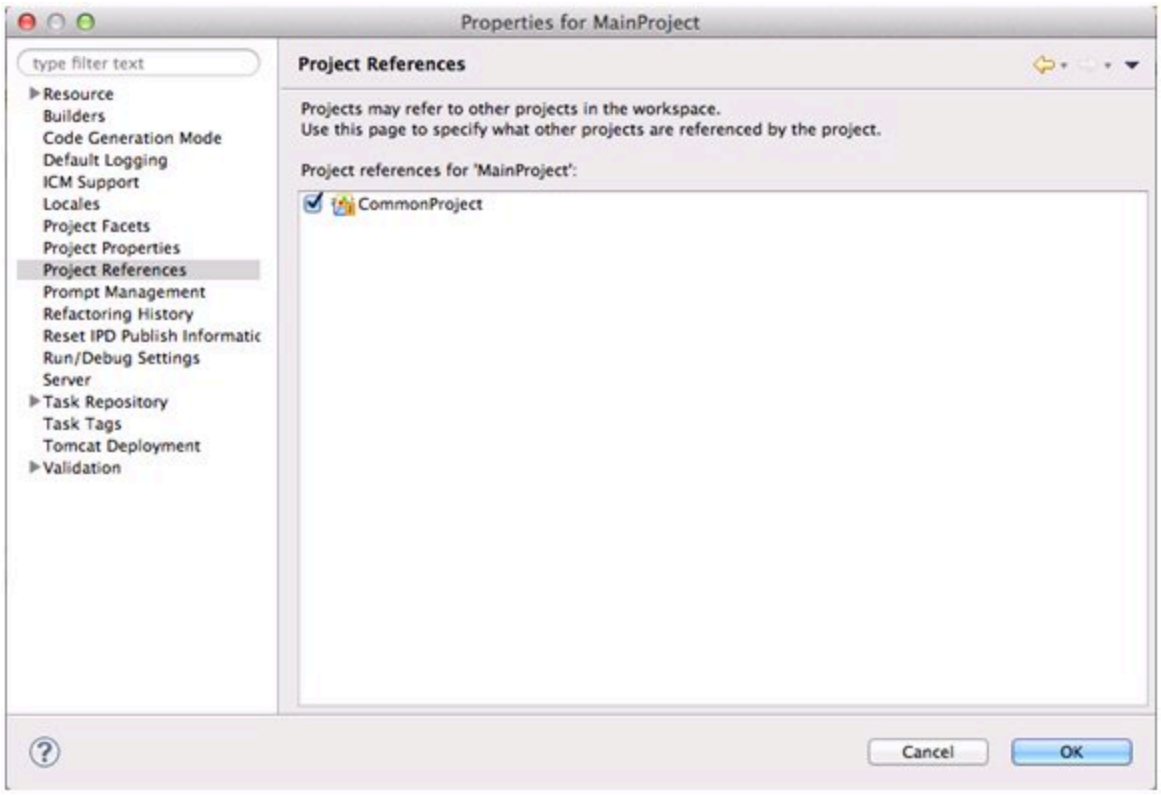
In our example, we will create two Projects:

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

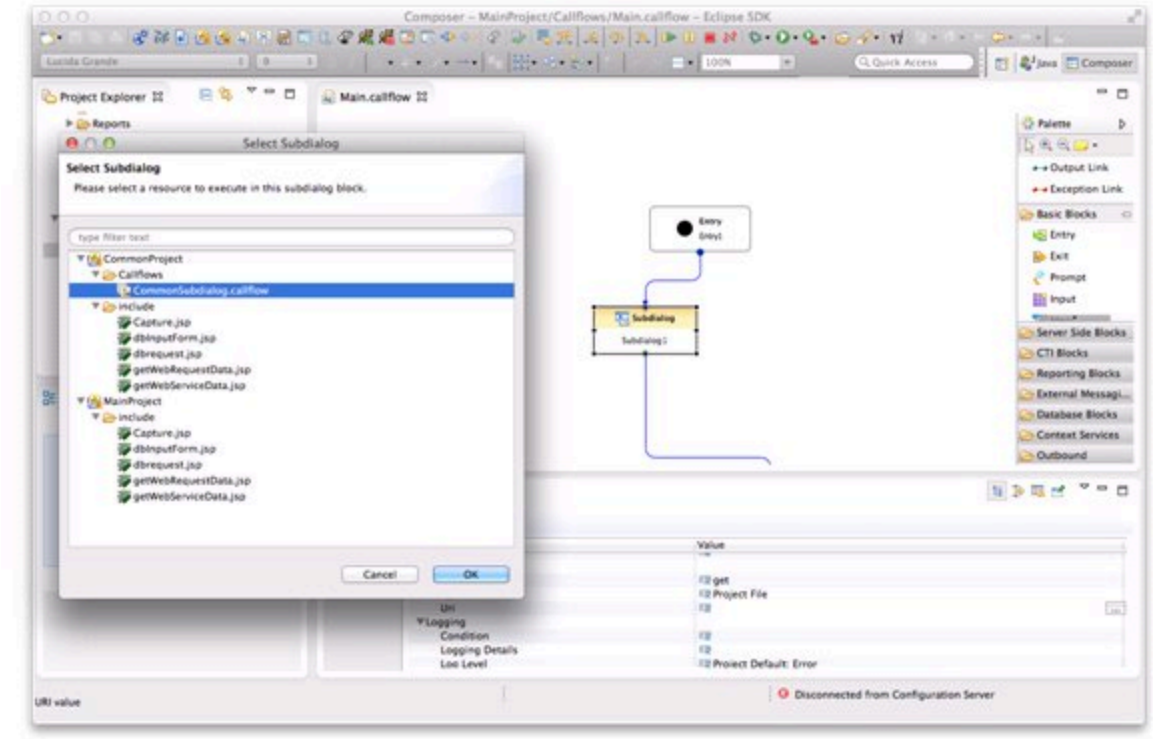


After subroutines have been created in `CommonProject`, `MainProject` must be set to reference `CommonProject`. This means that `MainProject` can use the subroutines files in `CommonProject`.

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`:

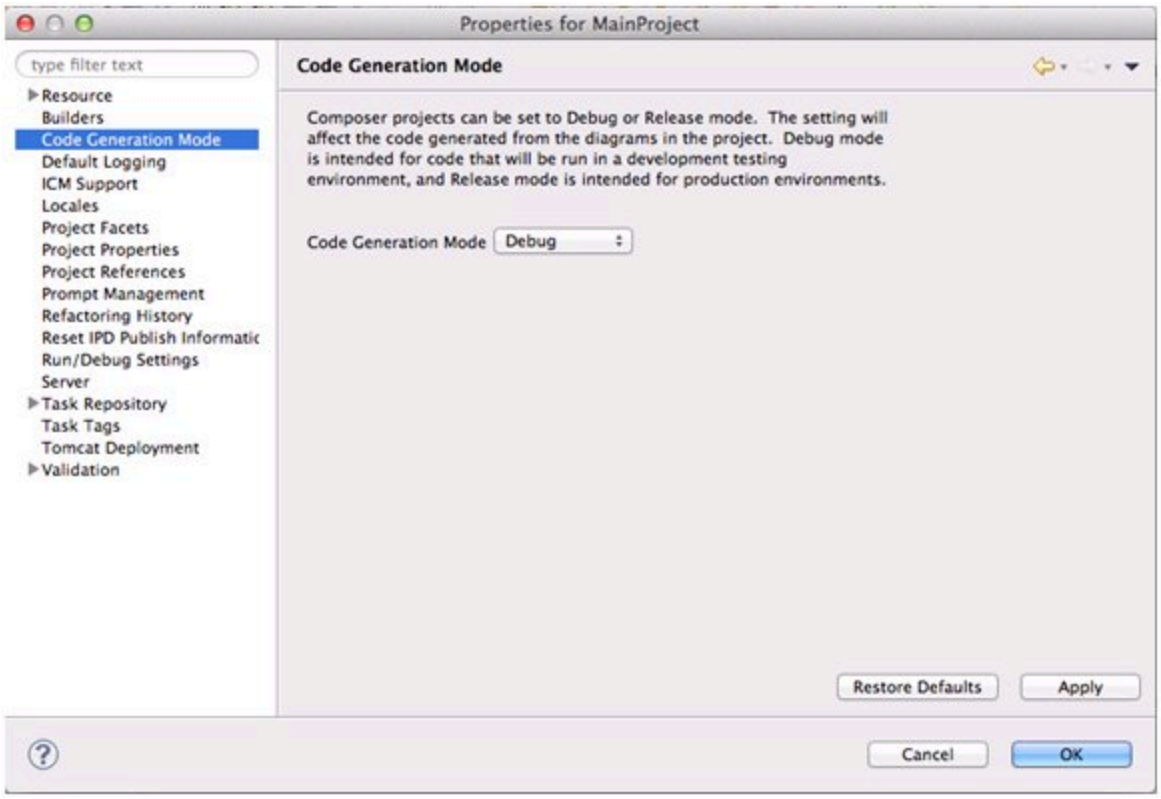


Debug and Release Modes

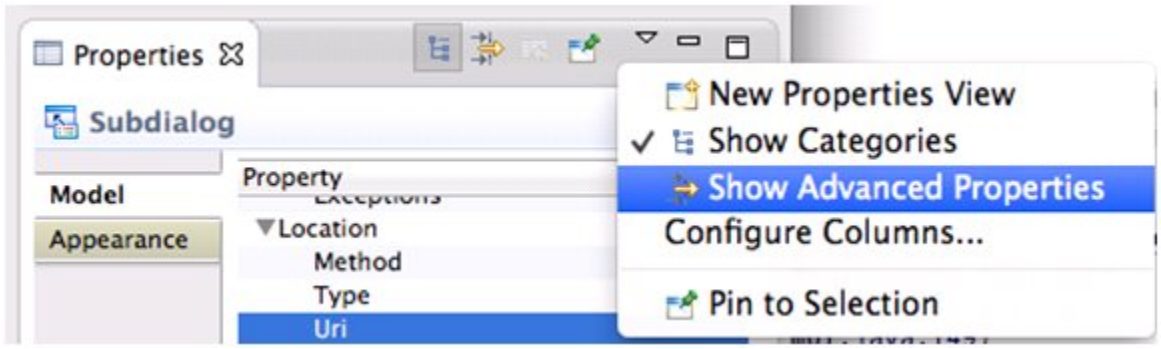
When using shared subroutines, it may be helpful to separate the development process from the final deployed application. During the development process, it is assumed that `CommonProject` resides in the same Workspace as `MainProject`. However, in a production environment, a more complex service may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

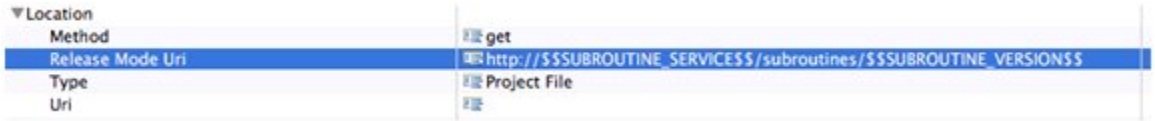
Debug and Release mode can be set by Project properties dialog:



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:



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.