Composer Help

Server-Side Common Blocks

12/13/2025

Contents

Server-Side Common Blocks

Both routing and voice applications use the Server-Side blocks.

- **Backend** (voice and route). Use to invoke custom backend Java Server Pages (JSP).
- **Business Rule** (voice and route). Use this block to have Composer query the Genesys Rules Authoring Tool (GRAT). For the Rule Package that you specify, Composer will query the GRAT for the Facts associated with the Rule Package. You can then set values for the Facts, call the Genesys Rules Engine for evaluation, and save the results in a variable.
- **DB Data** (voice and route). Use for connecting to a database and retrieving/manipulating information from/in a database. This block uses a connection profile to read database access information. It accepts a SQL query or a Stored Procedure call, which can be defined using the Query Builder or Stored Procedure Helper. It can also use a SQL script file.
- **DB Input** (voice only). Accepts a DB Data block as its data source and acts as an input field that accepts input based on a grammar created from the results returned from the database.
- **External Service** (route only). Enables routing applications to invoke methods on third party servers that comply with Genesys Interaction Server (GIS) protocol. Use to exchange data with third party (non-Genesys) servers that use the Genesys Interaction SDK or any other server or application that complies with the GIS communication protocol.
- **NDM Block**. Starting with Composer release 8.1.410.14, callflow diagrams add an NDM block to work with Nuance OSDM 6.1 modules used for speech and touch-tone IVR applications.
- **OPM Block** (voice and route). Enables VXML and SCXML applications to use Operational Parameters (OPM) which allow a business user to control the behavior of these applications externally. Operational Parameters are defined and managed in the Operational Parameter Management (OPM) feature of Genesys Administrator Extension (GAX)
- **URS Function** (route only). Introduced in 8.1.440.18. Use this block to call Universal Routing Server functions via `<session:fetch>` by the `urs` method.
- **Web Request** (voice and route). Use to invoke any supported HTTP web request or REST-style web Service. It supports PUT, DELETE, GET, and POST methods.
- **Web Service** (voice and route). Use to invoke Web Services for both routing and voice applications. Based on common Web Services standards such as XML, SOAP and WSDL instead of proprietary standards. You can pass parameters (as in subdialogs) and store the return values in variables. GET, POST, and SOAP are supported. **Note**: Currently Composer does not support SOAP 1.2. Only SOAP 1.1 is supported.

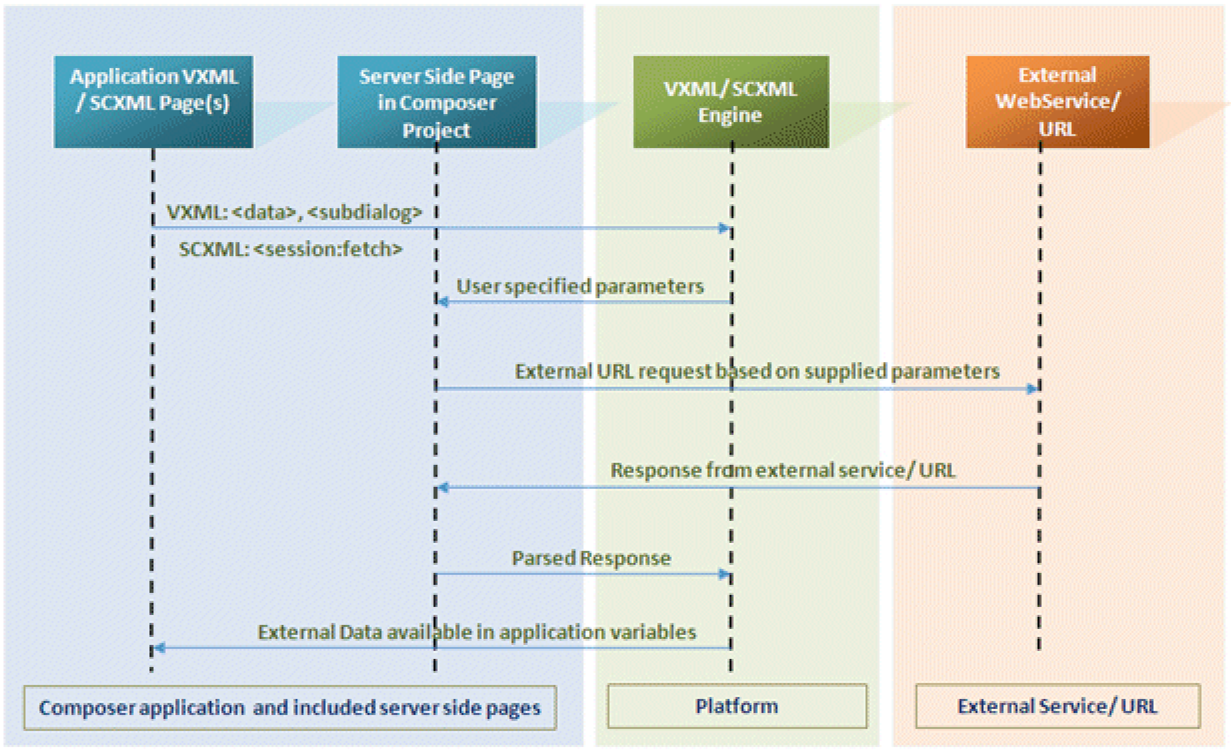
Server-Side blocks provide the ability to interact with internal and external custom server-side pages, Web Services, and URLs. These blocks can be used to exchange data like VoiceXML and SCXML variables, JSON strings between GVP interpreter, and custom server-side pages. With the exception of the Business Rule block, Composer uses server-side pages (ASP.NET or JSP) for implementing Server-Side block functionality. If you include these blocks in a diagram, server-side pages provided in Composer Projects are used at run time.

Important

Starting with version 8.1.450.33, Composer supports fetching HTTPS (HTTP over SSL) URLs in the Web Request and Web Service blocks. A new property category, **HTTPS Server Authentication**, with two properties, **Trust Store Location** and **Trust Store Password**, is introduced to extend support for HTTPS URLs.

Example Web Scenarios

In a typical scenario for the Web Service or Web Request block, the Composer-provided server-side page is invoked first via the platform through language appropriate tags (<session:fetch for SCXML and <data>, <subdialog> for VXML). This page, based on the input parameters specified in the block, invokes any external URL for the Web Service or Web Request blocks. In case of the Web Service block, it forms the appropriate SOAP request and sends it out. It then parses the response it receives from the external request and makes it available to the application. The figure below depicts the flow.


Application VXML / SCXML Page(s)
Server Side Page in Composer Project
VXML/SCXML Engine
External WebService/ URL
VXML: <data>, <subdialog>
SCXML: <session:fetch>
User specified parameters
External URL request based on supplied parameters
Response from external service/URL
Parsed Response
External Data available in application variables
Composer application and included server side pages
Platform
External Service/URL

The Need for Server-Side Pages

Composer provides the Server-Side blocks in anticipation that users will usually map either their callflows or workflows to their business logic via these blocks. For example, the Backend block offers the ability to create custom backend server pages that can be more tightly coupled with business logic and at the same time provides more flexibility since the backend logic is provided by the user. The different server-side functions offer a proxy service that can be used to query Web Services, web servers and backend server pages while providing a user interface that is simple enough to use, but also offering advanced features. Regarding security, the Web Request and Web Service blocks offer proxy clients which support HTTP, as well as SOAP. Composer supports Server-Side pages in both Java and .NET.

- Java server pages are hosted on Apache Tomcat, which is packaged and deployed with Composer.
- .NET applications are hosted on Microsoft IIS. The latter should be deployed by the user on the same server as Composer.

The choice between using Java or .NET is mainly dependent on what technologies are available to the user as well as the platforms. Below is a decision matrix outlining the some common situations where the most appropriate server-side block is recommended.

Situation	Recommended Block	Comments
A callflow/workflow needs to consume a Web Service which has a WSDL definition.	Web Service block	The Web Service block provides utilities to design the way the Web Service will be consumed, such as a WSDL parser. During runtime, the output results can also easily be assigned to callflow or workflow variables.
A callflow/workflow needs to query a web server for data	Web Request block	The Web Request block provides a proxy client for sending the web request, while offering functionality such as assigning the result to variables, and so on.
A REST-style web service needs to be consumed by the application.	Web Request block	The Web Request block is used to invoke any supported HTTP web request or REST-style web Service.
A callflow/workflow needs to access some data using some specific interface not using HTTP or SOAP	Backend block	The Backend block offers a proxy service to a backend application that is developed by the user and customized accordingly. The Backend block allows you to reuse custom JARs and .NET assemblies quickly since it provides an easy mechanism to pass parameters to and from the backend server page. The backend pages provide a skeleton implementation, which makes it easy and quick to start implementing custom logic which can use other user-provided libraries.
A callflow/workflow needs to do some customized post-processing to data retrieved	Backend block	The backend application will have to be created such that it retrieves the data and post-processes it accordingly.
My application does not work with either the Web Service or	Backend block	Try starting with the Backend block since the implementation is

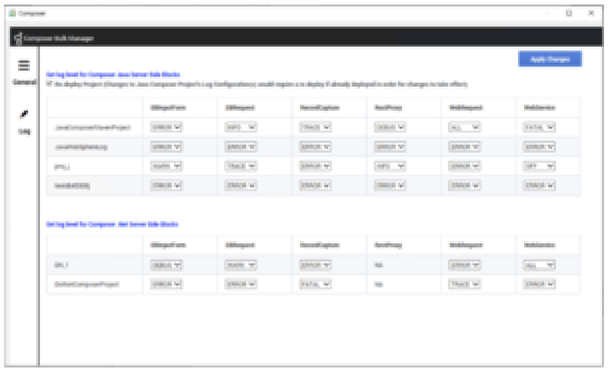
the Web Request blocks. What can I use?		open by nature. The Backend application is designed to provide a simple interface to the actual user-specific application. Note: The Backend server-side page called from the Backend block will be part of the project and will be included when the application project is deployed.

Configuring Log Levels for Server-Side Blocks

Beginning with release 8.1.500.03, you can configure the log levels for the various server-side blocks across projects using the Bulk Manager.

> **Important**
>
> For Java projects, information is recorded in the Composer.log file generated under `<tomcat-dir>\logs\`. For .NET projects, a log file is generated for each project under the **Logs** folder within each project.

- Click the **Log** tab on the Bulk Manager. The following is displayed:



- Select the required log level from the drop-down under each server-side block. On loading Bulk Manager, the existing log level for each block (based on the **log4j.xml** file for Java projects and **web.config** file for .NET projects) is selected by default. When you change the level here, the next time Bulk Manager is loaded, the updated levels are displayed based on the corresponding config files.
- Click **Apply Changes**.
- For the server-side blocks in a Java project, the Tomcat server must be restarted for the new log file settings to be effective. The Tomcat server is restarted automatically when you click **Apply Changes**.

Important

For .NET projects, changes to the log levels are effective immediately on clicking **Apply Changes**. Restarting IIS is not required for .NET projects.

- To disable logging for a particular server-side page, select the **Off** option from the drop-down.

Note: For both .NET and Java projects, log levels can be configured for each project individually. However, prior to release 8.1.550.08, for Java projects, the log levels configuration is applied to all the projects as a whole. Each Java project cannot have log levels configured individually in releases prior to 8.1.550.08.

The following are the different log levels available:

Log Level	Description
All	All levels inlcuding custom levels.
DEBUG	Informational events that are useful in debugging an application.
ERROR	Error events that might still allow the application to continue running.
FATAL	Very severe error events that might cause the application to abort.
INFO	Informational events that highlight the progress of the application at a high-level.
OFF	No information is recorded. Logging is turned off.
TRACE	Informational events that are a level higher than the *Debug* level.
WARN	Events that maybe potentially harmful.

Note: Changes to the log levels are recorded in the **log4j2.xml** file for Java projects (for each project under the location `project/WEB-INF/lib/`), and the **web.config** file for .NET projects. However, prior to release 8.1.55#.##, changes to the log levels are recorded in the **log4j.xml** file for Java projects.

Important

You must launch the Composer application using the **Run as Administrator** option to be able to configure log levels for server-side blocks.

Below is a video that shows you how to configure log levels for server-side blocks using the Bulk Manager. Link to video