Composer Help

Composer 8.1.3

12/29/2021

Table of Contents

Composer Overview

Use to Create Routing and Voice Applications

Composer is an Integrated Development Environment (IDE), based on Eclipse, for developing routing applications for the Genesys Orchestration Platform 8.x, which includes:

- Universal Routing Server (URS)—which enables intelligent distribution of voice and multimedia interactions throughout the enterprise.
- Orchestration Server (ORS)—an open standards-based platform with an SCXML engine, which enables the customer service process. ORS is responsible for executing orchestration logic (SCXML) that is provided by an application server (such as an application server hosting an SCXML-based routing application created in Composer). The responsibility of URS within the Orchestration Platform is to provide a necessary service to Orchestration Server to support Routing functions.

You can also use Composer to create voice applications for Genesys Voice Platform (GVP) 8.1+—a software suite, which unifies voice and web technologies to provide a complete solution for customer self-service or assisted service.

Notes:

- In the past, **Interaction Routing Designer** was used to create routing applications. Genesys Composer is now the tool of choice for creating both routing and voice self-service applications.
- Previously Composer was known as "Composer Voice," as it was used only to develop voice applications for Genesys Voice Platform. Starting with 8.0.2, the capabilities of the IDE were expanded to include support for Universal Routing application development. Due to this expansion in scope, the product name was shorted to "Composer." The terms Composer Voice and Composer Route are used in some places in the product, to refer to the collection of product features that are used specifically for Genesys Voice Platform application development, and Universal Routing Application development, respectively.
- Users may enable/disable Composer Voice and/or Composer Route capabilities through a Composer preference setting. This is useful for developers who are only using one of these Genesys platforms.

Application Development

Composer provides both drag-and-drop graphical development of voice applications (or “callflows”) and routing strategies (or “workflows”) as well as syntax-directed editing of these applications.

- For voice applications for the Genesys Voice Platform, Composer supports editing of VoiceXML 2.1, CCXML1.0 and SRGS 1.0.
- For routing applications for the Genesys Orchestration Platform, Composer supports editing of SCXML 1.0. Applications may be developed in an "offline” mode, without requiring the user to connect to

Genesys Configuration Server.

Application Debugging

Composer provides real-time debugging capabilities for both voice and routing applications.

- The Genesys Voice Platform Debugger is integrated with GVP for making test calls, viewing call traces, and debugging applications. It supports accessing SOAP and REST-based Web Services. Database access is possible using server-side logic and a Web Services interface.

- The Orchestration Server Debugger, integrated within the workflow editor, works with both live and simulated calls. For live calls, it places those calls into a T-Server/SIP Server connected to a URS/ORS system. The capabilities include setting breakpoints, stepping through a workflow, viewing and setting the values of variables, and viewing event messages from the URS/ORS platform.

Eclipse

Composer is an Eclipse-based application. The use of Eclipse as the underlying framework enables the use of third party IDE plug-ins, supporting integration with third party source code control systems, server-side development enhancements, and side-by-side development of any business logic required to support your applications.

Operating Systems

Composer can run on the following operating systems: Windows 2003 and 2008, Windows XP, Windows Vista, Windows 7, and Mac OS.

Composer Help Wiki URL

The URL to the Composer Help wiki is configurable by using the Online Wiki URL field: **Window** > **Preferences** > **Help**. The default works with English but if, for example, Japanese pages were available in a different location, then you could change the URL accordingly.

Third Party Software

This product includes software developed by:

- The Eclipse Foundation (http://www.eclipse.org).
- The Apache Software Foundation (http://www.apache.org/).

- The JDOM Project (http://www.jdom.org/).
- The Jaxen Project (http://www.jaxen.org/).
- The SAXPath Project (http://www.saxpath.org/)

This software contains code from the World Wide Web Consortium (W3C) for the Document Object Model API (DOM API) and SVG Document Type Definition (DTD).

The audio prompts used in Composer are provided by GMVoices (http://www.gmvoices.com).

Also see the Legal Notices section on the installation CD ReadMe file.

How to Use This Guide

- Depending on your location in the user interface, Composer's context sensitive help triggers wiki pages.
- You can use the TOC to locate help topics.
- As described in the above **watch** tab, you can also get notified when pages are updated (watch pages) or create a PDF.

Help for Help!

The majority of the Help for Composer exists in this online wiki. In addition, there remains a small help guide, available on the main Composer landing page, that explains how to use the help and its search capabilities. Introduction to Composer 8.1.3 Help.

The contents of both the help wiki and help guide is in English, regardless of the user's operating system locale, or any language packs they have installed on Composer. This does not affect the ability to use Composer in languages other than English, or to access the online Composer documentation in languages other than English. For more information, see Localization.

Block Palette Reference

For information on Composer blocks and block properties, you can go directly to the following:

- **Voice Block Palette Reference** (when building applications for Genesys Voice Platform (GVP))
- **Interaction Process Diagram Block Palette Reference** (used for routing applications)
- **Routing Block Palette Reference** (when building routing applications for the Orchestration Platform)
- **Common Blocks Block Palette Reference** (blocks used for both voice and routing applications)

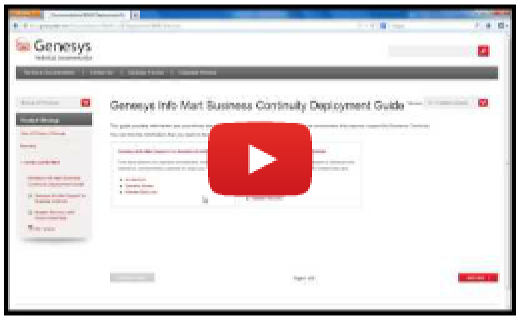
Composer Installation Video

Video Tutorial

Below is a video tutorial on installing Composer 8.1.3 on Windows in an Eclipse 4.2 environment. When Eclipse 3.7 is installed, there is a small extra step described in the Installation chapter of the *Composer 8.1.3 Deployment Guide*. Also see the Deploying Composer topic.

Notes:

1. This video is a high-level tutorial on Composer 8.1.3 installation. Be sure to consult the Composer 8.1.3 Deployment Guide *after watching this video.*
2. Silent installation of Composer is not supported.
3. Download the correct Eclipse for your computer’s processor, i.e., download 64-bit Eclipse download or 32-bit Eclipse based on the target computer.
4. Java Development Kit and Eclipse must match, i.e., both 32-bit or both 64-bit.



Deploying Composer

When deploying Composer, in addition to the Composer Installation video topic:

- Consult the *Composer 8.1.3 Deployment Guide* for Composer product installation information. This includes supported Eclipse versions to download, Java Development Kit variables that must be set, and other software requirements that must be in place prior to installing Composer.
- Consult the Deploying Composer Applications topic for information on deploying a Composer application to a web server.

Introduction to Composer

Composer is an Integrated Development Environment (IDE) based on Eclipse for building voice applications for the Genesys Voice Platform (GVP) 8.1+ and routing applications for the Orchestration Server 8.0+.

What is Composer?

Composer is the next generation version of Genesys Studio based on Eclipse 3.5.1. It provides a rich development experience, which Web Application developers are already used to, for building VoiceXML, CCXML, and SCXML applications. Familiarize yourself with basic Eclipse concepts by referring to the *Workbench User Guide* (**Help** > **Help Contents**). Composer provides ability to develop the following types of applications. For the GVP 8.x NGI Interpreter:

- Pure VoiceXML Applications with full support for Genesys extensions
- CCXML + VXML Applications requiring advanced call control features including Conferencing
- CTI + VXML Applications for Genesys Framework

For the Orchestration Server 8.x SCXML Engine/Interpreter:

- Pure SCXML Applications with full support for all Genesys predefined SCXML functional modules and extensions used for creating SCXML-based strategies and routing applications.
- Voice Routing SCXML applications for handling media of type voice for Inbound channels.
- Integrated CTI + VoiceXML applications for end-to-end treatment handling in conjunction with GVP and Stream Manager.

GUI Designer

Composer provides a drag-and-drop based interface for creating VXML and SCXML applications. Users can easily create flow diagrams by placing and connecting blocks and configuring properties. This approach also provides an easy mechanism for invoking Web Services and doing database lookups from the Application server-side. Custom blocks can be created that can be added to the supplied palette of blocks.

Code Editors

For those who prefer to write their own code, Composer provides a set of rich editors for SCXML, VXML, CCXML, and GRXML along with use case templates.

Templates

Out-of-the-box, reusable template applications are provided. These can act as a starting point for new projects and visual flows and serve as guidelines and tutorials for routing and voice application developers. Composer also provides templates for its rich editors with the ability to create user-defined custom code snippet templates, which can be exported and imported to share across team members.

Code Generation

When generating code, Composer provides the ability to generate Static VXML pages to take advantage of the Platform optimizations. For SCXML routing strategies, Composer provides the ability to generate Static SCXML pages for improved performance due to caching.

Debugging

Debugging functionality includes the ability to debug VoiceXML applications and callflow diagrams with the GVP Debugger and GVP Debugging perspective. The real-time GVP Debugger supports both Run and Debug modes. In the Run mode, call traces are provided and the application continues without any breakpoints. In the Debug mode, you can input breakpoints, single-step through the code, inspect variable and property values, and execute any ECMAScript from the query console. Composer provides real-time debugging capabilities for SCXML-based Orchestration Server (ORS) applications. The ORS Debugger is integrated within the workflow designer for making test calls, creating breakpoints, viewing call traces, stepping through an SCXML document/workflow, and debugging applications. Debugging can be started on an existing session or it can wait for the next session that runs the application at a given URL.

Deployment

Composer provides the ability to deploy applications. Future releases will provide the ability to deploy routing applications.

Project Management

Composer provides full project management capabilities for managing all the resources in a Composer project.

Routing Strategies

Composer is integrated with Genesys Configuration Server, which provides the ability to define and fetch strategy-related data residing in the Configuration Server database. This integration allows developers to find and select routing targets when using Composer's Target block. When creating routing strategies, developers can define List objects and routing-related Statistics.

Software Prerequisites

See the Installation chapter of the *Composer Deployment Guide* for specific requirements for Composer 8.1.2 and 8.1.3 (installed as an Eclipse plugin).

In general, to obtain the full functionality of Composer 8.1, the following Genesys products/software components are required:

- Orchestration Server 8.1.2 or later for debugging and testing SCXML session-based applications.
- Genesys Voice Platform (GVP) Media Control Platform 8.1.6 or later for debugging and testing VXML-based self-service applications.
- If you wish to use Context Services in routing workflows and voice callflows, you will need Universal Contact Server/Context Services 8.1.000.10 or later.
- Genesys Rules System 8.1.0 or later for business logic, which can be customized, and then invoked by Genesys applications.
- For information on components required by Orchestration Server, such as eServices components for processing multimedia interactions, see the Packaging section of the *Orchestration Server 8.1 Deployment Guide*.

Interface Overview

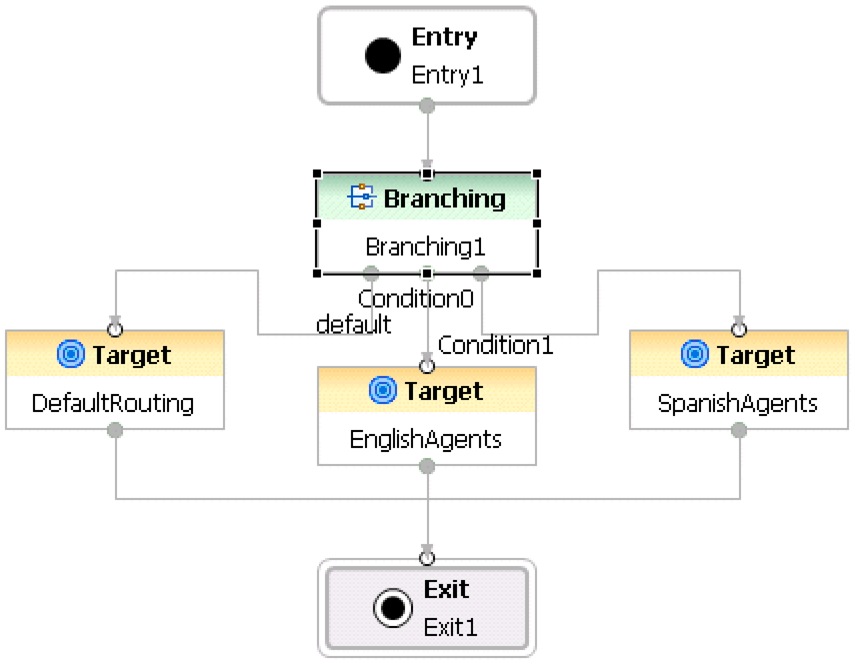
Note: The minimum screen resolution for Composer is 1024x768 on a standard 4:3 aspect ratio monitor. The recommended resolution is 1280x1024. Lesser resolutions, such as 800x600, are not supported.

Tutorials

For a tutorial on voice applications, see Your First Application, which describes a text-to-speech application. For a tutorial on routing applications, see Your First Application, which describes a DNIS routing application.

Blocks, Connectors, and Properties

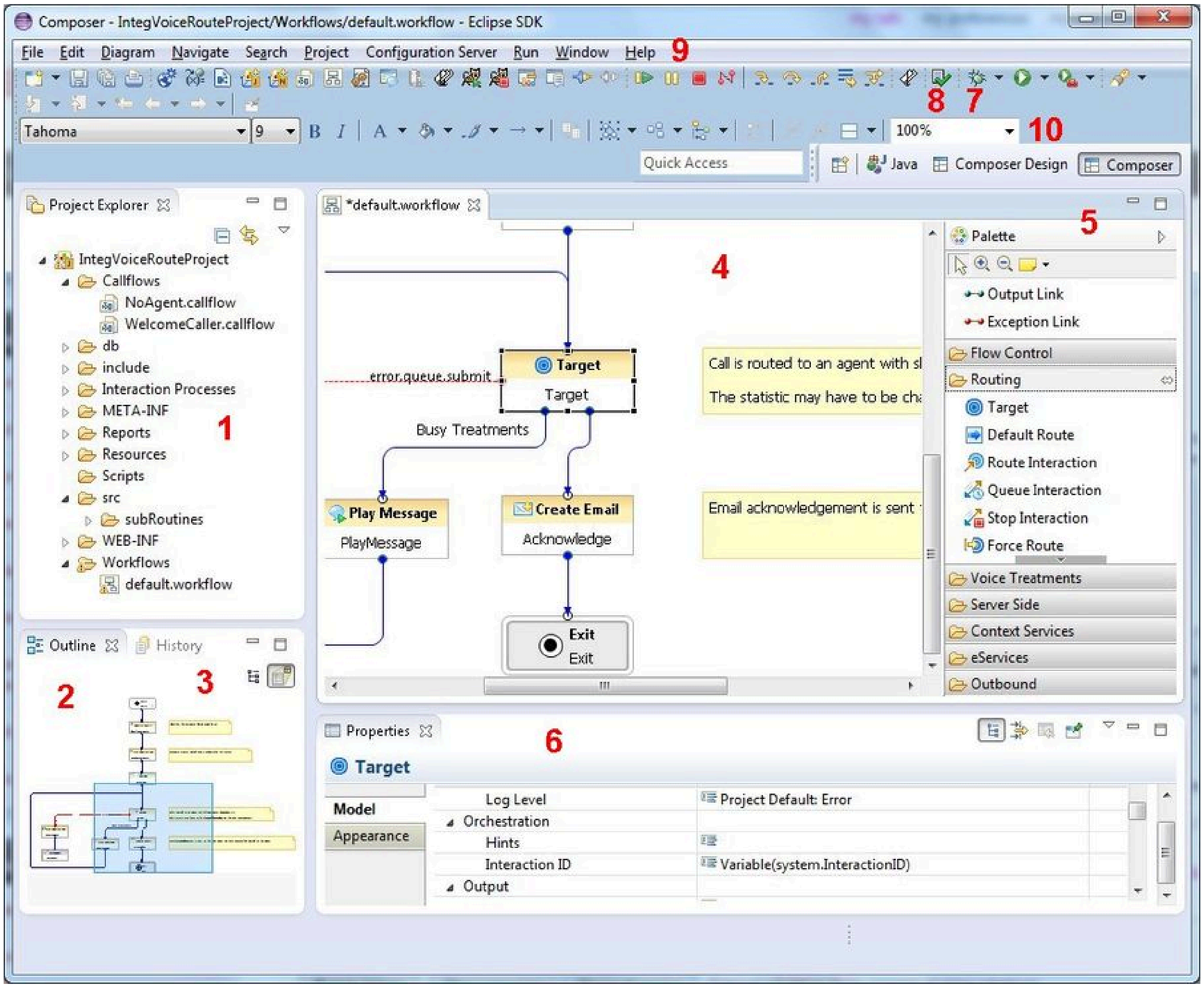
The Composer interface uses workflow and callflow design components (*blocks* and *connectors*) to create voice and routing applications.



It uses drag-and-drop to arrange, add, and delete blocks on a *canvas* area. The blocks are connected within the canvas (work area) to build the flow for the application. You define the properties for a selected block in Composer's *Properties view*. Also see: Working With Diagram Layouts

Interface Elements

The first time you enter the Composer perspective, since your workspace is empty and does not contain any Projects, you will see an empty Project Explorer on your top-left, and a blank center area. After you create a voice or routing Project, the Package Explorer shows all the files and resources that make up the Project. The figure below shows the GUI elements in **Composer perspective** for a sample routing application.



GUI Element Descriptions

The numbers in the figure above are keyed to the table below.

1	The **Package Explorer** shows all the files and resources that make up a Project. See Composer Projects and Directories for more information.
2	For large flows, the **Outline view** (shown above) allows you to navigate to the portion of the flow to view in the main canvas.
3	The **History view** maintains previous versions of flows and application files, allowing you to revert to any previous version if needed.
4	The **Canvas** is where you create flows by placing and connecting blocks. Composer's canvas area is the overall workplace that you will use for building your applications.
5	Depending on your Perspective, the **Palette** contains workflow diagram-building blocks or callflow diagram-building blocks grouped in various categories.
6	The **Properties view** shows block properties and allows you to modify settings, set variables, and otherwise change or set the properties corresponding to a block. This area also displays Call Traces during debugging, or Problems during validation or testing.
7,8	In the top toolbar, the **Validate** button allows you to check for syntax errors. The **Generate Code** button creates VXML and SCXML pages from the diagrams you create.
9	**Menus** and **Toolbars** provide commands and operations for running Composer.
10	**Perspective** buttons show the active perspective and let you easily move between perspectives. By default, when you enter the workbench for the first time, you will be taken inside the Composer perspective. Perspectives are arrangements of different sections of the GUI in a manner that facilitates easy use of a particular feature. For example, the GVP and ORS Debugging perspectives will show those sections (Breakpoints, Call Trace, Variables, and so on) that are useful when debugging an application.

Composer displays a Help view on the right if you select **Help** > **Search** or **Help** > **Dynamic Help**.

Perspectives

Composer includes the following perspectives for building applications:

- GVP Debugger, for debugging applications you build or import
- ORS Debugger, for debugging routing applications you build or import

- Prompts Manager perspective, which provides the ability to quickly review all prompts in a Composer Project
- Composer perspective shows the Project Explorer, Outline view, Canvas, Palette, and can show the following tabs in the lower pane: Properties, Prompts Manager, Problems, Console, and Call Trace.
- Composer Design perspective can be used to simplify the workbench to show only the palette of blocks, the canvas area, and the Properties tab.

When you click the open perspective button and select **Other**, the window lists all perspectives available in Eclipse. You can use this window to change perspectives or select **Window** > **Open Perspective**. Any customized perspective appears in this list. You can configure perspectives on the **Window** > **Preferences** > **General** > **Perspectives** preference page.

Customizing the Show View Menu

A view can be displayed by selecting it from the **Window** > **Show View** menu. You can customize this menu by using **Window** > **Customize Perspective**. Click the **Submenus** down arrow and select **Show View**.

Using the Designer

Blocks

A block is the basic building unit that you use to create applications. In Composer perspective, the palette of blocks is located in the right-most part of the main window (unless the Help window is also visible) and contains various categories of blocks. Every application must start with an voice Entry block or routing Entry block. You can also create Custom Blocks. A routing applications starts with Interaction Process Diagram Blocks.

Using Blocks

When creating voice application callflows and routing application workflows using the designer:

- You double-click or drag-and-drop callflow blocks and/or workflow blocks to place them onto the center area (canvas).
- You configure properties for each block.
- You connect the blocks together by drawing connection links to define the flow.

For additional information, see the the Eclipse *Workbench User Guide* (**Help** > **Help Contents**).

Block Names and Multi-Byte Characters

Composer block names can contain only alphanumeric characters. If multiple-byte characters are used in block names, the code generation step fails and no SCXML or VXML file is generated from the Composer diagram.

Methods for Adding Blocks

There are a few ways to add blocks from the Palette to the canvas. The most common methods are as follows:

- Click on the block icon on the palette, release the mouse and click on the target location on the canvas area.
- Double-click a block icon on the palette.
- Click on the block icon on the palette, and while holding down the mouse button, drag and drop the block to the canvas.

Any of these methods will add the new block and you can then type the name of the block on the canvas itself. Click here to read about block naming restrictions.

Outline View

For large call or workflows, the Outline view allows you to navigate to a portion of the flow diagram to view in the main canvas. It can also be used to facilitate navigation for other types of elements that might appear in the canvas or editor window, such as a large VXML file displayed in the VXML Editor. For more information, see the Outline View topic in the Eclipse *Workbench User Guide* (**Help** > **Help Contents**).

Simulation View

Simulation view shows the VoiceXML or SCXML code (read only) for a selected block (IPD blocks do not have this view). To add the Simulation view to the current perspective:

- Click **Window** > **Show View** > **Other** > **Composer** > **Simulation**.

Block Context Menus

Or, you can use a block's context menu as follows:

- Select a block in the canvas, then right-click the box and select Simulate Code from the context menu as shown in the figure below. The Simulation View displays the code for the selected block.

The History view maintains previous versions of call and workflows and application files, allowing you to revert to any previous version if needed.

- For more information, see the Local History topic in the Eclipse *Workbench User Guide* (**Help** > **Help Contents**).

The Problems view is used during validation of callflows, workflows, and files (VXML, SCXML, GRXML, and so on). It displays information about errors encountered when validating an application.

- For more information, see the Problems View topic in the Eclipse *Workbench User Guide* (**Help** > **Help Contents**).

Connection Links

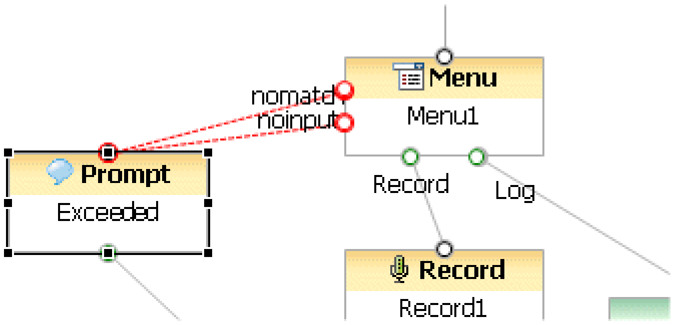
Blocks are connected to each other using connection links. There are two types of connection links:

1. Use *OutLinks* to connect one block's output port to another block's input port: OutLink
2. Use *Exception Links* to indicate error or exception conditions by connecting from a block's exception port to another block's input port:
   ExceptionLink

Find the connection links at the top of the palette on the right side of the Composer window.

Example

The figure below shows examples of using the link tools:



In the above example, the red links (going from the Menu block to the Prompt block) result from using the Exception Link tool to connect the two blocks. The black links (going from the Menu block to the Record block and the Log link) result from using the Outlink tool.

Adding a Link

To add a new Output Link (or Exception Link):

1. Click the Output Link (or Exception Link) icon in the palette.
2. Move the mouse over to the source block. The cursor will change to an upward arrow.
3. Click once on the source block and keep the mouse button pressed. Then drag the mouse onto the target block and release the mouse button.

This will add the connection link between the two blocks. To use an Exception Link, the source block must have an exception port defined. This is done by selecting at least one supported exception within the block's Exceptions property. Another method for adding an Output Link or Exception Link between two blocks is as follows:

1. Click once on the source block to select it.
2. Hold the Ctrl key and click once on the target block to select it as well.
3. Double-click the Output Link (or Exception Link) icon in the palette to create a connection between the two blocks.

Again, to use an Exception Link, the source block must have an exception port defined.

Changing Style and Appearance

Composer allows you to change the style and appearance of your connection links to suit your needs. To change connection link appearance:

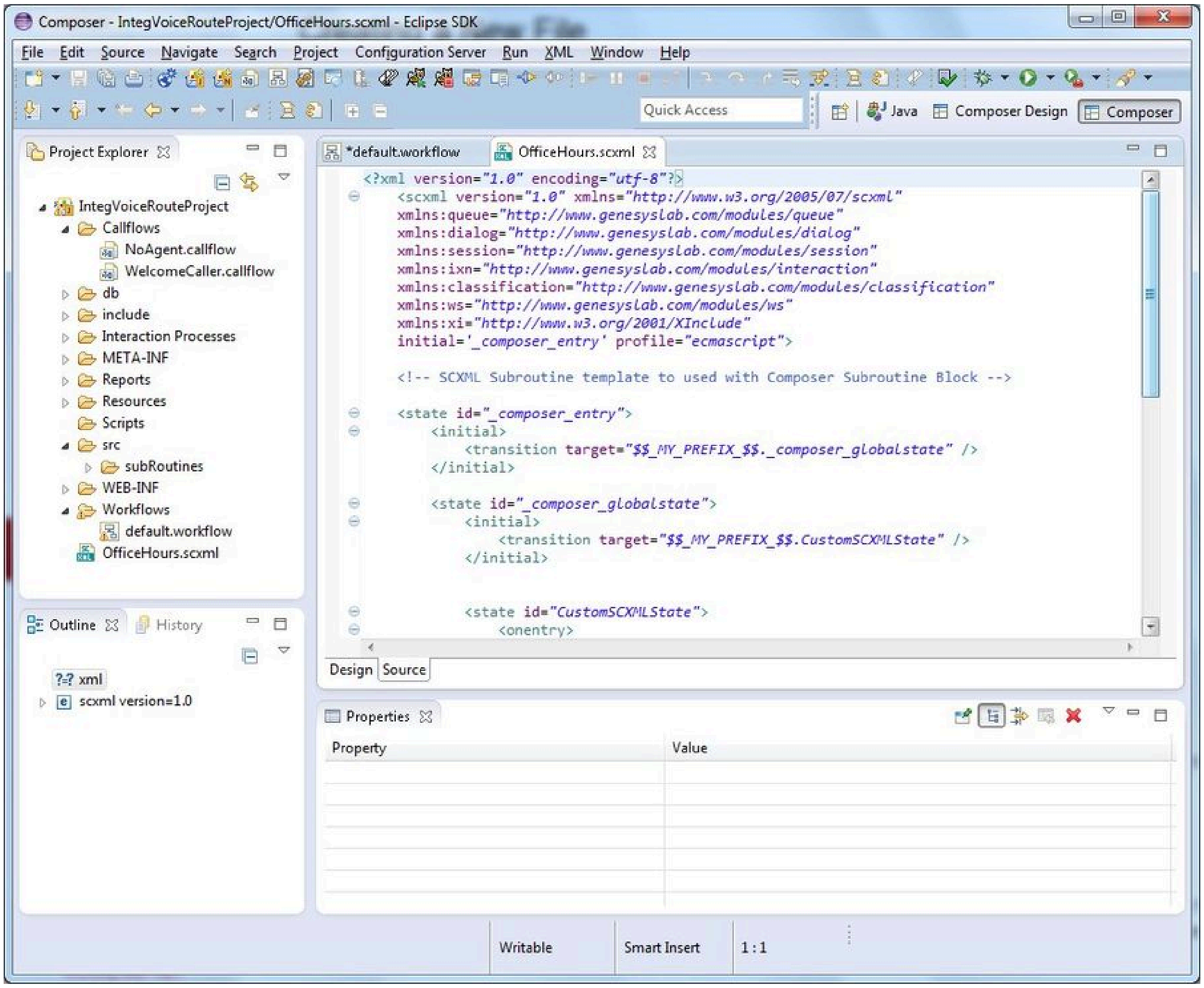
1. Select the connection link(s) you wish to change on the diagram (Ctrl-click to select multiple links). If the Properties view for the selected link is not visible, select **Window** > **Show View** > **Properties**. Or right-click the link and select **Show Properties View** from the shortcut menu.
2. In the Properties view, click the **Appearance** button to the left. A palette of appearance options is displayed in the Properties view.

You may change any of the following:

- Font, Font Size, Font Style, and Color
- Line and Arrow Style
- Smoothness of the connection line (None, Normal, Less, or More)
- Routing Style (Oblique, Rectilinear [default], or Tree), as well as the option to avoid obstructions or choose the closest distance for the link (even if it must cross over another block)
- Jump links set how links will be displayed when a link needs to jump over or cross another link (None, All, Below, Above), and the shape of the link crossing over can be Semi-Circle, Square, or Chamfered
- Reverse jump links switches the orientation of the semi-circle, square, or chamfered shape of a crossing link

Composer Code Editors

Composer provides the ability to hand-code SCXML, VoiceXML, CCXML, GRXML, JSP, and ASP.NET for custom scripts as a part of the application development process. The editors have standard editing capabilities and time-saving features such as a code snippet library, validation checks for errors (during design and save time), and syntax highlighting.



```
<?xml version="1.0" encoding="utf-8"?>
    <scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
    xmlns:queue="http://www.genesyslab.com/modules/queue"
    xmlns:dialog="http://www.genesyslab.com/modules/dialog"
    xmlns:session="http://www.genesyslab.com/modules/session"
    xmlns:ixn="http://www.genesyslab.com/modules/interaction"
    xmlns:classification="http://www.genesyslab.com/modules/classification"
    xmlns:ws="http://www.genesyslab.com/modules/ws"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    initial='_composer_entry' profile="ecmascript">

    <!-- SCXML Subroutine template to used with Composer Subroutine Block -->

    <state id="_composer_entry">
        <initial>
            <transition target="$$_MY_PREFIX_$$._composer_globalstate" />
        </initial>

        <state id="_composer_globalstate">
            <initial>
                <transition target="$$_MY_PREFIX_$$.CustomSCXMLState" />
            </initial>


            <state id="CustomSCXMLState">
                <onentry>
```

Code Editing Features

The editors provide:

- Standard editing features such as cut, copy, paste, undo, show line numbers, search and replace, bookmarks and TODO markers
- Standard Eclipse Editor features; local file history support, Team support for source code control, compare files.
- The ability to do background validation of the text as the user types, showing squiggly marks for errors as is done in Microsoft Word.
- A Validate option to validate against the schemas.
- A code snippet library with the ability for developers to add their own custom scripts.
- An outline view for quick navigation and the ability to view and edit the XML in tree format.
- Syntax coloring with the ability to customize the colors.
- A spell checking feature; a yellow squiggly line is shown below words that are misspelled.
- Quick-fix choices to fix the spelling or ignore / disable the check.
- Task tags features for setting preferences to auto scan comments with TODO in comments, and automatically add tasks corresponding to these comments.
- Context-sensitive help as the user types in the code

Using the Editors

Composer editors are embedded/integrated within the user interface and are made available to you whenever a .scxml, .vxml, .ccxml, .grxml, or .jsp file is created or accessed within Composer.

For additional information, see Accessing the Editors and Templates.

Eclipse Workbench

- As described in the *Composer Deployment Guide*, Composer 8.1.3 is installed as an Eclipse plugin.

Since Composer is based on based on Eclipse, you should familiarize yourself with basic Eclipse concepts by referring to the *Workbench User Guide*. The *Workbench User Guide* presents an overview of many of the same concepts used within Composer, but from the Eclipse development environment framework perspective. Reviewing this information is valuable as a first step in getting familiar with the Eclipse user interface on which Composer is based. To access the *Workbench User Guide* online help:

1. Select **Help** > **Help Contents**. The system displays the Help - Composer window.
2. In the Contents navigation pane, click **Workbench User Guide**.
3. Click **Concepts** in the main Help pane.

Review the multiple sections of the Concepts section to gain familiarity with Eclipse.

- Select Search from the main Help menu to search the Eclipse Help.

Enabling/Disabling Functionality

You may hide voice application (GVP) and/or routing application development capabilities through a Composer preference setting. This feature is useful for developers who are only developing applications for GVP or Universal Routing. To hide voice or routing development capabilities:

1. Select **Window** > **Preferences**.
2. Expand **General** and select **Capabilities**.
3. Click the **Advanced** button.
4. In the Advanced Capabilities dialog box, expand **Composer**.
5. Check or uncheck **Composer Route** or **Composer Voice** based on your need.

If you uncheck Composer Voice, the ability to create Projects and diagrams with callflows will no longer be available. Also, perspectives and views exclusive to callflows will not be available. This means you temporarily won't be able to design voice applications for GVP until you re-enable Composer Voice capability. If you uncheck Composer Route, the ability to create Projects and diagrams with workflows is not available. Also, perspectives and views exclusive to workflows are not available. This means you temporarily won't be able to design routing applications for Universal Routing 8.x until you re-enable Composer Route capability. <ol start="6"> <li>Click **OK** in both dialog boxes.</li> </ol>

Hiding File Types

You can hide Composer file types by using base Eclipse functionality in Composer. This may be desired when certain functionality is not applicable to your environment. For example, when using Voice capabilities, and VXML is used but not CCXML, you may wish for the CallControlXML file type to be hidden from the File > New menu. The following steps may be used:

1. Right-click the one of the buttons for the Composer-provided perspectives (e.g. Composer Design, Composer) and click **Customize**....The Customize Perspective dialog appears.
2. Click the **Shortcuts** tab.
3. To remove **CallControlXML File** from the **File** > **New** menu, set **Submenus** to **New**.
4. Expand **Composer** and check **Others**.
5. In the list of shortcuts, uncheck <**file-type**>, where <file-type> is the type to be hidden.
6. Click **OK**.
7. Repeat for other perspectives if desired.

This customization is specific to the workspace. If you use other workspaces, you must customize them as well. This is base Eclipse behavior where customization is saved within the workspace.

Localization

Localization in Composer allows you to use Composer in your preferred language. There are some limitations to localization, which are detailed here.

Bundled Help Always in English

The majority of the documentation for Composer exists in this online wiki. In addition, there remains a small help guide that is bundled with Composer, available in the product by clicking on **Help** > **Help Contents** > **Composer**.

The contents of this bundled help guide are always in English, regardless of the user's operating system locale, or any language packs they have installed on Composer. This does not affect the ability to use Composer in languages other than English, or to access the online Composer documentation in languages other than English.

Translating the Palette

After installing the Composer Language Pack, if you open Composer and continue to use an existing workspace, you will discover that the Palette is not localized. This is because Palette customization happens within Composer. You can change the labels, descriptions, and visibility of individual tools in the Palette. Composer preserves your customizations and, for that reason, Composer keeps the settings of your workspace Palette even after a Language Pack is installed. To workaround this issue, you can perform one of the following actions.

1. Start a new workspace. The Palette will be localized in the new workspace.
2. If you are willing to lose your current Palette settings:
    - Shut down Composer.
    - In a file explorer, go to <workspace>\.metadata\.plugins\com.genesyslab.composer.diagram, where <workspace> is your Composer workspace.
    - Delete the file `paletteSettings.xml`.
    - Restart Composer. The Palette will be localized in your existing workspace.

Translating Diagram Properties

The diagram preferences shown below use the Eclipse GMF runtime preferences, including their message strings. Due to limitations in Eclipse related to translating GMF Runtime, when localizing, certain diagram preferences are not translated.

- Window > Preferences > Composer > Composer Diagram > **Appearance**
- Window > Preferences > Composer > Composer Diagram > **Connections**
- Window > Preferences > Composer > Composer Diagram > **Pathmaps**
- Window > Preferences > Composer > Composer Diagram > **Printing**
- Window > Preferences > Composer > Composer Diagram > **Rulers and Grid**

Translating Enumeration Properties

In the Properties view, many properties of blocks are enumerations, which are collection of fixed values, from which one is selected. Internally within Composer, a number of different dialogs, drop-down boxes, and various other mechanisms are used to display the various enumeration properties. The 8.1.3 release of Composer does not to enforce uniform behavior when translating enumeration properties. As a result, some enumeration properties will appear localized, while others may not. This has no impact on diagram validation, code generation, or runtime behaviour. It is merely the values displayed in the Properties view that appear either translated or untranslated. The underlying code that is generated will always be the same.

East Asian Characters

If you are using characters from an East Asian language (for example, Chinese, Japanese, Korean) in Expression Builder, you may find that they do not display properly, and appear as squares rather than the expected characters. The most likely cause of this issue is that the operating system font does not support East Asian language characters. As a workaround, change the Dialog Font setting from within Composer.

1. In Composer, go to **Window** > **Preferences**.
2. Select **General** > **Appearance** > **Colors and Fonts**.
3. In the preference dialog, select **Basic** > **Dialog Font**.
4. Click **Edit** to bring up the font selection dialog.
5. Choose a font which can render East Asian languages, such as Arial Unicode MS.
6. Click **OK** on the font selection dialog, then click **OK** on the preferences dialog.

Connection Profile and Non-ASCII Characters

Database Connection Profiles do not support non-ASCII characters. Use only ASCII characters when creating Connection Profiles.

Composer Versus IRD

Integrated Development Environment

- Composer is a single Integrated Development Environment (IDE) for creating applications to orchestrate the entire customer experience. Composer-created voice and routing applications can command and control the customer experience through all channels (IVR, voice, e-services, and so on).

- Composer's open framework enables widely-available, existing competencies to be used to create reusable components that manage the customer experience. The IDE allows both customers and integrators to utilize existing code sets (HTML, VXML, java, perl, REST and others) to control the customer experience.

- The open framework also allows simplified integration into all Enterprise applications to harness the information within the Enterprise to drive and personalize the customer experience.

Differences from IRD

In the past, Interaction Routing Designer was the only Genesys tool to create routing applications. Genesys Composer is now the tool of choice for creating both routing and voice self-service applications.

A few of the differences between Composer and Interaction Routing Designer are listed below.

- Composer is integrated with Orchestration Server allowing you to manage customer conversations spread out over time using the ORS session-based functionality and persistent storage as well as Orchestration Extensions.
- Composer encompasses IRD's functionality and much more routing functionality in general.
- Composer lets you create routing applications using an open language (SCXML) and ECMAScript for decision-making. In contrast, IRD uses a closed Genesys proprietary language (IRL) and you are limited to IRD's objects and functions.
- Composer gives the option of writing your own SCXML code and/or using predefined blocks.
- Unlike IRD, you can also use Composer to create voice self-service applications for Genesys Voice Platform, including VoiceXML and CCXML-based applications. You can also create integrated voice and routing applications.

Composer Routing Application Types

You can use Composer's predefined blocks and/or write your own code to create routing applications that route based on various criteria such as:

-
    - **Agent, Agent Group, ACD Queue, Place, Place Group, Route Point, Skill, or Variable**
    - **last called agent**
    - **date and time**
    - **the value of a statistic**,
    - **dialed number (DNIS)**
    - **originating number (ANI)**
    - **percent and conditional routing**

The above list is by no means complete. It represents only a few types of routing applications that can be created in Composer. Since Composer uses open languages (SCXML and ECMAScript), you are not limited to its pre-defined blocks, but are free to create many types of routing applications.

Migrating IRD Strategies into Composer

Note: You can migrate routing strategies created in IRD into Composer. For more information, see the IRD to Composer Migration Guide.

Composer Blocks Mapped to IRD Objects

Composer refers to the fundamental element of a workflow as a "block" whereas in IRD documentation, this element is referred to as an "object." The tables below group IRD objects based on their IRD toolbar category name and point to the corresponding functionality in this release of Composer. Summary information is presented below.

Data & Services

IRD Object Name	Composer Block Name	Description
Database Wizard	DB Data	DB Data retrieves information from the database. Uses a the Query Builder Query Builder.
Web Service	Web Service	Invokes Web Services. GET, POST and SOAP over HTTPS are supported.
	Web Request	Invoke any supported HTTP web request or REST-style web Service. See sample: Routing Based on Web Request.

Miscellaneous

IRD Object Name	Composer Block Name	Description
Assign	Assign	Assigns a computed value/

Multi-Assign		expression or a literal value to a variable. Variables are defined in the Entry block. Capable of multiple assignments.
Call Subroutine	Subroutine	Creates reusable sub-modules.
Entry	Entry	Sets global error (exception) handlers. Defines global variables (see Variables section below).. All routing strategy diagrams must start with an Entry block.
Exit	Exit	Terminates the strategy and returns control back to calling workflow in case of a subroutine.
Error Segmentation	Multiple error output ports can be created in Composer blocks based on each block's Exception property.	
Function Multi-Function	ECMAScript	Builds an ECMAScript expression using the Expression Builder. Many URS functions are available as Genesys Functional Modules described the Orchestration Server Documentation Wiki Can invoke multiple functions.
If	Assign, Branching, ECMAScriptBlock blocks all open Expression Builder	Expression Builder can be used to create IF expressions.
Multi-Attach	ECMAScript	Can be used for attaching data to an interaction.

Routing

IRD Object Name	Composer Block Name	Description
Selection	Target	Routes an interaction to a target, which can be Agent, AgentGroup, ACDQueue, Place, PlaceGroup, RoutePoint, Skill, or Variable. Skill target uses Skill Expression Builder.
Percentage	Target	Order Property Statistics Order property in Target block, lets you perform percentage allocation. Also see sample: Routing Based on Percent Allocation.
Default	Default Route	Routes the interaction to the default destination.
Routing Rule		Orchestration Server 8.1 does not support service level routing

		rules.
Switch to Strategy		Orchestration Server 8.1 does not support switch to strategy routing rules.
Force Route	Force Route	Not exposed as a routing rule in Composer.
Statistics	Target	Although statistical routing rules are not yet supported as in IRD's Statistics routing object, users can use the Target object Property Statistic property to route based on the value of a statistic. A Statistics Manager and Builder let you create your own statistics from URS predefined statistics.

Segmentation

IRD Object Name	Composer Block Name	Description
ANI	Branching	See Your First Application: DNIS Routing for an example.
DNIS	Branching	See Your First Application: DNIS Routing for an example.
Date	Branching	See the sample: Routing Based on Date and Time.
Day of Week	Branching	See the sample: Routing Based on Date and Time.
Time	Branching	See the sample: Routing Based on Date and Time.
Classification Segmentation	Branching	For classification segmentation, an ECMAScript function determines if a particular category name or ID exists in the array of category objects represented by an application variable.
Generic	Branching	Use as a decision point in a workflow. It enables you to specify multiple application routes based on a branching condition.

Also see Context Services Blocks.

Voice Treatment

See Composer Equivalent to IRD Treatment.

eServices Multimedia

See Composer Equivalent to IRD Multimedia.

Outbound

See Outbound Common Blocks

Context Services

See Context Services Blocks

Business Process

See Interaction Processing Diagrams Overview and Interaction Process Diagram Blocks.

Reusable Objects

- IRD List Object: See Composer's List Object Manager.
- IRD Variable List Dialog Box: See Entry block Variables property.

In contrast to IRD, which defines variables in a special dialog box outside of the strategy, Composer defines both workflow and Project variables.

Mac OS Support

Starting with Composer 8.1.3, the MacOS X operating system is partially supported. Consult your Genesys representative for availability or more information on this.

See the Supported Operating Environment: Composer page for more detailed information and a list of all operating systems.

Important

Support for MacOS X ended with version 8.1.300.80. The last 8.1.3 release that supported MacOS X is 8.1.300.78.

Features Introduced in Composer 8.1.x Releases

This section describes the new Composer 8.1 features.

Composer 8.1.3

This section describes the new features in Composer 8.1.3.

- Support for Eclipse 3.7 (Indigo) and 4.2 (Juno).
- Composer is installed as a set of plug-ins.
- Localization support. Language Packs that provide translations for Composer can now be produced and then installed on top of Composer, allowing Composer to run in languages other than English. Localization of generated VXML and SCXML applications is also supported.
- Support for Mac OS (see Composer 8.1 Deployment Guide for features supported).
- Database passwords in connection profile can be encrypted.
- Common bundled Composer Project files can be updated at any time.
- Command line code generation.
- Composer Projects can track change revisions, and revision history can be viewed by the user.
- Customizable global system event handlers in interaction process diagrams.
- New properties in Target block to support updating the DN of the reserved resource to include the access code returned by URS.
- The ECMAScript block is now also available in callflow diagrams, similar to its workflow/interaction process diagram counterpart.

New blocks for workflow diagrams:

- The TLib block adds support for TSendRequest-based requests to Genesys T-Server through the TLIB protocol.
- The SingleStepTransfer block adds support for the <ixn:singlesteptransfer> ORS action. This transfers a voice call directly, without creating another call leg.
- The Raise Event and Cancel Event blocks are provided to raise events in the current SCXML session or to cancel a delayed event.

Voice and Route:

- All toolbar button can be used to generate code for all diagrams in a project.
- Expression Builder now lists custom Javascript functions from a Project's included JS scripts.

- Block tooltips allow the user to see a summary of a block's properties at a glance, without pulling up the Properties View (experimental feature).

Composer 8.1.2

This section describes the new features in Composer 8.1.2.

Common Features Across Applications:

- The Business Rules block works directly with Genesys Rules Engine and does not require going through GRAT server at runtime. This simplifies the usage of Genesys Business rules in VXML and SCXML applications.
- Enhancements made to Database blocks support Database clusters and secure connections; this enables users to connect to Oracle RAC and SQL clusters.
- Database Connections can use service names in addition to SIDs. Connection strings can be dynamically generated and support variables. This helps developers to simplify the usage of database connections.
- Blocks in either Orchestration workflows (SCXML) or Voice call-flows (VXML) can be disabled. For example, you may wish to temporarily remove a block during debugging or during development. Disabled blocks do not participate in the application at runtime.
- New properties for Logging are available for all blocks. Additional support for Alarms is added to workflows. This feature allows developers to minimize insertion of Log blocks and improves readability.

GVP application (VXML) Enhancements:

- Support for Outbound Campaigns in callflows. New Outbound blocks enable callflow applications to update, add or delete records in Campaign Calling Lists and work as a solution in tandem with Genesys Outbound Solutions. Users can also update Do Not Call lists in an Outbound Solution through callflow diagrams.
- Callflow applications (VXML) can use the Operation Parameter Module (OPM) and Audio Resource Management (ARM) features of Genesys Administrator Extension. OPM enables simplification of the overall solution by allowing business users to easily control and manage callflows.
- This release adds a new utility function to access SIP header values in callflows.
- A new VXML code block allows the embedding of VXML code directly into callflows through <subdialog>. This feature provides developers the flexibility to modularize callflow diagrams.
- Users can specify custom formats for Voice prompts in a VXML applications. Custom formats can be created via ECMAScript functions in callflows.
- Input and ICM variables defined in callflow Entry blocks are initialized to default variables if no value is supplied at runtime. This behavior is controlled by a flag. Older version of callflows will continue to have this flag reset to maintain backward compatibility.

Orchestration Application (SCXML) Enhancements:

- Enhanced debugger support provides the ability to debug SCXML applications. The Composer interface provides full debugging functionality for Composer-generated and hand-coded applications.
- New Outbound Campaign blocks support integration with Genesys Outbound Contact features, such as

adding, deleting, and updating Calling List records; updating Do Not Call lists; and other Calling List manipulation features. This functionality provides more robust integration between Routing logic and Genesys Outbound Contact functionality.

- New blocks support the SCXML <parallel> functionality allowing developers to define applications that can simultaneously perform multiple operations. Entry, Subroutine and Begin parallel blocks in workflows and sub-workflows support target-less transitions, which could be based on some condition.
- Support for Genesys Administrator Extension Operation Parameters (OPM) and Audio Resource Management (ARM) functionality in SCXML applications. This feature simpifies the solution and provides control to the end user, addressing Total Cost of Ownership (TCO).
- Voice Treatment blocks provide direct access to Extension data returned after treatment completion. Composer now supports Orchestration Server-based treatments instead of Universal Routing Server-based treatments.
- Support for multiple views for Workbins and existing queues within interaction process diagrams (IPDs).

Composer 8.1.1

This section describes the new features in Composer 8.1.1.

New Routing Application Features

New features for creating SCXML-based routing applications include:

-
- An Orchestration Server (ORS) Debugger, which gives ability to debug SCXML applications including routing applications. The applications can be Composer-generated, hand-coded or a mix of both.
- When specifying ORS preferences, you can enable secure communications (SSL/TLS) between the Composer client and ORS, for SCXML debugging sessions. The connection between Composer and ORS is mutually-authenticated TLS if implemented on the ORS side.
- Routing blocks, as well as those involved in interaction processing, support multi-site routing: Target, Route Interaction, Queue Interaction, Force Route, Routing Rule, Default Route, Create E-mail, E-mail Response, E-mail Forward, Chat Transcript, and Create SMS. See new properties Detach and Detach Timeout.
- Support for development of "interaction-less" processing has been added, which allows the creation of SCXML applications that may be started/interacted with via ORS Web Services, rather than an interaction. The following features support "interaction-less" processing:
    -
    - Blocks that influence interactions now support selecting the interaction they should use. The default behavior is to use the current interaction which is backwards compatible.
    - Wait for Event in the interaction process diagram, which can be set to not wait for a startup or triggering event thereby enabling interaction less workflows.
- To support "interaction-less processing," the following blocks add a new property, Interaction ID:
    - Routing blocks: Default Route, Force Route, Route Interaction, Queue Interaction, Routing Rule, Stop Interaction, and Target.
    - Flow Control blocks: Disconnect, and Exit
    - eServices blocks: Chat Transcript, Create Email, Create SMS, Email Forward, Email Response,

Identify Contact, Update Contact, Create Interaction, and Render Message.
    - Voice Treatment: Create User Announcement, Delete User Announcement, IVR, Play Application, Play Message, and Play Sound.

- Interaction process diagrams add a Namespaces property, which gives the ability to refer to custom namespaces in generated code.
- Interaction Queue blocks in interaction process diagrams support segmentation based on views. Multiple views can be defined and each can redirect flow to a different workflow diagram.
- The following Flow Control blocks are available when creating an interaction process diagram: Branching, ECMAScript, and Log.
- When segmenting interactions to take different paths in a workflow, you now have the ability to define a default limit for each segment.
- When using the Media Server block to specify interactions of a particular media type for an interaction process diagram, the following servers are now available for selection: Chat Servers and Third Party Servers (such as one used for Capture Point application). The Publish operation creates endpoints for these server types.
- When using the Route Interaction block, a new Hints property allows you to specify extension data. The following blocks also add the Hints property: Cancel Call, Create User Annoucement, Delete User Annoucement, Default Route, Queue Interaction, Play Application, Play Sound, Play Message, Routing Rule, Target, User Input.
- When using the Route Interaction block, a new Hints property allows you to specify extension data. The following blocks also add the Hints property: Cancel Call, Create User Annoucement, Delete User Annoucement, Default Route, Queue Interaction, Play Application, Play Sound, Play Message, Routing Rule, Target, User Input.
- The Play Application block adds a new property, Use User Data. When set to true, Composer will automatically update the interaction's user data with the input/inout parameters specified in the Parameters property.
- The Target and Force Route blocks add a Type property, which you can use to define the type of redirection processing.
- The Route Interaction block and Target blocks add a new property, Include Requests From Previous Blocks, which can be used for cascaded target lookups.
- A new Wait block can be used to have Orchesration Server transition out when one of a defined list of events is received and the associated condition is true.
- When using Composer's Business Rule block to request the Genesys Rules Engine to execute a Rule Package in a routing workflow or voice callflow, the getUData() function is now available.

New Voice Application Features

- New features for creating voice applications for GVP include:
- VXML callflows now support a VXML application root document. This enables features like global variables that are available across all callflows and sub-callflows.
- The Prompts property in the following blocks allows VoiceXML to overlay text into an existing video image/stream: Prompt, Menu, Input, DB Prompt, DB Input, Grammar Menu, Record, and Menu.
- The Menu block supports specifying DTMF for repeating a menu.

New Voice & Route Application Features

- While exporting a .WAR file, each Project can specify a unique name which is included in the .WAR file.
- When using Context Services, you can specify a particular media type for a service, which can be a Configuration Server Business Attribute, such as for an Application Type. The following blocks add the Media Type property: Start Service, Associate Service, Complete Service, Enter State, Complete State, Start Task, and Complete Task.
- When defining parameters for the Backend, Web Request, Web Service, Subroutine, and Subdialog blocks, you can now use Expression Builder.

Composer 8.1

This section describes the new Composer 8.1 features.

IRD to Composer Migration Support

Starting with Composer 8.1, you can migrate routing strategies created with Interaction Routing Designer (IRD) 8.0+ into Composer Projects as SCXML-based workflow diagrams, which can run on the Orchestration Platform. The migration process uses an import wizard to handle the transformation from an IRD strategy into a Composer workflow diagram. The *IRD to Composer Migration Guide Wiki* details the migration process.

- Composer can now interface with the Genesys Rules Engine, which is part of the Genesys Rules System. A Composer-compatible plug-in is available for developing business Rule Templates. This plug-in is provided as part of the Genesys Rules System. For information on installing the plugin
- A new Business Rule block lets you request the Genesys Rules Engine to execute a particular set of business rules in a routing workflow or voice callflow and get the results back.

Note:	In the Genesys 8.1 release, the Genesys Rules System will only be packaged with the intelligent Workload Distribution product and the Conversation Manager product.

Composer moves closer to parity with Universal Routing's strategy creation tool, Interaction Routing Designer (IRD).

- An E-mail Response block combines the functionality of IRD's Acknowledgement, Autoresponse, and Create Notification objects.
- A Chat Transcript block allows you to generate a reply e-mail to a chat interaction and attaches a chat transcript.
- An E-mail Forward block combines the functionality of IRD's Forward E-mail, Redirect E-mail, and Reply E-mail from External Resource object.
- A Screen Interaction block allows you to screen a text-based interaction for specific content (specific words or patterns), and then (optionally) segment the interaction to different logical branches based on the result of the screening query.
- A Classify Interaction block allows you to classify a text-based interaction based on content, and attach one or more Classification categories to the interaction.
- For classification segmentation, an ECMAScript function determines if a particular category name or ID exists in the array of category objects represented by an application variable. This variable can be the output of the Classify Interaction block, enabling the Branching block to be used for segmentation based on category.

- For manually attaching categories to an interaction, the User Data block can be used and then a branching block can be (optionally) used to segment interactions to different logical branches based on the different categories.
- An Update Contact block allows you to update customer profile information in the UCS Database, based on data attached to an interaction.
- An Identify Contact block can identify a contact based on the interaction User Data; return a list of matching Contact IDs based on the User Data; create a contact record in the UCS Database with information in the User Data if a matching contact is not found; or update the UCS Database record of the matching contact with information from the current interaction's User Data.
- A Create Interaction block allows you to create an interaction record in the Universal Contact Server Database for a customer contact. This saves the current interaction being processed by the strategy, in the database.
- A Render Message block provides the ability to render field codes in arbitrary text.

Other New Routing Application Features

- Composer's existing Create E-mail block is enhanced to allow you to: pick up standard response text from User Data; specify that the "To" address be picked up from the Customer Profile in the Universal Contact Server Database; and use Field Codes in standard responses that will later be filled in with user-specific values.
- Composer's existing Route Interaction block now allows you to create applications where routing is based on schedules from Genesys Workforce Management.
- The Flow Control palette for routing applications contains a new SCXML State block. When used in a workflow diagram, it allows you to write custom SCXML code that Composer will include in the SCXML document that it generates based on the workflow diagram.
- The Flow Control palette for routing applications contains a new User Data block for updating an interaction's User Data and for attaching Business Attributes, Categories, and Skills.
- When an interaction process diagram (IPD) uses a Workflow block, if the referenced workflow diagram contains an eServices block that names a server performing an action or operation, Composer adds a visual indicator in the form of a node (similar to an IRD strategy-linked node).
- When developers work with Context Services, Composer accepts HTTP basic authentication credentials and uses them for authentication, including digest authentication for working with Web Services.
- You can now use variables in Skill Expression Builder. You can also disable Skill Expression validation from the Configuration Server preference page.
- You can now include your own custom JavaScript (*.js) files in workflows by placing them in the /include/user folder. The JavaScript functions in the specified .js file can then be used in Assign or Branching block expressions.
- A new Composer Route Project template is available: Forward to External Resource.
- Composer's database Query Builder and Stored Procedure Helper now support table synonyms.
- New Integrated Voice and Route Project templates are available: Load Balancing and Working Hour Routing, External File-Based Routing, and Play Application and Busy Treatment.

GVP-Specific Enhancements

- The Transfer block provides a property for setting an authorization code (authcode). It can be populated either from free-form text or from an application variable.
- The Call Trace view used for debugging a callflow displays the line number for each incoming metric.

- A "barge-in" option is available for prompts. The Interruptible property for the following blocks add a new option for DTMF-only barge-in mode: Prompt block, DB Prompt block, Input block, Menu block, Grammar Menu block, and DB Input block.
- Automatic selection of language-specific pre-recorded prompts, grammars, and TTS prompts is now available during application execution. The following blocks add a new Language property: Prompt block, DB Prompt block, Input block, Menu block, Grammar Menu block, DB Input block, and Record block.
- The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of the block. The language setting reverts back to its previous state after the block is done.
- The Language property affects the language of grammars used for ASR input for the following blocks: Input block, Transfer block, and Route Request block.
- The Record block's Capture Filename Prefix and Capture Location properties now allow selection from application variables in addition to accepting literal strings.
- You can now use the GVP ICM Adapter in VoiceXML applications, including invoking services, responding to requests, and sharing data.
- A new ICM Interaction Data block, available on the CTI Blocks palette, supports sending of variables to Intelligent Contact Management (ICM).
- A new ICM Route Request block, also available on the CTI Blocks palette, supports routing the call to CTI.
- The Exit block's Return Values property dialog allows you to select the ICM variables to be returned.
- Voice Projects now have a Project-level flag, which controls whether ICM variables are available for selection and assignment to variables within Composer's Entry block.
- SSML tags can be used in prompts.

Security Enhancements

- The Web Service block now supports certificate-based authentication. You can develop both voice (VXML) and routing (SCXML) applications that support secure mutual authentication and communication with a Web Service. Composer supports the use of both a digital client certificate and server certificate contained in a keystore file.
- When creating a routing application and connecting to Configuration Server, Composer displays informational text associated with both successful and unsuccessful authentication.
- You can configure an inactivity timeout for the connection to Configuration Server as well as when the timeout warning dialog should appear.
- You have the option of having a configurable security banner appear when Composer is first launched, similar to other Genesys applications. For information on configurable items related to the banner, see the Genesys 8.1 Security Deployment Guide.
- Composer supports secure connections when connecting to GVP's Media Control Platform and when connecting to Context Services and for Universal Contact Server.
- Composer now has Transport Layer Security (TLS) support and adheres to Federal Information Processing Standards (FIPS) in its connection to Configuration Server and to GVP Media Control Platform.

Routing & Voice Applications

- When organizing custom blocks, you have the option of creating new drawers in the palette. You can also select from a set of bundled custom icons for the custom blocks you create.

- Expression Builder now returns to its last state when re-opened, which includes displaying the tree and the location in the tree in the Expression Builder Data area.
- The Expression Builder filter now works on the description of the functions in addition to the function signatures.
- Expression Builder data loading is optimized to run in a separate thread. As a result, dialogs remain responsive while data loading is in progress.

New operating system support for 8.1 is as follows:

- Composer can run on the Windows 2003, Windows 2008 (32-bit and 64-bit in 32-bit compatibility mode), Windows XP, Windows Vista, and Windows 7 (32-bit and 64-bit in 32-bit compatibility mode) operating systems. For more information, see "Operating Systems Supported" on page 41.

Getting Started with Composer

This section is your first stop for getting started with Composer. It discusses the following topics:

- Running Composer for the First Time
- Software Updates Functionality
- Integrating with Source Control Systems
- Composer Projects and Directories
- Security Configuration
- Upgrading Projects/Diagrams
- Keyboard Shortcuts
- Menus
- Toolbars

Also see: *Composer 8.1 Routing Applications User's Guide*.

Running Composer for the First Time

Setting Up Your Workspace

When you run Composer, before the user interface appears, a dialog box opens with a suggested *workspace*, which is a location (folder) for your projects and files in addition to any special folders that Eclipse needs to maintain for its internal bookkeeping. The dialog box gives the option of changing the workspace to a different location. New projects created in Composer will be created under this workspace as subfolders. After the Composer interface opens, the Project Explorer shows this location. You can change this location by selecting **File** > **Switch Workspace**. **Notes:**

- Genesys recommends that the workspace folder name has no spaces in its path (for example, `c:\comp81dev`). This recommendation is not required and Composer does not enforce this. Genesys also recommends using the component version in the name to avoid confusion during upgrades.
- When prompted for a workspace folder, do not specify parenthesis in the workspace path.
- The workspace should not be located in a ClearCase view, as this will cause problems accessing files later during development.

Opening Composer

The first time you open Composer, it asks you to specify the location of your workspace. Eclipse remembers this location and will present it for all subsequent times when you open Composer. If you do not wish to be prompted each time for this path and plan to use the same location for all your projects, you can check the Use this as the default and do not ask again option to skip this screen on future launches.

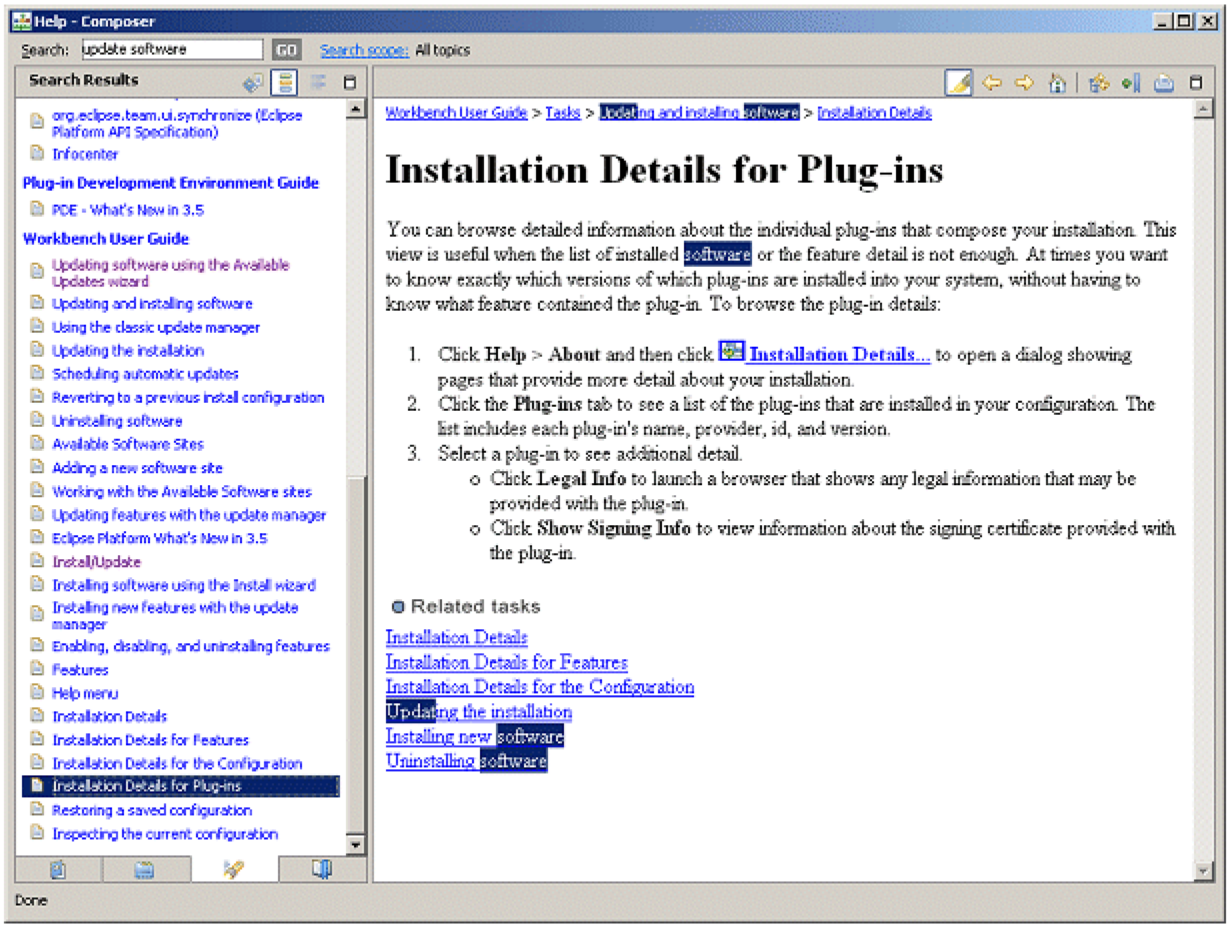
Welcome Screen

When a new Workspace is created for the first time, you will be taken to the Welcome screen, which provides getting started overview topics, tutorials and links for references on the Web. The first time you enter the Workspace, select **Tutorials** and choose **Configure Settings**. The next time you open Composer, if the Welcome screen still opens, click on the **Workbench** link. Or close the Welcome screen by clicking the "**x**" on the Welcome tab. To go back to the Welcome screen at any time:

- Select **Help** > **Welcome**.

Software Updates Functionality (Plugins)

You can find information on installing new software (such as for Dynamic Web Projects or updating existing plug-ins in the Eclipse help. For example, using "update software" as search words displays plugin topics in the Eclipse *Workbench User Guid*e (**Help** > **Contents**).



org.eclipse.team.ui.synchronize (Eclipse Platform API Specification)
Infocenter
Plug-in Development Environment Guide
PDE - What's New in 3.5
Workbench User Guide
Updating software using the Available Updates wizard
Updating and installing software
Using the classic update manager
Updating the installation
Scheduling automatic updates
Reverting to a previous install configuration
Uninstalling software
Available Software Sites
Adding a new software site
Working with the Available Software sites
Updating features with the update manager
Eclipse Platform What's New in 3.5
Install/Update
Installing software using the Install wizard
Installing new features with the update manager
Enabling, disabling, and uninstalling features
Features
Help menu
Installation Details
Installation Details for Features
Installation Details for the Configuration
Installation Details for Plug-ins
Restoring a saved configuration
Inspecting the current configuration

Workbench User Guide > Tasks > Updating and installing software > Installation Details

Installation Details for Plug-ins

You can browse detailed information about the individual plug-ins that compose your installation. This view is useful when the list of installed software or the feature detail is not enough. At times you want to know exactly which versions of which plug-ins are installed into your system, without having to know what feature contained the plug-in. To browse the plug-in details:

1. Click Help > About and then click Installation Details... to open a dialog showing pages that provide more detail about your installation.
2. Click the Plug-ins tab to see a list of the plug-ins that are installed in your configuration. The list includes each plug-in's name, provider, id, and version.
3. Select a plug-in to see additional detail.
   - Click Legal Info to launch a browser that shows any legal information that may be provided with the plug-in.
   - Click Show Signing Info to view information about the signing certificate provided with the plug-in.

Related tasks

Installation Details
Installation Details for Features
Installation Details for the Configuration
Updating the installation
Installing new software
Uninstalling software

Plugin Installation Requirements

It is standard Eclipse behavior for plugins to be installed in C:\Program Files\... (i.e. the base Composer installation directory), and NOT in the user's workspace. The installation of plugins must be performed by an adminn user.

Dynamic Web Projects

After you install Java EE Developer Tools plugins, you can create a Dynamic Web Project containing pages with active content. Unlike with static Web Projects, dynamic Web Projects enable you to create resources such as JavaServer Pages and servlets. Here's how to get started:

1. **Composer Help** >> **Install New Software**.
2. Click **Add**. In the resulting box, enter http://download.eclipse.org/releases/galileo/
3. Select it to see the available package.
4. Select the **Web, XML, and Java EE Development Eclipse Java EE Developer Tools** entry.Install the plugins.
5. Restart Composer.
6. Create a Dynamic Web Project.

Note: Other missing project types can be similarly enabled.

Integrating with Source Control

This section describes setup instructions for using source control with Composer. The ClearCase source control system is supported as well as Subversion.

ClearCase Plug-in Installation

To install ClearCase plug-ins for use with Composer:

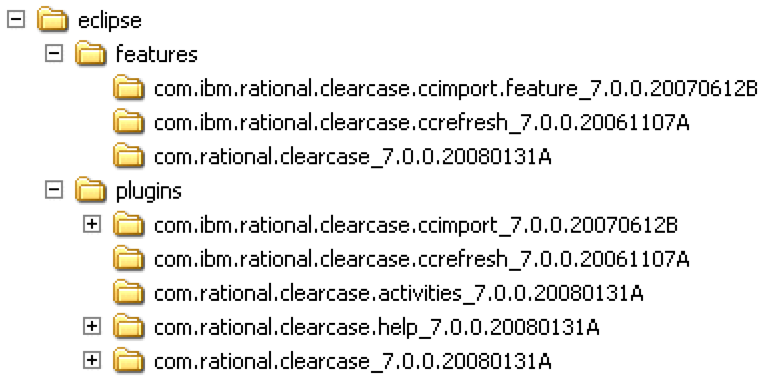
1. Install ClearCase on the machine that Composer will run on.
2. Exit Composer. If you want to print these instructions to use after closing Composer, click the Print Page icon at the top-right of the Eclipse Help window.
3. Install the ClearCase Eclipse plug-in from IBM:

    a. Get the following files from IBM's website:com.rat.cc.win32-20080131A.zip and com.r.cc.ccrefresh.all-20061107.zip

http://www.ibm.com/developerworks/rational/library/content/03July/2500/2834/ClearCase/clearcase_plugins.html

Note: Actual ZIP file and directory names may change as IBM continues to update the plug-ins. These documented names are current at the time of this writing. These plug-ins are for Eclipse v3.3 and not 3.4. Even though Composer 8.1. is based on Eclipse 3.4, Genesys recommends that you install the plug-ins listed for Eclipse 3.3.

    b. Extract the two ZIP files to a location of your choosing, essentially merging the contents. There is a duplicate file .eclipseextension in the two ZIPs; you can safely ignore that file. The ZIP files, when extracted, create a directory structure like:



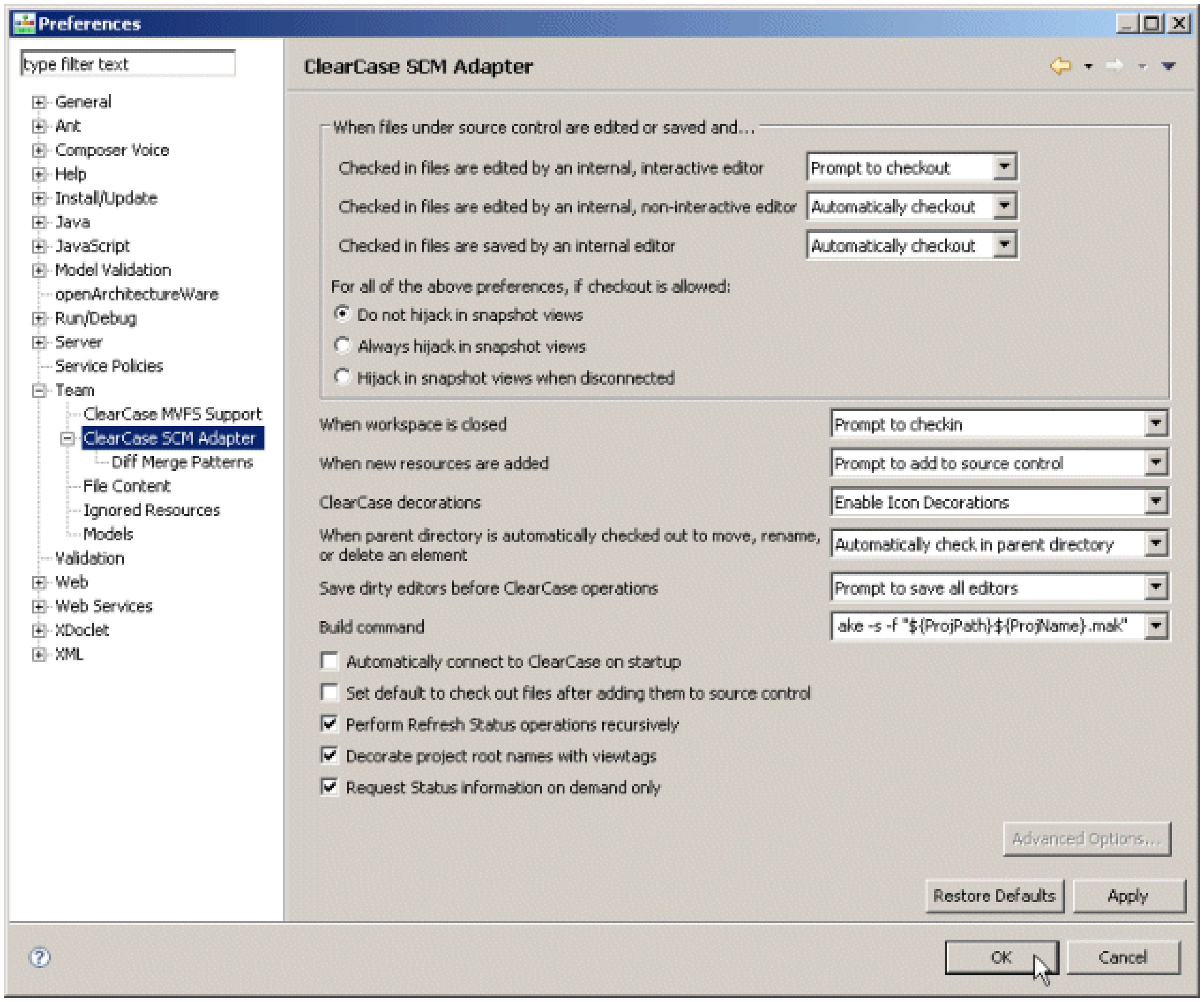
```
eclipse
    features
        com.ibm.rational.clearcase.ccimport.feature_7.0.0.20070612B
        com.ibm.rational.clearcase.ccrefresh_7.0.0.20061107A
        com.rational.clearcase_7.0.0.20080131A
    plugins
        com.ibm.rational.clearcase.ccimport_7.0.0.20070612B
        com.ibm.rational.clearcase.ccrefresh_7.0.0.20061107A
        com.rational.clearcase.activities_7.0.0.20080131A
        com.rational.clearcase.help_7.0.0.20080131A
        com.rational.clearcase_7.0.0.20080131A
```

    c. If it does not already exist, create the directory `${Composer 8.1.Install}\features`, where ${Composer 8.1.Install} is the installation root of your Composer 8.1 installation.For example, `${Composer 8.1.Install} might be C:\Program Files\GCTI\gvp\Composer 8.1`

d. From the directory where the ZIP files were extracted, copy the folders in the features directory to `${Composer 8.1.Install}\features`, and copy the directories in the plugins directory to `${Composer 8.1.Install}\plugins`.

Note: Instead of dropping the extracted zip file content in these two locations, you may place the entire (eclipse folder) extracted content in the following folder location ${Composer 8.1.Install dir}\dropins.

4. Ensure that ClearCase capabilities are enabled.
   a. Go to **Window** > **Preferences** > **General** > **Capabilities**.
   b. Make sure that the **Team** checkbox is checked. It must be checked in order for **ClearCase MVFS Support** and **ClearCase SCM Adapter** preference items to appear in the Preferences window. Please note you must restart Composer to see the changes.
5. Enable MVFS Support:
   a. Start Composer.
   b. Select **Window** > **Preferences** > **Team** > **ClearCase MVFS Support**.
   c. Click the **Workspace** link under**MVFS Support Preferences** and select the **Refresh Automatically** check box.
   d. Click the Apply button.
   e. Click **OK** to close the Workspace dialog box.
   f. Again, open **Window** > **Preferences** > **Team** > **ClearCase MVFS Support** and make sure that **Enable ClearCase dynamic view file system support** is selected, then click **OK**.
6. Configure the ClearCase plug-in. Configuration can be accessed from **Window** > **Preferences** > **Team** > **ClearCase SCM Adapte**r from the tree in the left-side panel. The recommended settings are shown in the image below:



ClearCase SCM Adapter

When files under source control are edited or saved and...

Checked in files are edited by an internal, interactive editor: Prompt to checkout

Checked in files are edited by an internal, non-interactive editor: Automatically checkout

Checked in files are saved by an internal editor: Automatically checkout

For all of the above preferences, if checkout is allowed:

- [✓ Do not hijack in snapshot views]
- Always hijack in snapshot views
- Hijack in snapshot views when disconnected

When workspace is closed: Prompt to checkin

When new resources are added: Prompt to add to source control

ClearCase decorations: Enable Icon Decorations

When parent directory is automatically checked out to move, rename, or delete an element: Automatically check in parent directory

Save dirty editors before ClearCase operations: Prompt to save all editors

Build command: ake -s -f "${ProjPath}${ProjName}.mak"

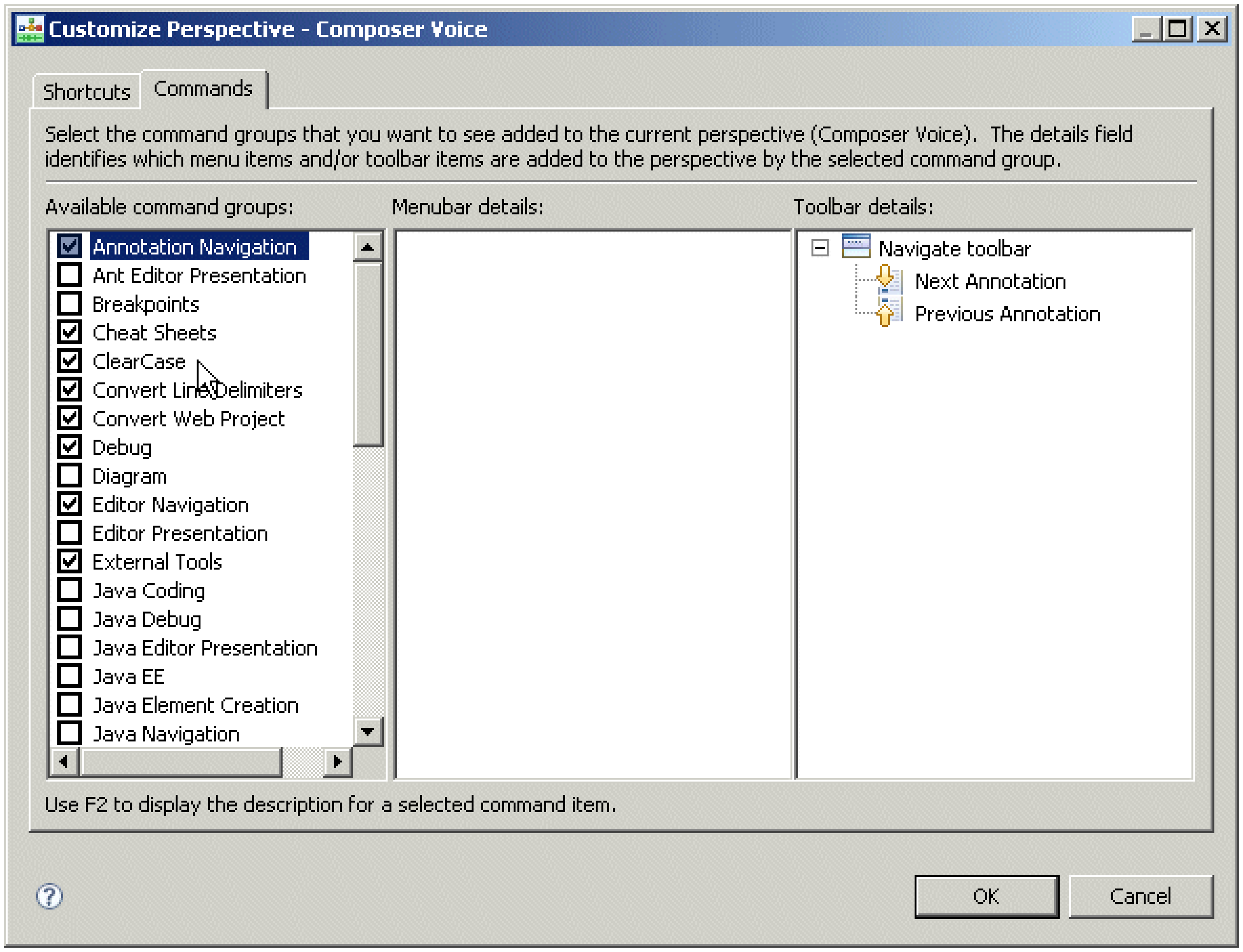
- [ ] Automatically connect to ClearCase on startup
- [ ] Set default to check out files after adding them to source control
- [x] Perform Refresh Status operations recursively
- [x] Decorate project root names with viewtags
- [x] Request Status information on demand only

Note: You can read more about ClearCase features and working with ClearCase view from the help topic Rational ClearCase SCM Adapter available from the Eclipse Help system. These help topics will be available only after installing ClearCase plugins.

ClearCase Usage

To use ClearCase functionality within Composer:

1. Create a view using ClearCase Explorer, or use an existing view.
2. In Eclipse, if you don’t see the ClearCase toolbar, make the toolbar appear by clicking **Window** > Customize **Perspective**. In the dialog box that displays, click the **Commands** tab, select the **ClearCase command group** and click **OK**.


Customize Perspective - Composer Voice

Shortcuts | Commands

Select the command groups that you want to see added to the current perspective (Composer Voice). The details field identifies which menu items and/or toolbar items are added to the perspective by the selected command group.

Available command groups:

- [✓] Annotation Navigation
- [ ] Ant Editor Presentation
- [ ] Breakpoints
- [✓] Cheat Sheets
- [✓] ClearCase
- [✓] Convert Line Delimiters
- [✓] Convert Web Project
- [✓] Debug
- [ ] Diagram
- [✓] Editor Navigation
- [ ] Editor Presentation
- [✓] External Tools
- [ ] Java Coding
- [ ] Java Debug
- [ ] Java Editor Presentation
- [ ] Java EE
- [ ] Java Element Creation
- [ ] Java Navigation

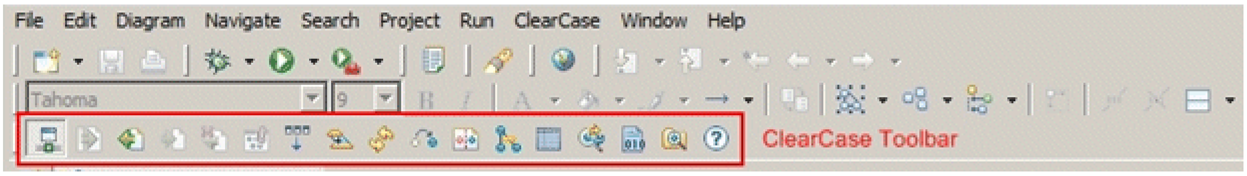
Menubar details:

Toolbar details:

- Navigate toolbar
  - Next Annotation
  - Previous Annotation

Use F2 to display the description for a selected command item.

OK | Cancel



3. Once the toolbar is available, click on the button.

After this, any Composer Project that resides in a ClearCase view will have the view name displayed next to it in the Project Explorer window. Also, the icons for files and folders under source control will show the status, such as checked out, hijacked, and so on.

Creating a Composer Project Managed by ClearCase

To create a new Composer Project that will be managed by ClearCase:

1. Bring up the Project wizard (**File** > **New**).
2. Clear the **Use default location** check box and enter in the **Location** field a path that resides inside your ClearCase view.

The path becomes the root of the Composer Project. Note that the files in the new project will not be checked into ClearCase until you use the Add to Source Control function.

To add a project that is already checked into ClearCase to your Composer 8.x workspace:

1. Use the Import wizard (**File** > **Import**).
2. Select **Existing Projects into Workspace**.
3. In the **Select root directory** field, enter the path of the project residing in your ClearCase view. Check the box corresponding to the Project name.

Important: Leave the **Copy projects into workspace** box unchecked. If it is checked, the imported project will not be under ClearCase control.

To edit a file that is under source control, it must be checked out from ClearCase. After editing, it can either be checked in to create a new source control version, or the checkout can be undone to revert the file back to its previous version. You can also compare changes to the previous version before checking it. All of these operations can be accessed in several alternative ways:

- Right-click the file name in Composer's Project Explorer view, and use the **Team** submenu.
- Select the file in Composer's Project Explorer view, and use the **ClearCase** menu on the top menubar.
- Select the file in Composer's Project Explorer view, and use one of the buttons on the ClearCase toolbar.

Note: If you choose to remove a ClearCase-managed Composer Project from your workspace, you will be prompted with a `Confirm Project Delete` message. Genesys strongly recommends that you choose the **Do not delete contents** option. This leaves the files in your ClearCase view untouched. Otherwise, the files may be removed from source control.

Subversion Configuration

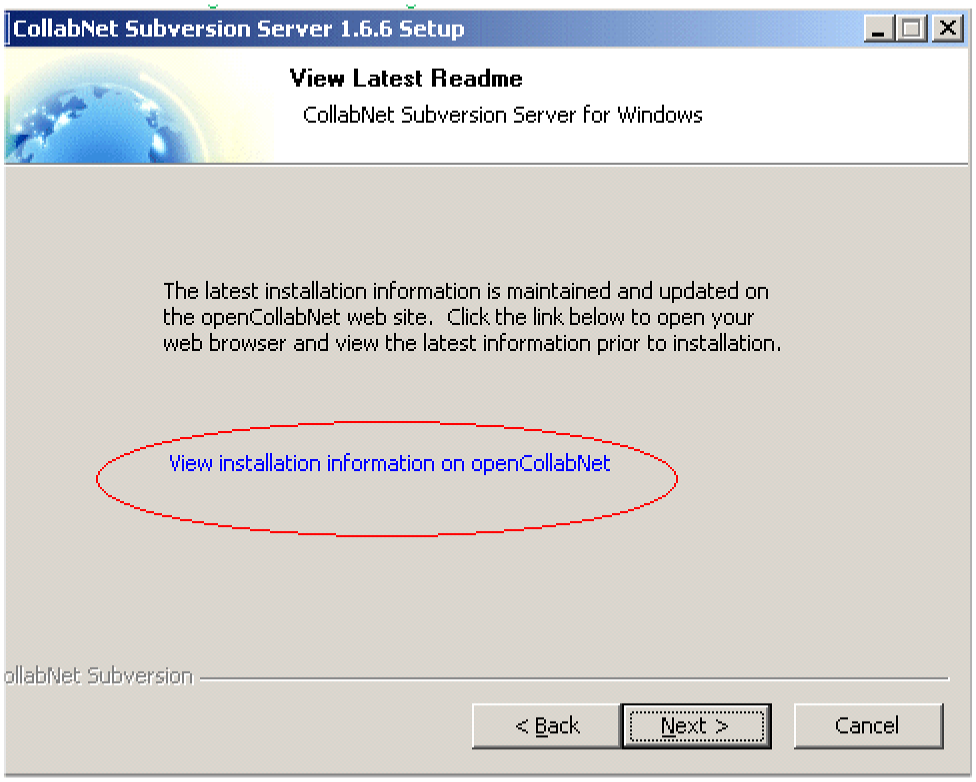
Subversion is a client–server versioning system. You can integrate Subversion with Composer in order to have a version control over Projects. Subversion creates and maintains a repository on the Project web server. Its clients run on Composer machines and connect to the Subversion server over the Internet.

Note! The integration steps below are not version-specific or Composer-specific. The steps refer to the interface and capabilities provided by the Eclipse Integrated Development Environment (IDE) (Subclipse Team Provider plug-in) and CollabNet (CollabNet Subversion Server) products.

The recommended steps are as follows:

1. Install and Configure Subversion Server.
2. Download the Collabnet Subversion server from the website: http://www.collab.net/downloads/subversion/

3. Follow the installation and configuration instructions from the vendor for the operating system you are working with. Complete registration if required. When the installation executable runs, the wizard has an option to View Installation Information.



CollabNet Subversion Server 1.6.6 Setup

View Latest Readme

CollabNet Subversion Server for Windows

The latest installation information is maintained and updated on the openCollabNet web site. Click the link below to open your web browser and view the latest information prior to installation.

View installation information on openCollabNet

Note: If you are running IIS on the server machine during installation of Collabnet Subversion server, you may wish to change the Apache port to “81” instead of the default 80 to avoid conflict.

4. Follow post installation instructions from the vendor and complete Collabnet Subversion server configuration.
5. Configure the Subversion repository. Follow the instructions provided by the vendor (http://www.collab.net/downloads/subversion/) to define the permissions, user name, and password for accessing the repository. Use the instructions for the specific version of the Callabnet Subversion server that you installed in step 1.

Note: Subclipse provides an option for creating a repository, but this is more suited for personal development where you do not need to share your code. Typically, you would set up a Subversion server, create the repository on the server and then point Subclipse at the server.

6. Install the Subversion client using the capabilities of Eclipse by adding Subclipse to the Eclipse IDE. Subclipse is a project to add Subversion support to the Eclipse IDE. Use Eclipse's Software Manager to add Subclipse to our Eclipse IDE.
    a. Add the Subclipse update version compatible with Subversion server installed in step 1.
    b. From Composer's Help menu, choose **Install New Software** to open the Install wizard.
    c. Select the required Subclipse update from the site: (http://subclipse.tigris.org/update_1.6.x/)

Note: The above link may not be current.   In this case, check the http://subclipse.tigris.org/ site. The Download and Install section lists update sites for various Eclipse versions. Choose the correct site based on the Eclipse version for Composer. Composer's Eclipse version can be determined from **Help** > **About** and clicking the Eclipse.org icon.

d. Click on the **Download** link to go to the download page. There you will find the URL to enter into the Eclipse Install wizard.

e. Follow the wizard and instructions provided in the **Help** > **Contents** > **Workbench User Guide** for installing new software.

f. Restart Composer if required by installer.

7. Create the CVN (Subversion) item in Composer menu bar. Follow the Eclipse instructions on customizing perspectives /creating command groups (**Help** > **Contents** > **Workbench User Guide** > **Concepts** > **Perspectives** > **Configuring perspective command groups**.

8. Define CVN repository location to your Eclipse IDE. Follow the instructions provided in **Help** > **Contents** > **Workbench User Guide** > **Getting Started** > **Team SVN Tutorial**. Use the instructions for Specifying a project location.

For more details working with CVN, please, see Help: **Subclipse** - **Subversion Eclipse plugin**.

Checkin Error

If you are using Source Control tools, checking in Composer Projects contents after a Project Upgrade may results in an error. See Checkin Error During Source Code Integration.

Composer Projects and Directories

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.

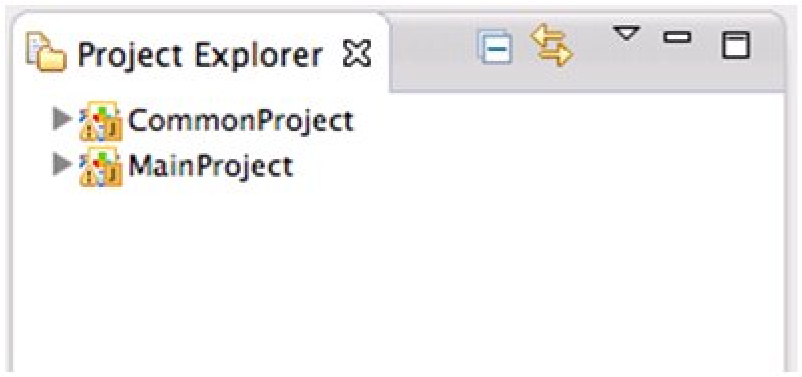
Also uncheck the **Read-only** and **Hidden** properties in the **General** tab for the Project and sub directories. Note: While importing Composer Projects, if the **Copy Projects into Workspace** option is selected, the above mentioned permissions needs to be set for the copied Project directory separately.

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.

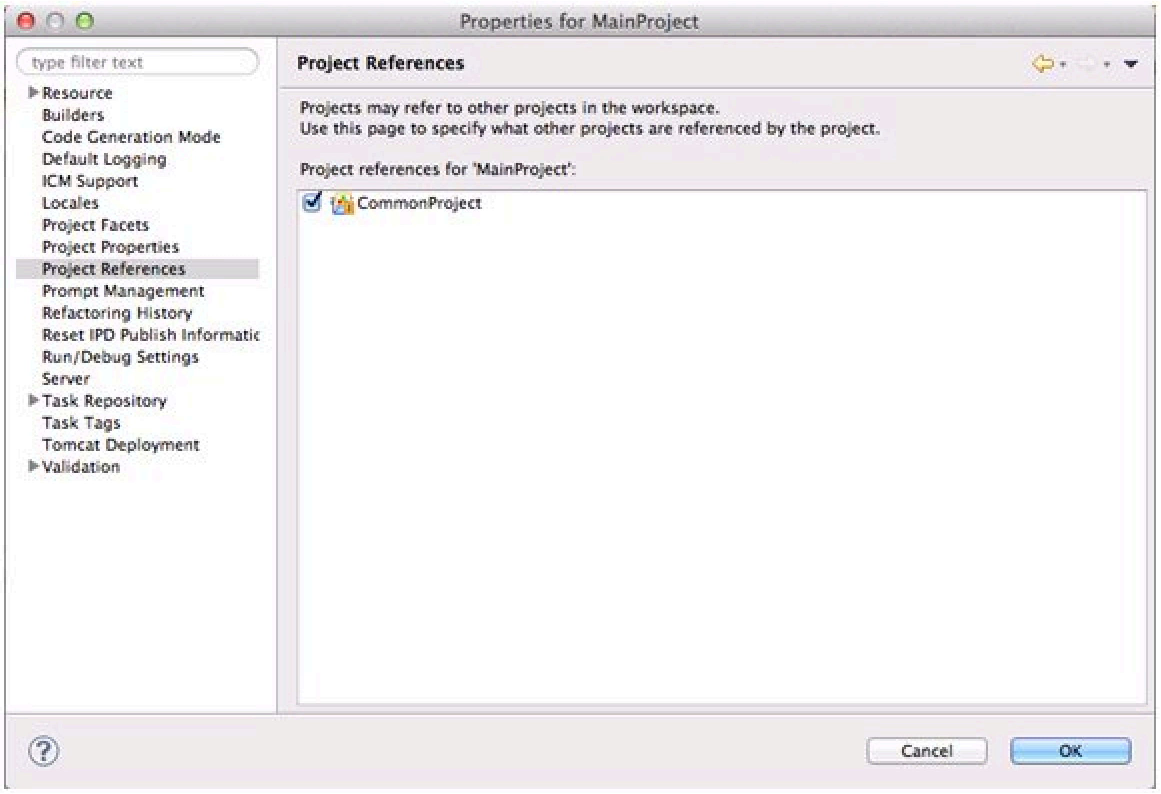
In our example, we will create two Projects:

- `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**:

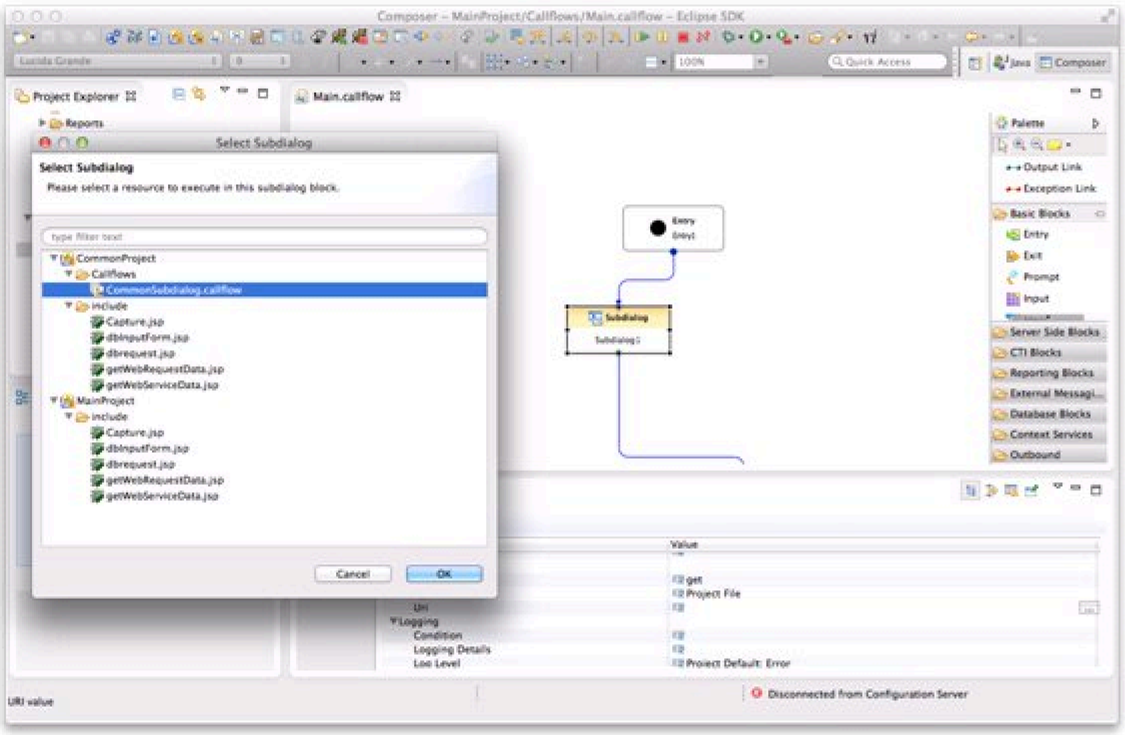


Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for 'MainProject':

☑ CommonProject

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

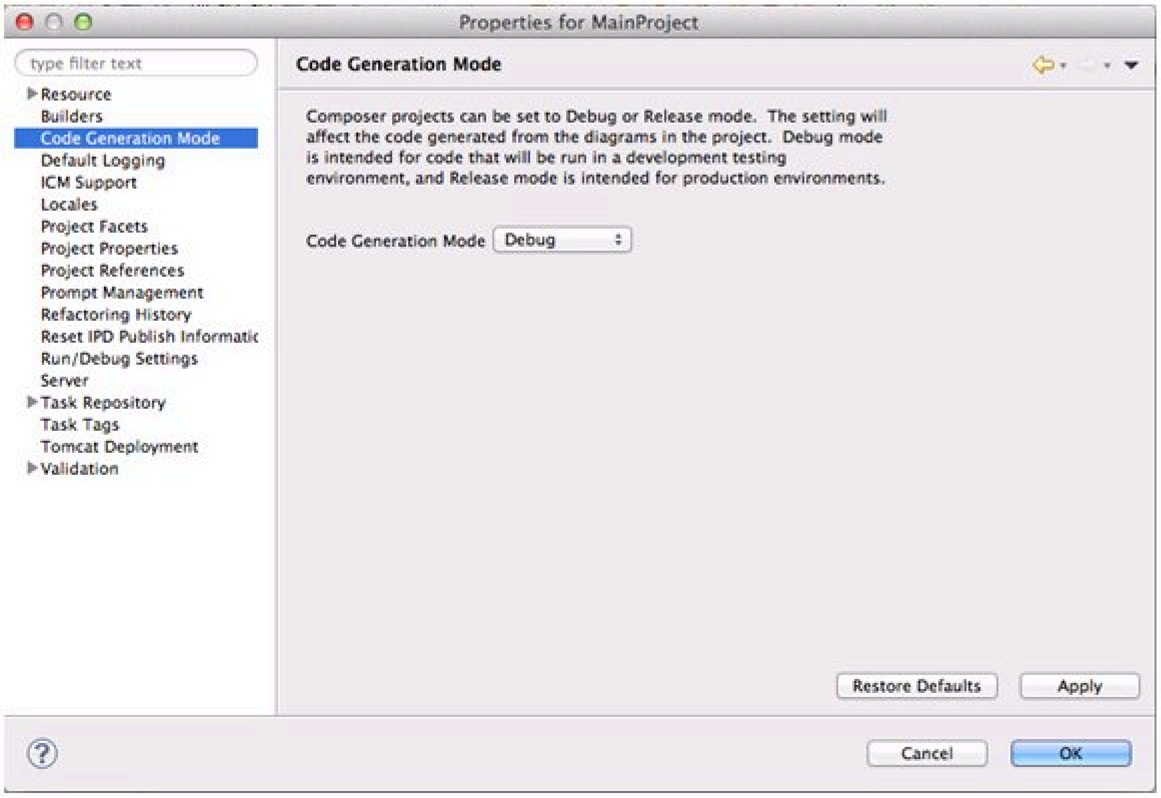


Debug and Release Modes

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:


Properties for MainProject

type filter text

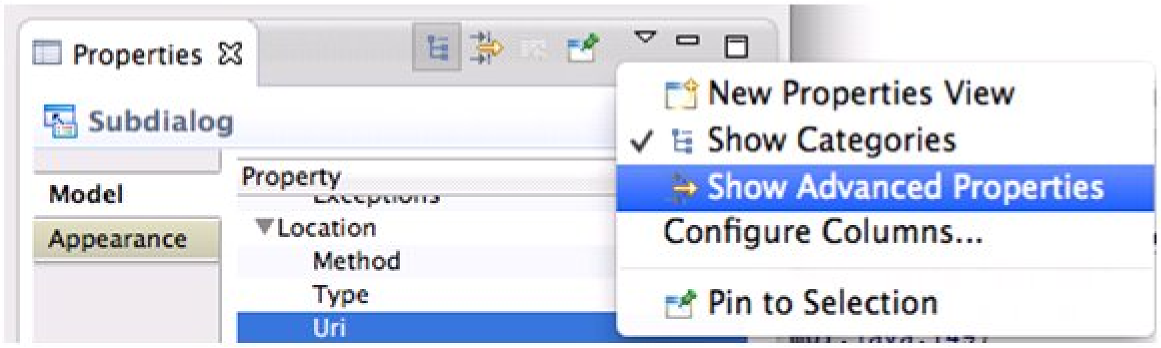
- Resource
- Builders
- Code Generation Mode
- Default Logging
- ICM Support
- Locales
- Project Facets
- Project Properties
- Project References
- Prompt Management
- Refactoring History
- Reset IPD Publish Informatic
- Run/Debug Settings
- Server
- Task Repository
- Task Tags
- Tomcat Deployment
- Validation

Code Generation Mode

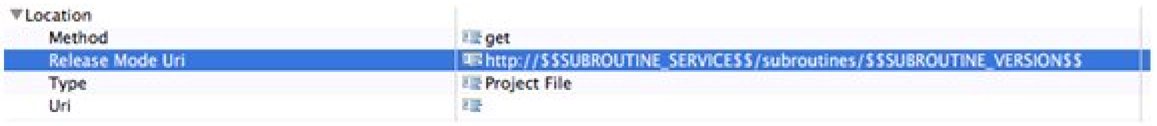
Composer projects can be set to Debug or Release mode. The setting will affect the code generated from the diagrams in the project. Debug mode is intended for code that will be run in a development testing environment, and Release mode is intended for production environments.

Code Generation Mode: Debug

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://$$SUBROUTINE_SERVICE$$/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.

Multiple User Environments

When more than one Composer user attempts to log into the same Workspace, the following message appears: `Workspace in use or cannot be created, choose a different one.` Whenever Composer uses a Workspace, it locks the Workspace so other Composer instances cannot access it. A Workspace is meant to be a "private" development area, until the developer decides to share it with the team. It is not possible to share a single Workspace among multiple users, so you need to set up (private) workspaces for each developer. To merge the work of different developers together, use source control, which could be be SVN (Subversion), Git, or something else. This is the best way to manage a Composer Project with multiple users working simultaneously on it, and prevent the developers from interferring with each other's work.

You could consider the Subversion plugin in Composer as a connector to source countrol like SVN. To install the Subversion plugin, see Software Updates Functionality (Plugins). Continuing with this example, once the Subversion plugin is installed, the Project can be shared using source control. When you right-click on a Project, you will find all the relevant options under the Team menu. For the first time, a Project needs to be shared with source control. After this, there will be options on the Team menu.

Security Configuration

You have the option of configuring:

- A secure Transport Layer Security (TLS) connection between Composer and Universal Contact Server (UCS) during application design when connecting to Context Services.
- A secure TLS connection when connecting to Configuration Server during design time.

You can also configure:

- A security banner that displays when users establish a Configuration Server connection.
- An inactivity timeout. If a Composer user has authenticated with Configuration Server, Composer times out after a configurable number of minutes of inactivity.
- Both certificate-based and key-based authentication.

For information on configuring the above features, see the *Genesys Security Deployment Guide*

Upgrading Projects and Diagrams

Note: Java Composer Projects were referred to as Java Voice projects in earlier versions of Composer, such as Composer Voice. While working with the current version of Composer, an upgrade is required for a previously-created Composer Project and Project diagrams. If you simply copy diagrams into a new Composer Project instead of upgrading the Project itself, then you must use the diagram upgrade procedure as described below. Genesys recommends that you create a dedicated workspace for 8.1 Projects and do not reuse the previously created workspace. This will provide a clean separation between the two versions as well as ensure that a backup copy is preserved for later reference or rollback.

Upgrade Summary

A summary of the Composer diagram upgrade process is as follows:

1. Obtain Composer 8.1 through Genesys Technical Support.
2. Uninstall the older version of Composer. Before uninstalling the older version of Composer:
    - Make a copy of your Composer workspace folder (which contains all your Project files), as your workspace may be deleted if it is located under the installation directory (`C:\Program Files\GCTI\Composer 8.1\workspace`).
    - Uninstall the older version of Composer.
3. Install Composer 8.1.
4. Upgrade at a Project level or at the Diagram level as described below.

Routing Upgrade Limitations

See Migrating IRD Strategies.

Project Upgrades

A Project-level upgrade will automatically apply diagram-level upgrades for all the diagram files directly residing within the diagram (Callflows or Workflows) folder. As part of the upgrade process, Composer makes a back-up of the Project and its files, which are saved under the archived folder; for example: `./JavaComposerProject/archive/JavaComposerProject20100809184446388.zip` To upgrade a Project created in a previous version:

1. Import an old Composer Project into Composer's Project Explorer view. From the menu, select **File** >

Import.

2. In the Import dialog, navigate to **General** and double-click **Existing Projects into Workspace**.
3. Browse to the Composer Project location and select the Project(s).
4. Mark the checkbox **Copy projects into workspace**.
5. Click **Finish**.
6. In the Project Explorer, select the imported project and type **F5** to refresh.
7. Right-click the imported Project and select **Upgrade Composer Project** from the context menu.
8. If the Project is upgraded, a message appears indicating that it is the current version. Otherwise, a prompt appears asking if you would like to upgrade this Project. Click the **Yes** button to start the upgrade process.
9. View the upgrade report. Once the upgrade process is complete, Composer displays a report. The report is located in the Reports folder of the Project; for example:`C:\Work\Temp1\Gate3IPTest\Reports\UpgradeReport_Gate3IPTest20090513155840979.html`

Upgrade Error Message

After a Composer Project upgrade, the Project Upgrade Report may display the following error message: `error while updating the .studio_config.properties`. In this case, permissions for `.studio_config.properties` may be read-only or hidden. To resolve this issue, go to the file system and check for the `studio_config.properties` file located under the Composer Project directory. Set the file permissions so that the read-only and hidden file attributes are disabled/unchecked. Hint: To find where the current Project directory is located, do the following:

1. Go to Composer's Project Explorer view.
2. Right-click the Composer Project.
3. From the shortcut menu, select **Properties** > **Resource** and look for **Location**., e.g., Location: `C:\Program Files\GCTI\Composer 8.1\workspace\JavaComposerProject`

Diagram File Upgrade

In Composer 8.0.2 and later, diagrams for voice applications are called callflow diagrams whereas in earlier versions of Composer they were called *studio_diagrams*. Follow the steps below if you have only copied older diagram files to a current version of Composer Project (or to an already upgraded Composer Project).

1. In Composer's Project Explorer, select the Project destination folder to where you want the files to be imported, such as the `Callflows` or `Workflows` Project folder.
2. Right-click and select **Import**.
3. In the Import wizard, select the diagram files to import.
4. After the import operation completes, right-click on the imported diagram file and select the upgrade option: **Upgrade Callflow Diagram** or **Upgrade Workflow Diagram**.

Changes as a Result of Upgrading

It is important to note the following:

- When upgrading to 8.1.1, references to internal variable names may have to be edited manually. See Variables Project and Workflow, Internal Variables Naming for details and examples. It is recommended that internal variables such as DB Data block database result variables not be used; instead, create User variables to store these results.
- A Project upgrade does not upgrade any custom blocks. When Composer is launched, it checks if any custom blocks need upgrading and upgrades them. There are no manual steps involved.
- When upgrading to 8.0.4/8.1, Project upgrading creates the folder `./WEB-INF/lib`, copies files from `./lib` to `./WEB-INF/lib`, then removes the `./lib` folder from the Java Composer Project.
- When upgrading from 8.0.2, the Entry block variable `_COMPOSER_WSSTUBBING` is renamed`COMPOSER_WSSTUBBING`.
- When upgrading from 8.0.1 to 8.0.2, the Studio Diagram file extension changes from `.studio_diagram` to `.callflow`. For example: `MyDiagram.studio_diagram` changes to `MyDiagram.callflow`.
- To avoid any resulting file name conflict, the diagram upgrade will append a timestamp to the file name only if a .callflow file with the same file name already exists in the same folder; for example: `Main_2010_02_19_123010.callflow`. The Timestamp is of the following format: yyyy_MM_dd_HHmmss
- Starting with 8.0.2, the following callflow blocks contain a mandatory Output Result property: Menu, DB Input, Grammar Menu, Input, Get Access Number, Transfer, Statistics and Record. You supply this property by selecting a variable. Since this property is mandatory; if not supplied, an error occurs in the Problems View when validating the callflow.
- Upgrading to 8.0.2 or higher automatically populates this variable. For example, if the block is a Menu block and the block's name is Main_Menu, upgrading will add a Main_Menu variable to the Entry block (as if you added it manually) and will set the Output Results property to this variable.
- The GVP Next Generation Interpreter does not support the error.badfetch.badxmlpage event. If upgrading a callflow application from an earlier version that listed this event under Supported in its Entry block Exceptions dialog box, you will need to modify that Entry block by removing that event under Supported in the Exceptions dialog box.
- Composer workflow and callflow diagrams do not directly store diagram grid information. This preference is workspace-specific. If you are using a new workspace, you can set this value prior to upgrading Projects and diagrams so that the grid information does not change during the upgrade process.

Note: Workspace preferences can be exported and imported from **File** > **Export** or **Import** > **General** - **Preferences**.

Generating Code for multiple diagrams from command line

A command line option is availabe in Composer 8.1.3 and later to generate code for all diagrams for all projects in a workspace `eclipse.exe -application com.genesyslab.composer.voice.generator.commandline.app -nosplash -console -consoleLog -data .\workspace`

- Eclipse should not be running. This commandline will launch a headless instance of Eclipse that will exit once code generation is complete.

- `Eclipse.`exe should be executed from its installed location.
- `.\workspace` is the relative path to the workspace that contains your projects for which code should be generated
- This will generate code for all supported types of diagrams
  - Callflow : VXML
  - Sub-Callfow : VXML
  - Workflow : SCXML
  - Sub-Workflow : SCXML
  - Interaction Process Diagram : SCXML

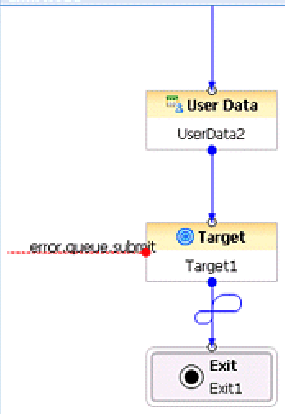
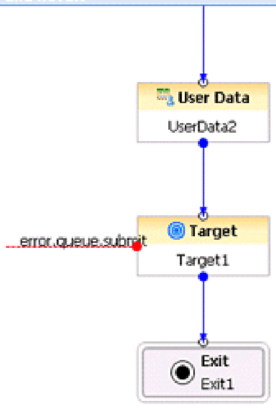
Migrating IRD Strategies

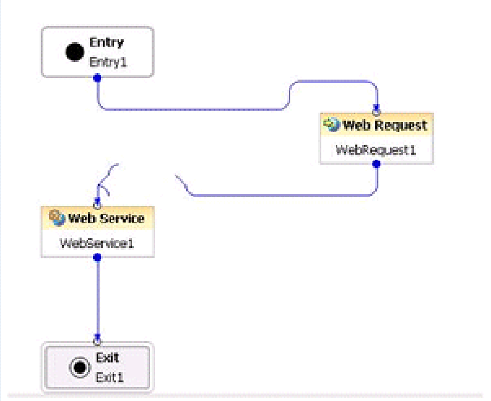
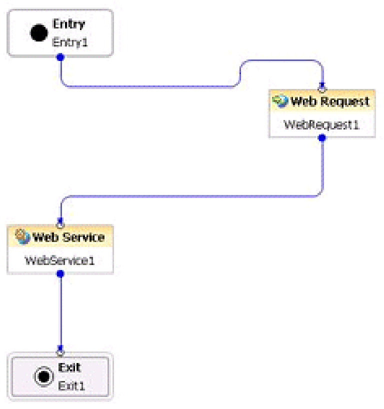
Starting with Composer 8.1, you can migrate routing strategies created with Interaction Routing Designer (IRD) 8.0+   into Composer Projects as SCXML-based workflow diagrams. For more information, see the Composer 8.1 IRD to Composer Migration Guide.

Working with Diagram Layouts

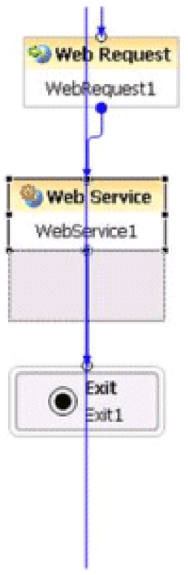
Composer routing workflow and voice callflow diagrams follow a vertical layout scheme by default. The in port of a block is always positioned at the top of the block while one or more out ports are positioned at the bottom edge of the block. Exception ports are displayed on the left edge. Following this vertical layout can quickly exceed the available vertical screen space. The Outline view can then be used to determine which part of a large diagram is being displayed currently and to quickly navigate to a different part by clicking the outline view.

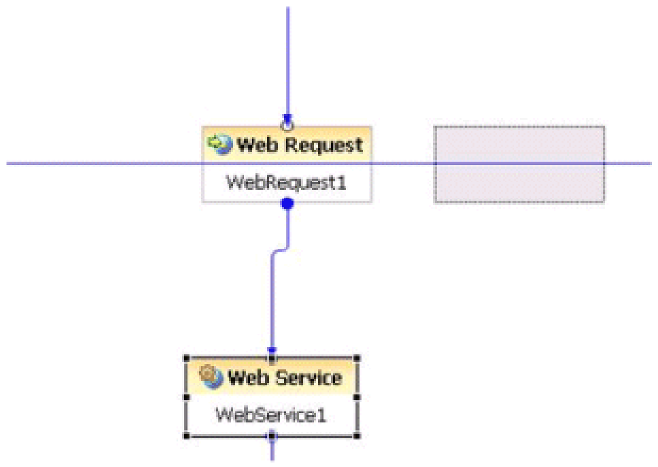
It is possible to follow a horizontal layout where the in ports and out ports can be manually re-positioned to any edge of the block and lose some features. For example, elbowed (bent) connectors and individual ports may not display on the block making it difficult to know how many unconnected ports are present and also to connect out ports out of order. See Show Connection Ports for more details. Please note that switching between the default vertical layout and the more flexible horizontal layout will rearrange connection links and manual rearrangement may be necessary. While working with diagrams, you may run into odd looking links. The figures below show some of these and lists suggestions on how to fix them.

	Link Issue	Steps to resolve	End Result
1.	 User Data UserData2 error.queue.submit Target Target1 Exit Exit1	Grab the extreme left and right points of the infinity curve and coax them towards the center until they merge into the vertical line.	 User Data UserData2 error.queue.submit Target Target1 Exit Exit1

2.	 Entry Entry1 Web Request WebRequest1 Web Service WebService1 Exit Exit1	Increase the vertical spacing between the two blocks linked by this connector until the broken connector fixes itself.	 Entry Entry1 Web Request WebRequest1 Web Service WebService1 Exit Exit1

To make it easier to align blocks, Composer diagrams have enabled "just in time" guides. They show up when a block is dragged near another block, when blocks are aligned, and help for about a second. To place the block in an aligned position, drop the block when the guides confirm the block is aligned.





Accessing the Editors and Templates

Composer editors are embedded/integrated within the user interface and are made available to you whenever a .scxml, .vxml, .ccxml, .grxml, or .jsp file is created or accessed within Composer.

Creating a New File

In **Composer** or **Composer Design** perspective, create a new VoiceXML, SCXML, or CallControlXML file as follows:

- Select **File** > **New** > **Other** > **Composer** > **Others**.
- Select the file type.
- Select the parent folder; usually an existing Project.
- Enter a name for the file.
- If applicable, click **Advanced** to link to the file system and use an existing file.
- Click **Finish**.

Using an Existing Template

- Select **File** > **New** > **Other** > **Composer** > **Others**.
- Select the file type.
- Select the parent folder; usually an existing Project.
- Enter a name for the file.
- If applicable, click **Advanced** to link to the file system and use an existing file.
- Click **Next**.
- Select the template.
- Click the **Use SCXML Template** checkbox.
  - Click **Finish**.

The editor opens with your new file. When working with XML files, the view contains **Source** and **Design** tabs. All editor functions described at the top of this topic are available to you. The appropriate Composer editor also opens whenever you open an existing .vxml, .ccxml, .grxml, .aspx, or .jsp file, whether previously created as described above, or previously imported into Composer.

Open an Existing File

Open an existing file as follows:

- Select **File** > **Open File**.
    - Navigate to the file to open, OR

Open a Composer Project's `src` or `src-gen` folder in the Project Explorer, then double-click the file to open it in the editor.

Creating a Custom Code Template

When writing manual SCXML/VXML/CCXML/GRXML code in the file editors, you may run into code that becomes repetitive. You may consider creating a code template to avoid retyping this block of code. Creating templates will improve the speed and consistency for writing code. The following steps show how to create a code template.

- Select **Window** > **Preferences**.
- In the Preferences dialog box, navigate through the Composer category, and expand the file type (VXML Files / CCXML Files / GRXML / SXCML Files) in which you want to add your template. Then select the **Templates** section. For example, select VXML Templates.
- Click the **New** button to add a new code template.
- Fill in the fields for the new template. The Context drop down box specifies at what context level you want the code template to appear as a context sensitive help.
- Click the **OK** button when finished.

XML File Preferences

You can also set XML File Preferences: **Window** > **Preferences** > **XML** > **XML Files**. When specifying Encoding formats in the XML Preference page: encoding formats are applicable only for new File creation using the Template option: (**File** > **New** > **XML** > **XML File** > **Create XML File from an XML Template** > **Select XML Template**). This applies only to new XML, VXML, CCXML and SCXML files. Existing files within the Project will not get impacted.

Creating a Backend JSP File

- Create a new JSP file by selecting **File** > **New** > **Backend JSP file**.
- In the **Create Backend JSP File** folder, navigate to the `src` folder within the Java Composer Project in which the Backend JSP file belongs.
- Type a name In the **File Name** field.

- Click **Finish**.

The Editor opens with a JSP file template. You can see your new file in the `src` folder of your Java Composer Project in the Project Explorer. A template is provided when you create a new Backend JSP file in Composer. You implement a performLogic method as a JSON object, store a result and return it to the voice application if desired. You have the flexibility to enter any valid JSP code that you wish.

Creating a Backend ASP .NET File

- Create a new ASP.NET file by selecting **File** > **New** > **Backend ASPX file**.
- In the ASPX File folder, navigate to the include folder within the .NET Composer Project in which the Backend ASPX file belongs.
- Type a name In the **File Name** field.
- Click **Finish**.

The Editor opens with an ASPX file template. You can see your new file in the include folder of your .NET Composer Project in the Project Explorer. A template is provided when you create a new Backend ASPX file in Composer. You implement a performLogic method as a JSON object, store a result and return it to the voice application if desired. You have the flexibility to enter any valid ASP.NET/C# code that you wish.

Keyboard Shortcuts

When working in Composer, you can use the following keyboard shortcuts. Click in the Package Explorer on the left. Then use the keyboard shortcuts shown below.

Ctrl+Alt+P	Create new interaction process diagram	Create Interaction Process Diagram Wizard opens
Ctrl+Alt+J	Create new Java Composer project	Wizard for Java Composer project opens
Ctrl+Alt+T	Create new .NET Composer project	wizard for .NET Composer project opens
Ctrl+Alt+O	Create a new voice callflow	Callflow Diagram wizard opens
Ctrl+Alt+R	Create a new routing workflow	Workflow Diagram wizard opens
Alt+I+C	Open dialog box for connecting to Configuration Server	Connect to Configuration Server dialog box opens
Alt+I+D	Disconnect from a connected Configuration Server	A connected Configuration Server is disconnected
Ctrl+Alt+C	Generate all	Brings up the Generate All wizard. Creates properly formatted VoiceXML (callflows) or SCXML (workflows) diagram files for the Project.
Alt+M	Open Prompt Manager perspective	Prompt Manager perspective opens
Alt+P+P	Open Project properties	Properties dialog box opens
Alt+H	Open Composer Help	Help menu appears. Select Help Contents.
Alt+H+C	Open Cheat Sheet	Help menu appears. Select Cheat sheets.
Alt+H+A	Open About Composer	About Composer dialog box opens
Ctrl+Alt+D	Open Database Connection Profiles	Database Connection Profiles opens
Ctrl+Alt+S	Open Statistic Builder	Statistic Builder opens.
Space	To toggle a check box	The check box mark toggles on/off
Alt+A	Jump to an Add button in a wizard	The Add button is selected
Alt+D	Jump to a Delete button in a wizard	The Delete button is selected
Alt+R	Jump to a Remove button in a wizard	The Remove button is selected

ALT+U	Jump to an UP button in a wizard	The UP button is selected
ALT+W	Jump to a DOWN button in a wizard	The DOWN button is selected
Alt+T	Jump to a Test/Preview button in a wizard	The Test/Preview button is selected
Alt+R/Alt+W	Jump to a Browse button in a wizard	The Browse Event button is selected
Alt+B	Jump to a Back button in a wizard	The Back button is selected
Alt+N	Jump to a Next button in a wizard	The Next button is selected
Alt+F	Jump to a Finish button in a wizard	The Finish button is selected

Composer Menus

This section discusses Composer's top-level menus.

File Menu

The commands active in the File menu change depending on the object you have selected, the perspective, and where you are within the perspective. Commands available from the File menu are described below. Also see the Hiding File Types topic.

New (Alt+Shift+N)	Select **New** > **Other**, which can be a new: • Java Composer Project • .NET Composer Project • Project • Callflow Diagram • Workflow Diagram • Grammar builder file • VoiceXML file • SCXML file • GrammarXML file • CallControlXML file • Backend JSP file • Folder • File You can also select Example... or Other... (for example, to create a new Interaction Process Diagram). Both of these bring up the Select a Wizard dialog box.
Open File	Opens the selected object.
Close (Ctrl+W)	Closes the current callflow or workflow diagram in the canvas.
Close All (Ctrl+Shift+W)	Closes all open elements in the workbench area.
Save (Ctrl+S)	Saves the selected object.
Save As	Saves the selected object under another name
Save All (Ctrl+Shift+S)	Saves all files in all open editors.
Revert	Reverts to an earlier saved version of a file

	selected from the History.
Move	Moves Project resources.
Rename	Renames Project resources.
Refresh	Reloads the configuration.
Convert Line Delimiters To	Converts line delimiters within the callflow design canvas to one of the following: • Windows (default) • Unix • MacOS 9
Print	Prints the selected object(s) within the callflow design canvas
Page Setup	Brings up a dialog box where you can specify to use workplace settings or diagram settings. You can also change orientation, units, size, and the margin as well as configure workplace settings.
Print Preview	Previews the output before printing.
Switch Workspace...	Browses for/selects a different workspace storage area. Changes the set of projects and resources that you are working on.
Restart	Restarts Composer.
Import	Brings up a wizard that leads you through the process of importing various types of files. Expand Composer to: Import an IRD Strategy or to import a Realtime Debugger Launch Configuration.
Export	Brings up a wizard that leads you through the process of exporting various types of files.
Properties	Shows properties for the selected resource (such as a Project). When a Project is selected, includes the Deployment property.
Exit	Exits Composer.

Edit Menu

Use the Edit menu to move around within the current application; cut, copy, paste, and delete blocks from the displayed callflow or workflow; find individual blocks within the callflow; and open the Properties dialog box for a selected block. Edit menu items include standard Windows and Eclipse edit functions:

Undo (Ctrl+Z)	After you perform an action on an object, the Undo command becomes Undo <action>. For example, Undo Deleting appears after you perform a deletion.
Redo	Select Redo <action> after using Undo <action> to go back to the most recent edit.
Cut	Removes selected object(s) and moves the objects to the clipboard.
Copy	Copies the selected object(s) to the clipboard.
Paste	Moves copies of selected object(s) from the clipboard to the selected location.
Delete	Deletes the selected object(s).
Select All	Selects all text or objects in the currently active view or editor.
Find/Replace	Use in text files, such as JSP, VXML, CCXML, and SCXML files. Place your cursor inside the file and then select from Edit menu. Not used for callflows or workflows. Brings up the Find/Replace dialog box.
Add Bookmark	When the cursor is positioned on a file in the Project Explorer, opens the Bookmark Properties window. A bookmark helps you quickly navigate to a frequently used resource. You can place an "anchor" either on a resource within the Workbench, or at a specific line within a file, by creating a bookmark. Then you can use the Bookmarks view to return to those files quickly. The Bookmarks view (Window > Show View > Bookmarks) displays all bookmarks that you have created.
Add Task	When a Project is selected in the Project Explorer, opens a properties dialog box. You can associate tasks with an editable resource, for instance to remind yourself to update a line of source code later.

Diagram Menu

This menu contains a number of standard diagram-related menu commands that can be used within the Project Explorer view and callflow/workflow diagram canvas.

Font	Invokes the system font dialog used to modify the font associated with the selected diagram element
Fill Color	Applies a color to the selected diagram element's interior
Line Color	Applies a color to the selected diagram element lines
Line Type	Modifies the style of the selected diagram connector element to one of the following: • solid • dash • dot • dash dot • dash dot dot
Line Width	Modifies the width of the selected diagram connector to one of the following: • one point • two points • three points • four points • five points
Arrow Type	Modifies either the source end or the target end of the arrow connector element to one of the following: • no arrow • solid arrow • open arrow
Line Style	Changes the diagram connector to one of the following: • Rectilinear Style Routing • Oblique Style Routing

	• Tree Style Routing
Select	Select all diagram elements, all shapes, or all connectors
Arrange	Applies a layout to all diagram elements, or to the selected ones only
Align	Aligns all selected diagram elements to: the left, the right, the center, the top, the bottom, or the middle of the selection
Text Alignment	Aligns the text left, right, or center
Order	Re-orders the selected diagram elements to: the front, the back, forward once, or backward once
Auto Size	Resets the size of the selected diagram elements to the default size, usually just enough to see an embedded label within the shape
Make Same Size	Sets the size of the selected diagram elements to the size of the last selected element, either horizontally, vertically, or both
Filters	Does one of the following: • sort/filter Compartment items • show/hide Compartment items (all Compartments or named Compartments only). Compartment items refer to Composite attributes within your editor, which can optionally be collapsed or expanded.
View	Shows or to hides various diagram features: • ruler • grid • page breaks Controls the snap to grid behavior.
Zoom	Changes the diagram magnification to one of: • in • out • 100% • To Fit • To Width • To Height • To Selection
Apply Appearance Properties	Copies various appearance properties, such as fill color, of the first selected diagram element to the

	other selected ones
Generate Code (Alt+G)	Creates a properly-formatted VoiceXML file from a callflow diagram built with Composer. Static VXML pages (pure VXML code) are generated in the src-gen folder of the Composer Project. This selection is enabled when the Project is selected in the Explorer after a new edit. In the case of a routing workflow, check the Problems tab for errors and fix any problems. If code generation succeeds, click OK at the confirmation dialog box. The SCXML code is generated in the src-gen folder.
Import Custom Blocks	Allows you to import a custom block that was previously exported so the block can be shared across multiple users/installations of Composer.
Export Custom Blocks	Allows you to export a custom block so the block can be shared across multiple users/installations of Composer.
Validate (Alt+V)	Validates the diagram that is open for completeness and accuracy. This selection is enabled when the Project is selected in the Explorer after a new edit.

Navigate Menu

This menu allows you to locate and navigate through resources and other artifacts displayed in the Workbench. The Navigate menu differs from the Find/Replace command on the Edit menu. Instead of entering text to find, the Navigate menu uses directional commands. The Navigate menu contains the following items:

Go Into	Refocuses the active view so that the current selection is at the root. This allows web browser style navigation within hierarchies of artifacts.
Go To	Refocuses the active view to one of the following: • Back: Displays the hierarchy that was displayed immediately prior to the current display. For example, if you Go Into a resource, then the Back command in the resulting display returns the view to the same hierarchy from which you activated the Go Into command. This command is similar to the Back button in an HTML browser. • Forward: Displays the hierarchy that was displayed immediately after the current display. For example, if you've just selected the Back command, then selecting the Forward command in the resulting display returns the view to the same hierarchy from which you activated the Back command. This command is similar to the Forward button in an HTML browser. • Up one level: Displays the hierarchy of the parent of the current highest-level resource.
Show In (Alt+Shift+W)	Finds and selects the currently selected resource in another view. If an editor is active, these commands are used to select the resource currently being edited in another.
Next	Navigates to the next item in a list or table in the active view. For example, when the search results view is active, this navigates to the next search result.
Previous	Navigates to the previous item in a list or table in the active view. For example, when the search results view is active, this navigates to the previous search result.
Last Edit Location	Jumps to the last edit position
Back	Navigates to the previous resource that was viewed in an editor. Analogous to the Back button on a web browser.
Forward	Navigates to undo the effect of the previous Back

	command. Analogous to the Forward button on a web browser.

Search Menu

Search results are displayed in the Search view, which appears if not previously present. The Search menu contains the following items:

Search	Opens the Search dialog box, where you can perform file, text or Java searches. Java searches operate on the structure of the code. File searches operate on the files by name and/or text content. Java searches are faster, since there is an underlying indexing structure for the code structure. Text searches allow you to find matches inside comments and strings.
File	Opens the Search dialog box. If it is not already selected, select the File Search tab. In the Containing text field, type the search string. For a Java search, make sure that the File name patterns field is set to *.java. The Scope should be set to Workspace. Then click Search. Note: To find all files of a given file name pattern, leave the Containing Text field empty.
Text	After selecting text, searches a workspace, a project, a file, or a working set. Working sets group elements for display in views or for operations on a set of elements. They restrict the set of resources that are displayed. If a working set is selected in the navigator, only resources, children of resources, and parents of resources contained in the working set are shown.

Project Menu

The Project menu contains the following items:

Open Project	Opens the currently selected Project(s). The selected Project(s) must currently be closed for this command to be available.
Close Project	Closes the currently selected(s) Projects. Closing a Project will remove all of that Project's state from memory, but the contents on disk are left untouched.
Build All (Ctrl+B)	Performs an incremental build on all Projects in the Workbench. This command builds (compiles) all resources in the Workbench that are affected by any resource changes since the last incremental build. This command is only available if auto-build is turned off. Auto-build is turned off via the Build Automatically menu option or from the General > Workspace preference page.
Build Project	Performs an incremental build on the currently selected Project. This command builds (compiles) all resources in the Project that are affected by any resource changes since the last build. This command is only available if auto-build is turned off. Auto-build is turned off via the Build Automatically menu option or from the General > Workspace preference page.
Build Working Set	Performs an incremental build on a working set. This command builds (compiles) all resources in the working set that are affected by any resource changes since the last build. This command is only available if auto-build is turned off. Auto-build is turned off via the Build Automatically menu option or from the General > Workspace preference page.
Clean	Discards all previous build results. If autobuild is on, then this invokes a full build.
Build Automatically	Toggles the auto build preference on and off. The auto-build preference is also located on the General > Workspace preference page.
Convert to a Dynamic Web project	[Not supported in Composer]
Properties	Opens a dialog box showing the properties of the selected project or of the Project that contains the selected resource.

Run Menu

- See the Debugging voice applications and Debugging routing applications topics for supported functionality.

The Run Menu contains all of the actions required to run, debug, step through code and work with breakpoints. Different parts of the menu are visible at different times, as each perspective can be customized to show only specific capabilities. The Run menu contains the following items:

Resume	Resumes execution of the currently selected Debug target.
Suspend	Halts the execution of the currently selected thread in a debug target. Once the selected thread is suspended, you can then examine it.
Terminate	Terminates the selected debug target.
Step Into	[Disabled for GVP Debugger]
Step Over	Steps over the highlighted statement. Execution will continue at the next line either in the same method or (if you are at the end of a method) it will continue in the method from which the current method was called. The cursor jumps to the declaration of the method and selects this line.
Step Return	[Disabled for GVP Debugger]
Run to Line	[Disabled for GVP Debugger]
Use Step Filters (Shift+F5)	Toggles step filters on and off. When on, all step functions apply step filters.
Run	Re-launches the most recently launched application, or launches the selected resource or active editor depending on the launch operation preference settings found on the Run/Debug > Launching preference page.
Debug	Re-launches the most recently launched application under debugger control, or launches the selected resource or active editor depending on the launch operation preference settings found on the Run/Debug > Launching preference page.
Run History	Displays a submenu of the recent history of launch configurations launched in run mode
Run As	When a callflow is selected, displays Run Callflow. In the Run mode, call traces are provided and the application continues without any breakpoints. Note: Run on Server is an Eclipse feature and is not used by Composer.
Run Configurations	Used for debugging callflow diagrams. Opens the

	Run Configurations dialog box that lets you create, manage, and run launch configurations of different types.
Debug History	Displays a submenu of the recent history of launch configurations launched in debug mode.
Debug As	Displays a sub menu of registered debug launch shortcuts. Launch shortcuts provide support for workbench or active editor selection sensitive launching. Note: Debug on Server is an Eclipse feature and is not used by Composer.
Debug Configurations	Used for debugging callflow diagrams. Opens the Debug Configurations dialog box that lets you create and modify launch configurations and debug applications.
External Tools	Displays external tools that allow you to configure and run programs, batch files, Ant buildfiles, and others using the Workbench. You can save these external tool configurations and run them at a later time. Output from external tools is displayed in the console view. Selecting External Tools presents the following sub-menus: Run As, External Tool Configurations, Organize Favorites.
Create URL Breakpoint	Creates a breakpoint, which suspends the execution of a workflow at the location where the breakpoint is set.
Toggle Breakpoint	Appears in Debugging perspective. Select to suspend the execution of a program at a particular location in a callflow. When a breakpoint is encountered during execution of a program, the program suspends and triggers a SUSPEND debug event with BREAKPOINT as the reason.
Toggle Line Breakpoint	Select to set a breakpoint on an executable line of a program.
Toggle Method Breakpoint	Use when working with types that have no source code (binary types). Open the class in the Outline View, and select the method where you want to add a method breakpoint. Select Toggle Method Breakpoint to have a breakpoint appear in the Breakpoints View. If source exists for the class, then a breakpoint also appears in the marker bar in the file's editor for the method that was selected. While the breakpoint is enabled, thread execution suspends when the method is entered, before any line in the method is executed.
Toggle Watchpoint	Appears in GVP Debugging perspective. You must select a Java field object to use this command. Use after you have created a watchpoint on the currently selected field. Whenever that field is accessed or modified, execution will be suspended. If the selected field already has a watchpoint, selecting this command will remove it.

Skip All Breakpoints	Select to mark all breakpoints in the current view as skipped. Breakpoints marked as skipped will not suspend execution.
Remove All Breakpoints	Select to remove all breakpoints from the Breakpoints View.

Configuration Server Menu

URS applications may be developed either:

- With a connection to Configuration Server
- Or in an offline mode, without connecting to Configuration Server

Connect	Select to connect to Configuration Server.
Disconnect	Select to disconnect from Configuration Server

Window Menu

The Window menu allows you to display, hide, and otherwise manipulate the various views, perspectives, and actions in the Workbench. The Window menu contains the following items:

New Window	Opens a new workbench window with the same perspective as the current perspective. The Composer perspective is the default for building your application.
New Editor	Opens an editor based on the currently active editor. It will have the same editor type and input as the original.
Open Perspective	Opens a new perspective in this workbench window
Show View	Displays the selected view in the current perspective. Views support editors and provide alternative presentations as well as ways to navigate the information in your workbench. For example, the Project Explorer and other navigation views display projects and other resources that you are working with. You can configure how views are opened on the Window > Preferences > General > Perspectives preference page.
Customize Perspective	Opens the Customize Perspective dialog box. The Shortcuts tab lets you select shortcuts you want added as cascade items to submenus. The Commands tab lets you select command groups that you want added to the current perspective.
Save Perspective As	Saves the current perspective thereby creating your own custom perspective. You can open more perspectives of this type using the Window > Open Perspective > Other menu item once you have saved a perspective.
Reset Perspective	Changes the layout of the current perspective to its original configuration
Close Perspective	Closes the active perspective
Close All Perspectives	Closes all open perspectives in the workbench window
Navigation	Displays the following submenu and shortcut keys: • Show System Menu (Alt+-): Shows the menu that is used for resizing, closing or pinning the current view or editor • Show View Menu (Ctrl+F10): Shows the drop down menu that is available in the toolbar of the active view • Quick Access (Ctrl+3): Shows a listing of available quick access categories

	• Maximize active view or editor (Ctrl+M): Causes the active part to take up the entire screen, or if it already is, returns it to its previous state • Minimize active view or editor: Causes the active part to be minimized. • Activate Editor (F12): Makes the current editor active • Next Editor (Ctrl+F6): Activates the next open editor in the list of most recently used editors • Previous Editor (Ctrl+Shift+F6): Activates the previous open editor in the list of most recently used editors • Switch to editor (Ctrl+Shift+E): Shows a dialog that allows switching to opened editors. Shows a dialog that allows switching to opened editors. • Next View: Activates the next open view in the list of most recently used views • Previous View (Ctrl+F7): Activates the previous open view in the list of most recently used editors • Next Perspective (Ctrl+F8): Activates the next open perspective in the list of most recently used perspectives • Previous Perspective (Ctrl+Shift+F8): Activates the previous open perspective in the list of most recently used perspectives
Preferences	Opens a dialog box for indicating your preferences for using the workbench. There are a wide variety of preferences for configuring the appearance of the workbench and its views, and for customizing the behavior of all tools that are installed in the workbench.

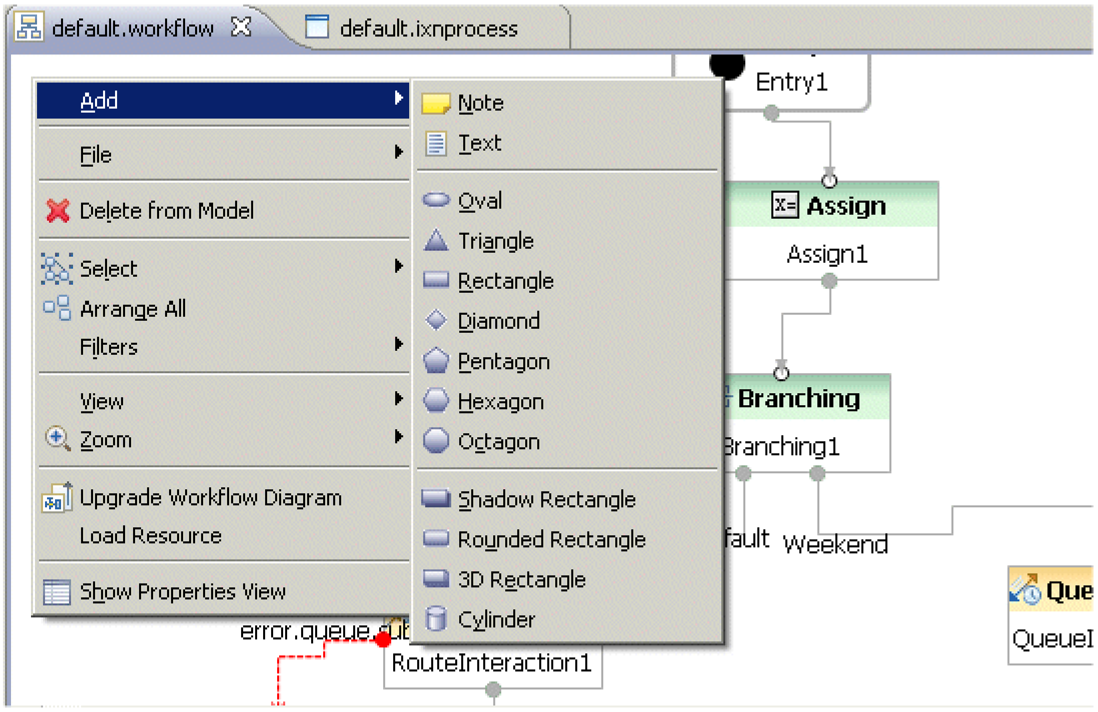
Help Menu

The Help menu contains the following items:

Welcome	Displays a welcome screen.
Help Contents	Displays the Eclipse help system. Note: The Composer Help, which introduces the Composer Help wiki, is integrated as a workbook within the overall Eclipse Help system.
Search	Opens a help pane where you can enter a search expression and view results.
Dynamic Help	Opens a help pane to show context-sensitive help.
Key Assist (Ctrl+Shift+L)	Opens a help pane with a listing of keyboard shortcuts.
Tips and Tricks	Opens a Tips and Tricks dialog box with a variety of topics:
Cheat Sheets	Opens the Cheat Sheet Selection dialog box with several available Cheat Sheets that lead you through key tasks.
Check for Updates	Currently not used by Composer.
Install New Software	Opens the Install dialog box where you can select or enter a site that has the software you want to install. As described in the Composer 8.1 Deployment Guide, use this menu item to install later versions of Composer. Use the About Eclipse SDK menu item to uninstall the current version of Composer prior to updating to a later version. For another usage example, see the Integrating with Source Control Systems topic, Subversion section.
About Composer	Opens the About Composer dialog box, which displays version, licensing, and Eclipse links. It also contains buttons to access Feature Details, Plug-in Details, and Configuration Details.

Canvas Shortcut Menu

When creating a callflow or workflow in Composer or Composer Design perspective, a shortcut menu opens when you right-click inside the canvas area. The figure below shows the menu when creating a workflow.



Canvas Menu

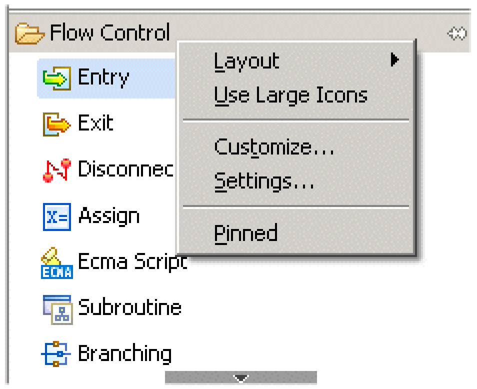
The Canvas menu contains the following items:

Add	Allows you to add a note, text, or one of the shapes shown in the figure above. When creating note objects in a diagram there are two ways to create them. After selecting the note tool, you can either click a single point or drag a box to indicate initial size. In the former case, the note will continue to grow horizontally as text is entered. With the latter case, text will automatically wrap text using the input width.
File	Gives the option of printing the diagram or saving it as an image file. Selecting Save as Image opens a dialog box giving the option to save in one of the following formats: GIF, BMP, JPEG, SVG, PNG,

	or PDF. You can also select Export to HTML.
Delete from Model	Deletes the selected block from the workflow.
Select	Allows you to select: • All • All Shapes • All Connectors
Arrange All	Use to arrange blocks and connectors in a callflow/workflow in a more orderly fashion. If you don't like the result, select Undo Arrange All from the Edit menu.
Filters	Allows you to show/hide connector labels.
View	Use to view a grid, snap to a grid, view rulers, view page breaks, and re-calculate page breaks.
Zoom	Use to: • Zoom In • Zoom Out • Zoom 100% • Zoom to Fit • Fit to Width • Fit to Height • Fit to Selection
Upgrade Workflow Diagram or Upgrade Callflow Diagram	Use to upgrade a previously created diagram to the current version of Composer.
Load Resource	Allows you to browse for/load Resource URIs.
Show Properties View	Shows the Properties view for the selected block or diagram.

Palette Group Menu

When creating a callflow or workflow in Composer or Composer Design perspective, a shortcut menu opens when you right-click on a palette title bar. The figure below shows an example:



The Palette Group menu contains the following items:

Layout	Allows you to specify how the blocks in this palette group should be displayed: • Columns • List • Icons Only • Detail
Use Large Icons	Allows you to increase the size of the icons representing the callflow or workflow blocks.
Customize	Opens a dialog box where you can change block names and descriptions, hide/unhide blocks from the palette, configure the block drawer to open upon Composer startup, and pin the block drawer open upon Composer startup.
Settings	Opens a dialog box where you can change the font, layout, and palette drawer options.
Pinned	Allows you to prevent a block drawer from closing when you switch to a different palette group.

Composer Toolbars

This section discusses Composer's toolbars.

Toolbars Overview

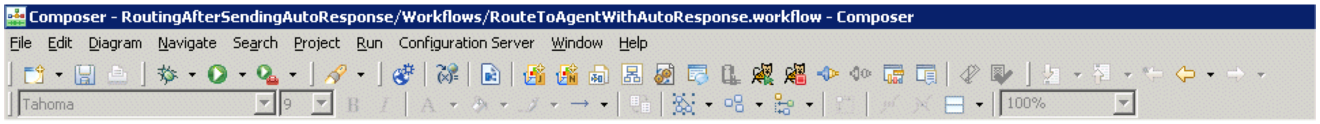
Composer has a number of toolbars for various purposes:

- Main Toolbar
- View Toolbars
- Perspective Switcher Toolbar
- Trim Stack Toolbar
- Debugging Toolbars
- Minimizing and Restoring Views

Note: To see a tooltip containing the name of a toolbar button (icon), hover the cursor over the button.

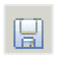
Main Toolbar

The main toolbar, sometimes called the workbench toolbar, is displayed at the top of the Workbench window directly beneath the menu bar. Note: Buttons on the main toolbar change based on the active perspective. Items in the toolbar might also be enabled or disabled based on the state of either the active view or editor. Sections of the main toolbar can be rearranged using the mouse. The figure below shows available toolbar buttons in the Composer perspective when a callflow diagram is selected.


Composer - RoutingAfterSendingAutoResponse/Workflows/RouteToAgentWithAutoResponse.workflow - Composer
File Edit Diagram Navigate Search Project Run Configuration Server Window Help
Tahoma
9
100%

Toolbar Buttons

The table below identifies buttons that can appear on the toolbar.

	New Select to create one of the following new resources: Java Composer Project (includes callflows and workflows), .NET Composer Project, Project..., Grammar builder file, VoiceXML file, GrammarXML file, CallControlXML file, Backend JSP file, SCXML file, or Folder, or File. You can also select Example or Other. Note: Before you can create a new file, you must create a project in which to store the file.
	**Save** Saves the content of the active editor.
	Print Prints the contents of the active editor.
	Debug Re-launches the most recently launched application under debugger control, or launches the selected resource or active editor depending on the launch operation preference settings found on the Run/Debug > Launching preference page. Used for voice applications.
	Run Re-launches the most recently launched application, or launches the selected resource or active editor depending on the launch operation settings found on the Run/Debug > Launching preference page. Click the down arrow to select Run As or Run Configurations. You can also organize favorites.

	Run Last Tool Allows you to quickly repeat the most recent launch in run mode or quickly run the selected resource, if that mode is supported (based on your current launch settings). Click the down arrow to select Run As or bring up the External Tool Configurations dialog box.
	Search Brings up a Search dialog box where you can perform one of the following types of searches: File, Java, Java Script. The General > Search preference page allows you to set preferences for searches.
	Launch GAX Server portal Launches the Genesys Administrator Extension used by the GAX Server (see GAX Server OPM Block). Composer uses the host, port, username, and password used on the GAX Server Preferences page to fetch ARM parameters or audio resource IDs list.
ButProjVar.gif	**Access Project Variables** Opens a dialog box where you can set or delete application variables. The appearance of this button changes depending on what type of diagram you are working with. When working with a callflow or workflow, the button appears as shown on the top left. When working with an interaction process diagram, the button appears as shown on the bottom left.
	Show Properties View Shows the properties of the selected diagram.
	Create Java Composer Project Brings up a wizard dialog box for creating a new Java Composer Project.
ButNewNet.gif	**Create .NET Composer Project** Brings up a wizard dialog box for creating a new .NET Composer Project.
	Create New Callflow Brings up a wizard dialog box for creating a main callflow diagram or a sub-callflow diagram.
	Create New Workflow Brings up a wizard dialog box for creating a main workflow diagram or sub-workflow diagram.
	Create New Interaction Process Brings up a wizard dialog box for creating an interaction process diagram.
	Open the Prompts Manager View

	Displays the Prompts Manager view in the lower center pane of the Composer main window.
	Open Database Connection Properties Opens the Connection Profiles tab where you can define a database connection profile and test the connection. This button becomes enabled when you select the connection.properties file in the Project db folder.
	Generate All Opens the Generate all dialog box, which lets you create properly formatted VoiceXML or SCXML files for all callflow and/or workflows in the Project.
	Start Tomcat Starts the Tomcat web server, which can be used for testing and deployment. If Tomcat has already started, displays a message to this effect.
	Stop Tomcat Stops the Tomcat web server.
	Connect to Configuration Server Opens a dialog box where you can connect to Configuration Server. Used for routing applications.
	Disconnect from Configuration Server Disconnects from Configuration Server.
	Statistics Manager Opens the Statistics Manager view for working with Universal Routing Server predefined statistics. Used for routing applications.
	List Objects Manager Opens the List Object Manager view, which allows you to create List Objects in Configuration Server. Use for creating parameterized applications. This provides System Administrators with the control to configure and change values from inside Configuration Server. Used for routing applications.
	Publish active interaction process diagram to Configuration Server If an interaction process diagram is selected, this toolbar button appears.
	Generate Code Creates a properly-formatted VoiceXML file from a callflow diagram or a SCXML file from a workflow diagram. Static pages (pure VXML or SCXML code) are generated in the src-gen folder

	of the Composer Project.
	Validate Checks your diagram files and other source files for completeness and accuracy. In the case of errors, the Problems view becomes visible and error markers are put on the blocks that contain errors. Double clicking on an error in the Problems view will take you to the corresponding blocks that contain the errors. Review each of the errors and do the fixes, then validate again.
	Next Annotation Selects the next annotation. Supported in the Java editor.
	Previous Annotation Selects the previous annotation. Supported in the Java editor.
	Last Edit Location Reveals the location where the last edit occurred.
	Back To Reveals the previous editor location in the location history.
	Forward To Reveals the next editor location in the location history.
	Turn Grammar Constraints Off When editing an XML file that has a set of constraints or rules defined by a DTD or an XML schema, you can turn the constraints on and off to provide flexibility in the way you edit, but still maintain the validity of the document periodically. When the constraints are turned on, and you are working in the Design view, the XML editor prevents you from inserting elements, attributes, or attribute values not permitted by the rules of the XML schema or DTD, and from removing necessary or predefined sets of tags and values.
	Reload Dependencies If you make changes to a DTD file or XML schema associated with an XML file (that is currently open), click to update the XML file with these changes. The changes will be reflected in the guided editing mechanisms available in the editor, such as content assist.
	Expand All Select to expand all of the items in the Breakpoints view.
	Collapse All Select to collapse all of the current elements in the view.
Tahoma	**Font Style**

	Allows you to change the font style of the selected text.
9 ▾	**Font Size** Allows you to change the font size of the selected text.
B	**Bold Font Style** Allows you to bold the selected text.
I	**Italic Font Style** Allows you to change the selected text to italics.
A ▾	**Font Color** Allows you to change the font color of the select text.
▾	**Fill Color** Allows you to change the fill color of the selected object.
▾	**Line Color** Allows you to change the color of the selected line.
→ ▾	**Line Style** Allows you to change the style of the selected line.
	Apply Appearance Properties Allows you to apply the applicable appearance properties of the first application shape to the other selected shapes.
▾	**Select All** Selects all objects in the diagram.
▾	**Arrange All** Arranges all or only the selected objects in the diagram.
▾	**Align** Aligns the selected objects in the callflow diagram: left, right, center, top, middle, bottom.
	Auto Size Allows you to change the size of the selected object.
	All Connector Labels Shows labels for all connector lines in the diagram.
	No Connector Labels

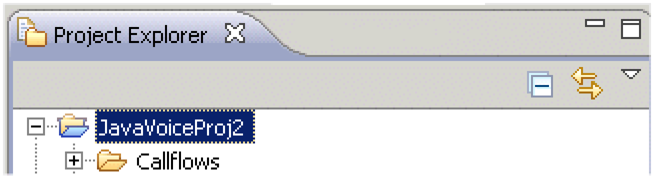
	Hides labels for all connector lines in the diagram.
⊟ ▾	**Show/Hide Compartment** Shows or hides composite attributes within an editor, which can optionally be collapsed or expanded.
75% ▾	**Magnification** Allows you to zoom and out of the current view, as well as to change the magnification from 5% to 400%. You can also fit to height, width, or selection.

View Toolbars

The title bar of a view contains a toolbar. This topic describes the following view toolbars:

Project Explorer

The Project Explorer toolbar is shown below.



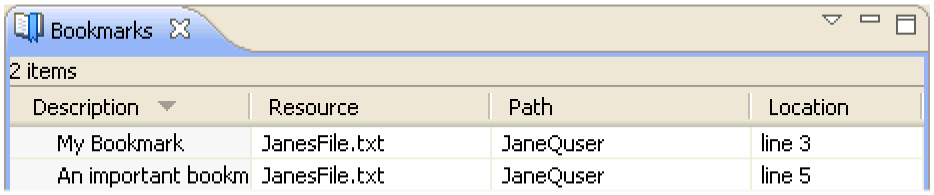
Each toolbar button is identified in the table below.

⊟	Collapse All Select to collapse all of the current elements in the view.
⇆	Link Open Editors When you have multiple files open for editing, select to bring an open file to the foreground (make its editor session the active editor) every time you select that open file in one of the navigation views.
▽	View Menu Select to show additional actions for this view. • Top Level Elements. Select from Projects or Working Sets (see below). • Folder Presentation. Select from Flat or Hierarchical. • Working Set. Select from Window Working Sets, No Working Sets, Selected Working Sets. Working sets group elements for display in views or for operations on a set of elements. The navigation views use working sets to restrict the set of resources that are displayed. If a working set is selected in the navigator, only resources, children of resources, and parents of resources contained in the working set are shown. • Deselect Working Set. Deselects the active working sets. All elements are shown after

	invoking this action • Edit Active Working Set. Opens the Edit Working Set wizard to edit the currently active working set. • Package Presentation. Select from Flat or Hierarchical. • Customize View. Allows you to filter the Project Explorer view to hide projects, folders, or files that you do not want to see. • Link Editor. Brings an open file to the foreground (makes its editor session the active editor) every time you select that open file in one of the navigation views.

Bookmarks View

The Bookmarks view is shown below.



Bookmarks

2 items

Description	Resource	Path	Location
My Bookmark	JanesFile.txt	JaneQuser	line 3
An important bookm	JanesFile.txt	JaneQuser	line 5

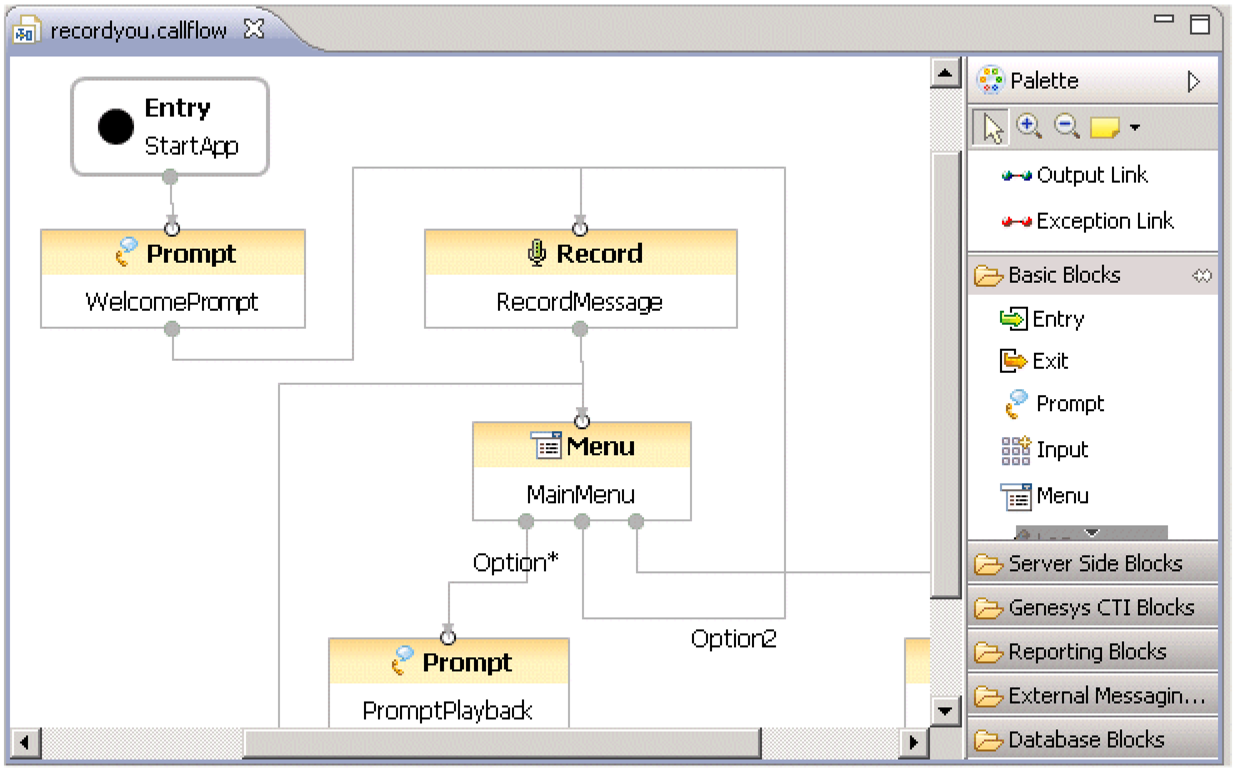
Each view button is identified in the table below.

▽	View Menu Select to show additional actions for this view. • Sort By: Select from Description, Resource, Path, Location, Ascending. • New Bookmarks View. • Configure Contents. Opens a window where you can filter the contents of the Bookmarks tab. • Columns. Opens a dialog box where you can set the width and move the following columns up and down: Description, Resource Path, and Location columns. • Preferences. Opens a dialog box where you can hide and show the following columns: Description, Resource, Path, Location, Creation Time, ID, Type.
▭	Minimize

	Minimizes the Bookmarks tab.
☐	Maximize Maximizes the Bookmarks tab.

Canvas View

The canvas is where you create callflows for your voice applications and workflows for your routing applications. The Canvas view toolbar is shown below in the upper-right.

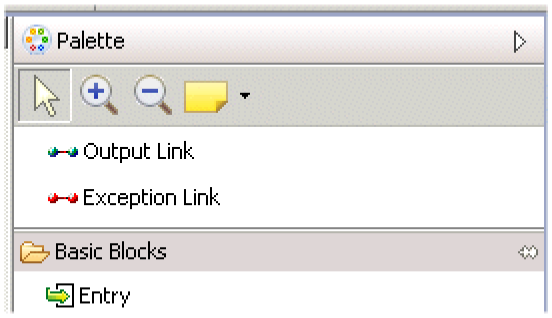


Each view button is identified in the table below.

▭	Minimize Minimizes the Canvas area.
☐	Maximize Maximizes the Canvas area.

Palette View

The Palette contains link tools as well as various types of blocks. To create callflow diagrams, the block categories are: Basic Blocks, Server Side Blocks, CTI Blocks, Reporting Blocks, External Message Blocks, Database Blocks, and Context Services Blocks. To create workflow diagrams, the block categories are: Flow Control Blocks, Routing Blocks, Voice Treatment Blocks, Server Side Blocks, eService Blocks, and Context Services Blocks. The Palette view toolbar is shown below.

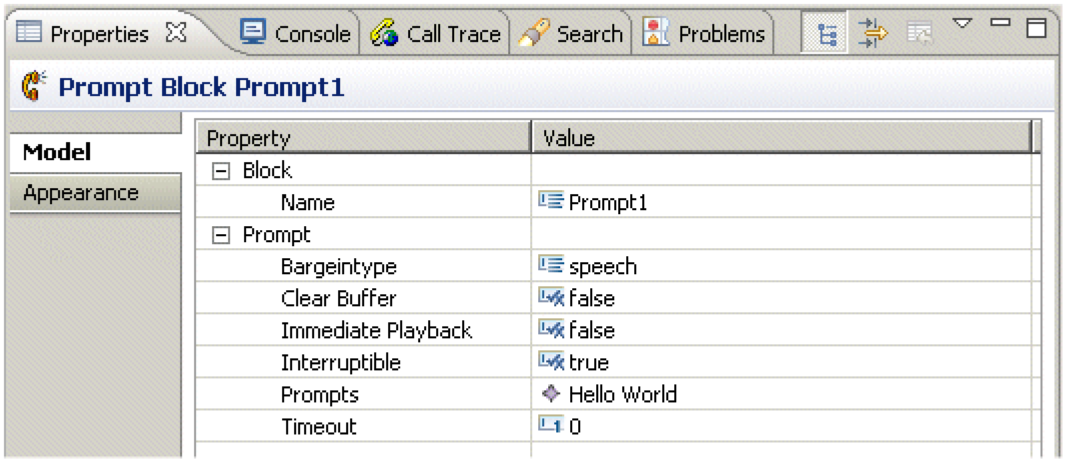


Each toolbar button is identified in the table below.

	Select Use to select a block for a callflow or workflow.
	Zoom In Click left to zoom in, Shift + left click to zoom out, drag to zoom to selection.
	Zoom Out Click left to zoom out, Shift + left click to zoom in.
	Create Note Click to create a note, text document, or note attachment. When creating note objects in a diagram there are two ways to create them. After selecting the note tool, you can either click a single point or drag a box to indicate initial size. In the former case, the note will continue to grow horizontally as text is entered. With the latter case, text will automatically wrap text using the input width.

Properties View

The Properties view shows the properties for a selected block and allows you to set/modify them. An example Properties view and toolbar is shown below.



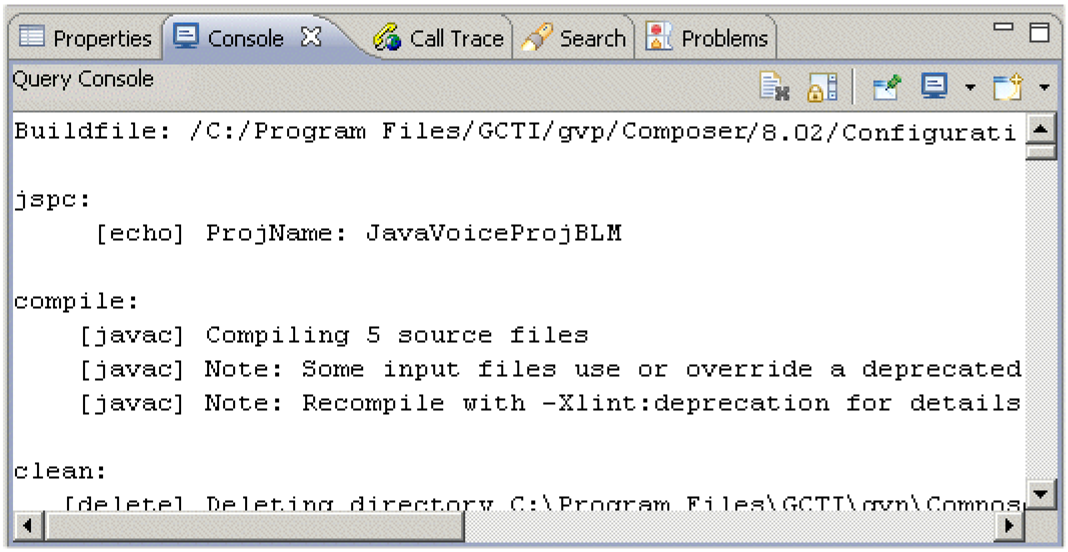
Prompt Block Prompt1

Property	Value
⊟ Block	
Name	Prompt1
⊟ Prompt	
Bargeintype	speech
Clear Buffer	false
Immediate Playback	false
Interruptible	true
Prompts	Hello World
Timeout	0

Each toolbar button is identified in the table below.

	Show Categories If enabled, method, field and type labels contain the categories specified in their block properties
	Show Advanced Properties If enabled, the Properties view shows advanced properties.
	Restore Default Value Use after changing a value in the Properties view to revert back to the default value.
▽	View Menu Select to show additional actions for this view: Show Categories, Show Advanced Properties, and Columns.
▭	Minimize Minimizes the Properties tab.
☐	Maximize Maximizes the Properties tab.

You can change settings for consoles on the Window Preferences Run/Debug Console page. An example Query Console view is shown below.



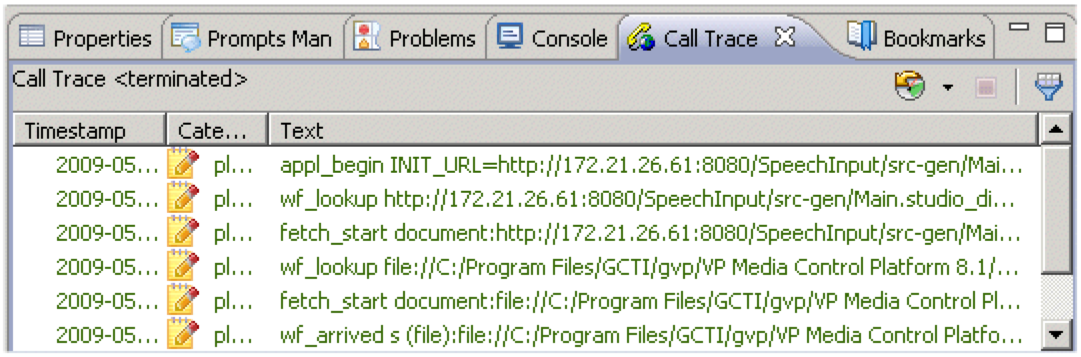
```
Buildfile: /C:/Program Files/GCTI/gvp/Composer/8.02/Configurati

jspc:
     [echo] ProjName: JavaVoiceProjBLM

compile:
    [javac] Compiling 5 source files
    [javac] Note: Some input files use or override a deprecated
    [javac] Note: Recompile with -Xlint:deprecation for details

clean:
   [delete] Deleting directory C:\Program Files\GCTI\gvp\Compos
```

Each toolbar button is identified in the table below.

	Clear Console Clears the currently active console.
	Scroll Lock Changes if scroll lock should be enabled or not in the current console.
	Pin Console Pins the current console to remain on top of all other consoles.
	Display Selected Console Opens a listing of current consoles and allows you to select which one you would like to see.
	Open Console Opens a new console of the selected type.

Call Trace View

The Call Trace view displays metrics which describe the events occurring in the application, such as recognition events, audio playback, user input, errors and warnings, and application output. An example Call Trace view and Toolbar are shown below.



Call Trace <terminated>

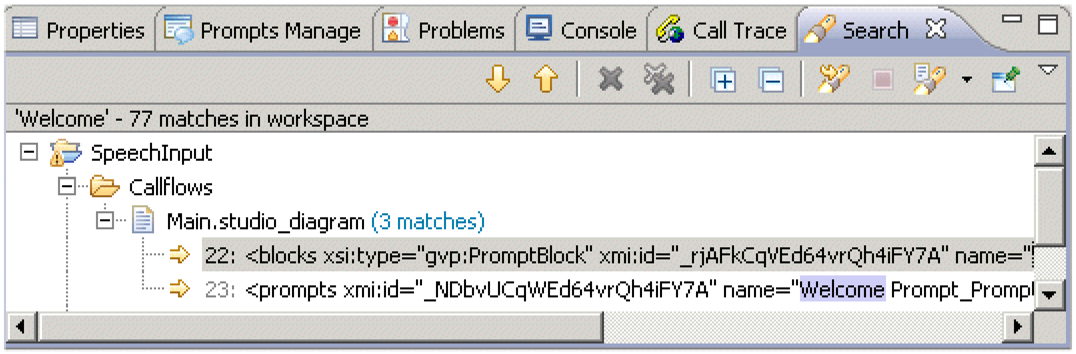
Timestamp	Cate...	Text
2009-05...	pl...	appl_begin INIT_URL=http://172.21.26.61:8080/SpeechInput/src-gen/Mai...
2009-05...	pl...	wf_lookup http://172.21.26.61:8080/SpeechInput/src-gen/Main.studio_di...
2009-05...	pl...	fetch_start document:http://172.21.26.61:8080/SpeechInput/src-gen/Mai...
2009-05...	pl...	wf_lookup file://C:/Program Files/GCTI/gvp/VP Media Control Platform 8.1/...
2009-05...	pl...	fetch_start document:file://C:/Program Files/GCTI/gvp/VP Media Control Pl...
2009-05...	pl...	wf_arrived s (file):file://C:/Program Files/GCTI/gvp/VP Media Control Platfo...

Each toolbar button is identified in the table below.

	Call Trace History Lists past calls. Once you select a past call, shows call trace history for that past call.
	Terminate Terminates the process that is associated with the current Process Console.
	Filter Metrics Brings up the Filter Metrics dialog box where you can select the following filters: Platform actions, User input, Application output, Document flow, Errors and warnings.

Search View

The search dialog lets you perform text string, File, Java, and JavaScript searches. When you first click the Search tab, there is a link to bring up the Search dialog box. The figure below shows the results of an example search and the toolbar.



'Welcome' - 77 matches in workspace

SpeechInput
Callflows
Main.studio_diagram (3 matches)
22: <blocks xsi:type="gvp:PromptBlock" xmi:id="_rjAFkCqVEd64vrQh4iFY7A" name="
23: <prompts xmi:id="_NDbvUCqWEd64vrQh4iFY7A" name="Welcome Prompt_Promp

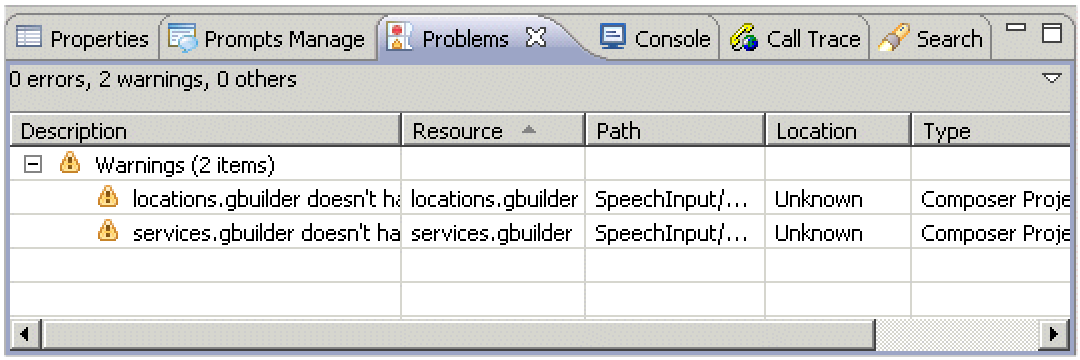
Each toolbar button is identified in the table below.

	Show Next Match Shows the next items that meets the search criteria.
	Show Previous Match Shows the previous item that met the search criteria.
	Remove Selected Matches Removes matched items that you have selected from the results .
	Remove All Matches Removes all matches from the results.
	Expand All Select to expand all of the current elements in the view.
	Collapse All Select to collapse all of the current elements in the view.
	Run Current Search Again Repeats the search with currently-defined parameters.
	Cancel Current Search Cancels the current search.
	Show Previous Searches Displays a list of previous searches.
	Pin the Search View Pins the current search view to remain on top of all other views.
	View Menu Select from the following: Show as List, Show as tree, Filters, Preferences.

As you work with resources in the workbench, various builders may automatically log problems, errors, or warnings in the Problems view. For example, when you save a Java source file that contains syntax errors, those will be logged in the Problems view. When you double-click the icon for a problem, error, or warning, the associated block is highlighted in the canvas area. Also see topics Diagram Validation and Validating a Single Flow Diagram.

Problems View

An example Problems view with toolbar is shown below.



0 errors, 2 warnings, 0 others

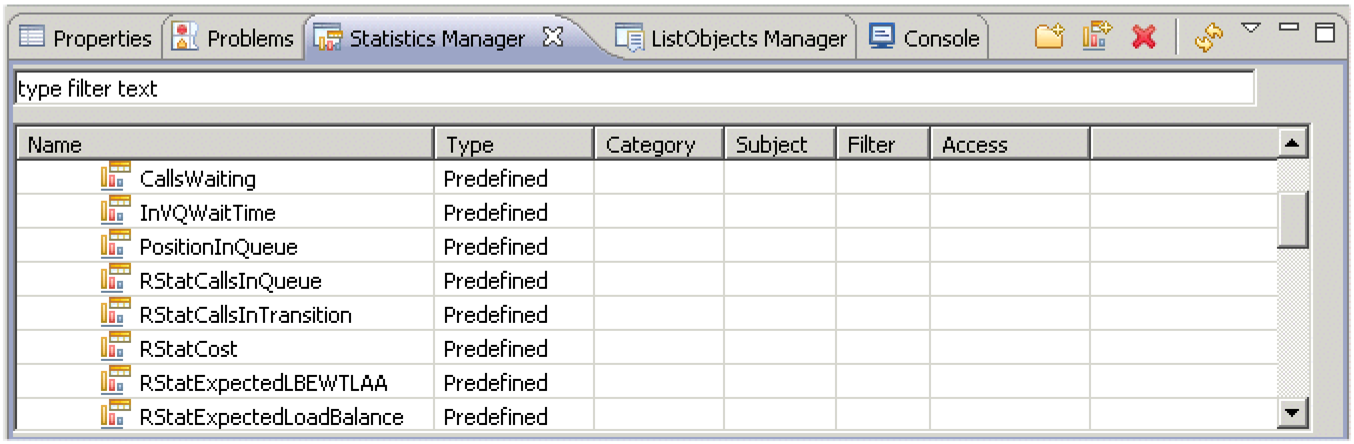
Description	Resource	Path	Location	Type
Warnings (2 items)				
locations.gbuilder doesn't ha	locations.gbuilder	SpeechInput/...	Unknown	Composer Proje
services.gbuilder doesn't ha	services.gbuilder	SpeechInput/...	Unknown	Composer Proje

Each toolbar button is identified in the table below.

▽	View Menu Select to show additional actions for this view. Show: All Errors, Warning on Selection, Show All. Group By: Java Problem Type, Type, JavaScript Problem Type, Severity, None. Sort by: Description, Resource, Path, Location, Type, Ascending New Problems View Configure Contents. Opens a window where you can filter the contents of the Problems tab. Columns. Opens a dialog box where you can set the width and move the following columns up and down: Description, Resource Path, and Location. Preferences. Opens a dialog box where you can hide and show the following columns: Description, Resource, Path, Location, Creation Time, ID, Type.
▭	Minimize Minimizes the Problems tab.
☐	Maximize Maximizes the Problems tab.

Statistics Manager View

The Statistics Manager view lets you easily create, delete, and organize created statistics into folders.



type filter text

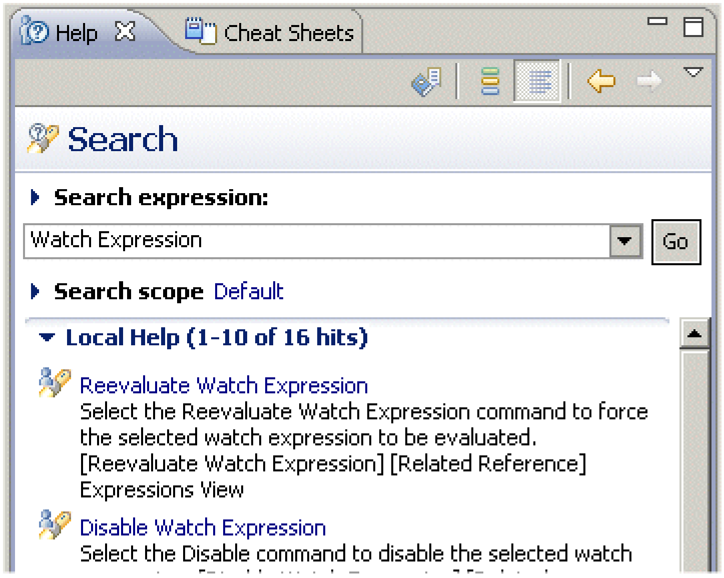
Name	Type	Category	Subject	Filter	Access
CallsWaiting	Predefined				
InVQWaitTime	Predefined				
PositionInQueue	Predefined				
RStatCallsInQueue	Predefined				
RStatCallsInTransition	Predefined				
RStatCost	Predefined				
RStatExpectedLBEWTLAA	Predefined				
RStatExpectedLoadBalance	Predefined				

Each toolbar button is identified in the table below.

📁	Add New Folder You have the option of creating folders to organize statistics that you create. Click this button to create a new folder.
📊	Add New Statistic To build a new statistic, select a folder and click this button to bring up Statistics Builder.
❌	Delete Selected Item To delete a statistic that you have created, select the statistic and click this button to delete.

Help View

The Help view shows the following toolbar after selecting Search from the Help menu.



Search

Search expression:

Watch Expression

Go

Search scope Default

Local Help (1-10 of 16 hits)

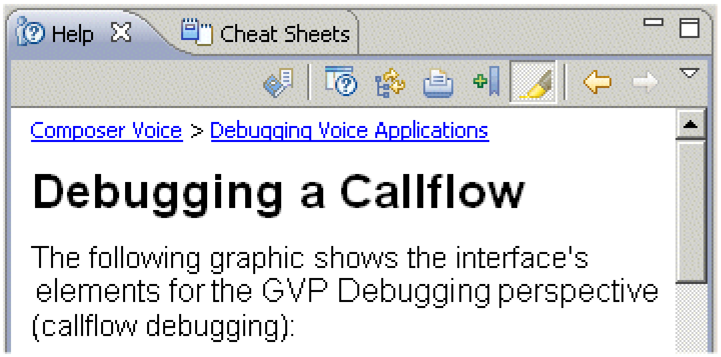
Reevaluate Watch Expression
Select the Reevaluate Watch Expression command to force the selected watch expression to be evaluated.
[Reevaluate Watch Expression] [Related Reference] Expressions View

Disable Watch Expression
Select the Disable command to disable the selected watch

Each toolbar button in the Help view is identified in the table below.

	Show All Topics Select to display all available Help topics.
☰	Show Result Categories Select to display the categories for the Help results.
☰	Show Result Descriptions Select to display the descriptions of the Help topics.
⇦	Back Move back through topics.
⇨	Forward Move forward to next topic.

Once you select a topic, the toolbar changes as shown below.



Composer Voice > Debugging Voice Applications

Debugging a Callflow

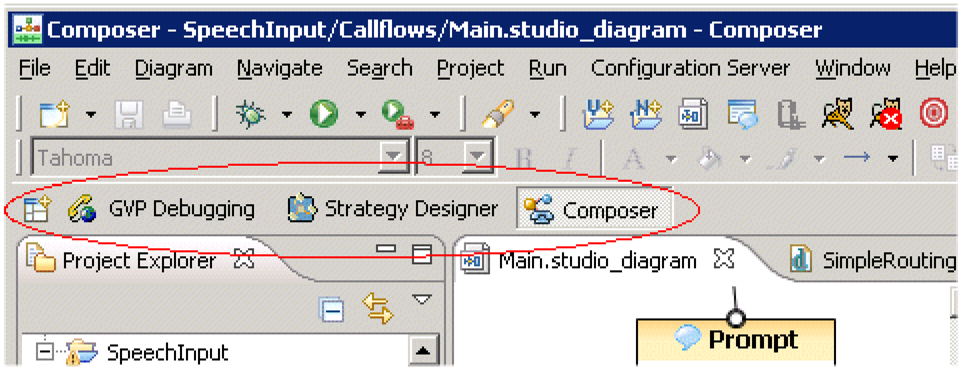
The following graphic shows the interface's elements for the GVP Debugging perspective (callflow debugging):

Each toolbar button is identified in the table below.

	Show All Topics Select to display all available Help topics.
	Show in External Window Select to display the results in an external window.
	Show in All Topics Select to display the results in all topics.
	Print Select to print the results/topic.
	Bookmark Select to bookmark the results/topic
	Highlight Search Term Select to highlight a search term.
	Back Move back through results.
	Forward Move forward to next result.

Perspective Switcher Toolbar

Perspectives are task-oriented layouts for organizing the views and windows in your workbench. The Perspective Switcher Toolbar allows quick access to perspectives that are currently open.


Composer - SpeechInput/Callflows/Main.studio_diagram - Composer
File Edit Diagram Navigate Search Project Run Configuration Server Window Help
Tahoma
GVP Debugging
Strategy Designer
Composer
Project Explorer
Main.studio_diagram
SimpleRouting
SpeechInput
Prompt

Open Perspective Button

An **Open Perspective** button (displaying all Eclipse perspectives) is located at the start of the Perspective Switcher toolbar. In the above figure, it is located in front of the GVP Debugging button.

Perspective Switcher Toolbar

The Perspective Switcher Toolbar is normally positioned below the main toolbar (top-left), but you can also position it vertically on the left-hand side of the workbench.

Shortcut Menu for Perspective Buttons

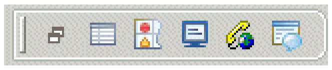
Right-clicking the button for an active perspective opens a shortcut menu. The first three entries in the table below do not appear if the perspective is not selected.

Customize	Opens the customize perspective dialog box.
Save As	Opens a dialog box for saving a customize perspective. Once saved, the customize perspective appears in the list that opens when you click the Open Perspective button.
Reset	Resets the changes you made to a perspective.
Close	Removes the button for the perspective.

Dock On	Allows you to dock the perspective button: Top Right, Top Left, or Left (left-hand side of work bench).
Show Text	Toggles between an icon and text on the perspective button.

Trimstack Toolar

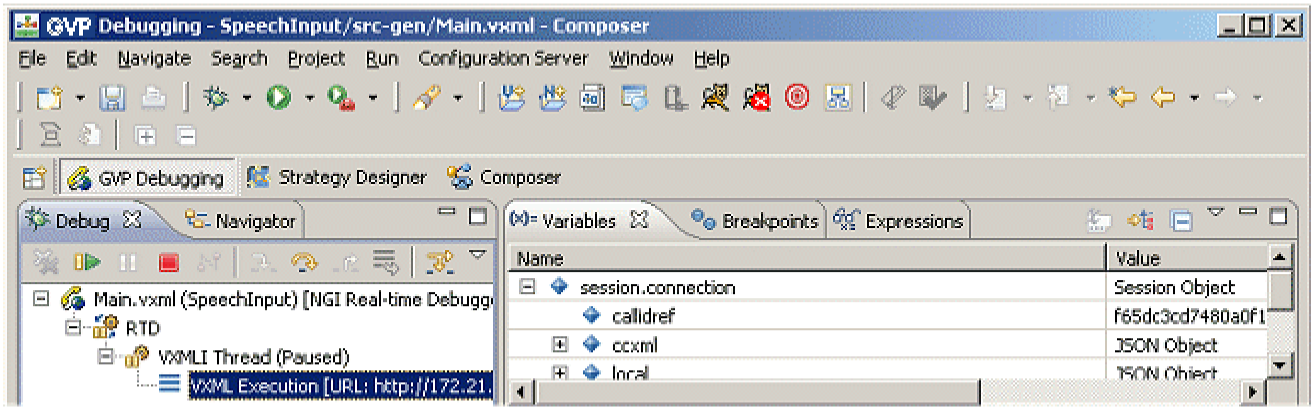
Minimizing a view stack will also produce a toolbar in the trim at the outer edge of the workbench window (a Trim Stack). This bar will contain a button for each of the views in the stack. Clicking on one of these icons will result in the view being displayed as an overlay onto the existing presentation. This is an example of a Trim Stack Toolbar containing buttons for Restore, Properties, Problems, Console, Call Trace, and Prompts Manager views:



The first button is Restore , which restores the normal view.

Debugging Toolbars

In GVP and ORS Debugging perspectives, the first pane contains Debug and Navigator views. The second pane contains views for Variables, Breakpoints, and Expressions. A GVP example is shown below.

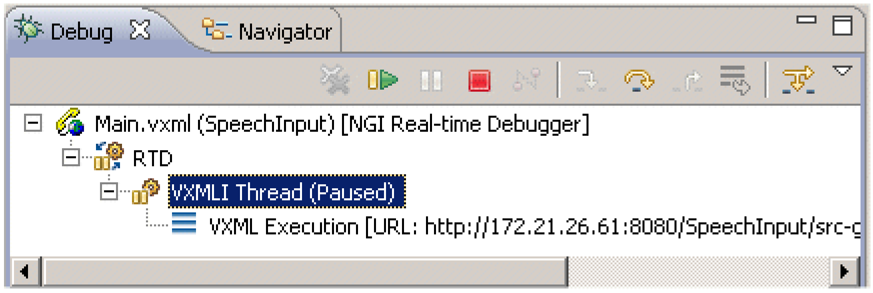


Main.vxml (SpeechInput) [NGI Real-time Debugg
RTD
VXMLI Thread (Paused)
VXML Execution [URL: http://172.21.

Name	Value
session.connection	Session Object
callidref	f65dc3cd7480a0f1
ccxml	JSON Object
local	JSON Object

Debug View

The Debug view shows the name of the callflow or workflow diagram being debugged, as well as the status of the debug progress or result.



Main.vxml (SpeechInput) [NGI Real-time Debugger]
RTD
VXMLI Thread (Paused)
VXML Execution [URL: http://172.21.26.61:8080/SpeechInput/src-g

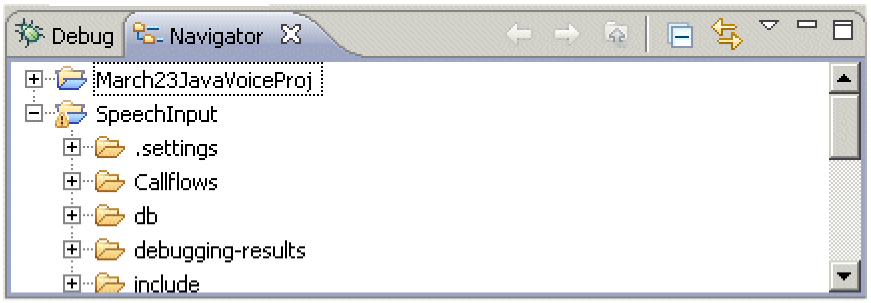
Each toolbar button is identified in the table below.

CVSearchRemAllMatches.gif	**Remove All Terminated Launches** Select to clear the Debug view of all terminated launches.
	Resume Select to resume the execution of the currently suspended debug target.
	Suspend

	[Not supported in Composer]
■	**Terminate** Select to terminate the launch associated with the selected debug target. Once a launch is terminated it can be automatically removed from the Debug view. When using the ORS Debugger, Terminate means that the session in ORS will end along with the debugging session.
	Disconnect Not supported for the GVP Debugger. When using the ORS Debugger, Disconnect means that the debugging session ends, but ORS will continue executing the SCXML.
	Step Into Disabled for both routing and voice applications.
	Step Over Step Over is the only way to step for both routing and voice applications. Select to step over the next method call (without entering it) at the currently executing line of code. Even though the method is never stepped into, the method will be executed normally.
	Step Return [Not supported in Composer]
	Drop to Frame [Not supported in Composer]
	Use Step Filters [Not supported in Composer]
▽	**View Menu** Select from the following: • View Management • Java (then select from: Show Monitors, Show System Threads, Show Qualified Names, Show Thread Groups)

Navigator View

The Navigator view shows the same Project folder structure shown in the Project Explorer window of the Composer perspective.



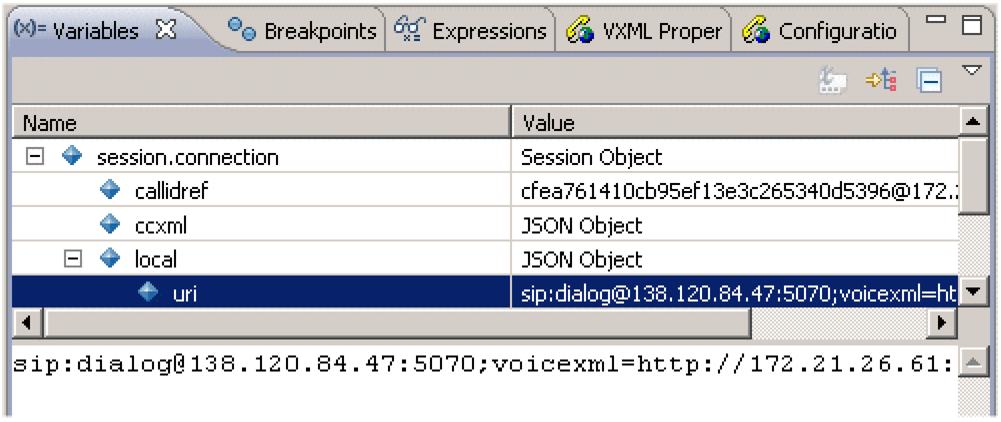
Each toolbar button on the Navigator toolbar is identified in the table below.

⇦	**Back** Moves back.
⇨	**Forward** Moves forward.
	Up Navigate up one level in the hierarchy
⊟	**Collapse All** Select to collapse all of the current elements in the view.
⇆	**Link Open Editors** When you have multiple files open for editing, select to bring an open file to the foreground (make its editor session the active editor) every time you select that open file in one of the navigation views.
▽	**View Menu** Select to show additional actions for this view. • Select Working Set. Working sets group elements for display in views or for operations on a set of elements. The navigation views use working sets to restrict the set of resources that are displayed. If a working set is selected in the navigator, only resources, children of resources, and parents of resources contained in the working set are shown. • Deselect Working Set. Deselects the active working sets. All elements are shown after invoking this action. • Edit Active Working Set. Opens the Edit Working Set wizard to edit the currently active working set. • Sort (by name or type).

	• Filters (class, JETEmitters, general, or *). • Link with Editor. Brings an open file to the foreground (makes its editor session the active editor) every time you select that open file in one of the navigation views.

Variables View

The Variables view displays information about the variables associated with the stack frame selected in the Debug view. When debugging a Java program, variables can be selected to have more detailed information as displayed below. In addition, Java objects can be expanded to show the fields that a variable contains.



Name	Value
session.connection	Session Object
callidref	cfea761410cb95ef13e3c265340d5396@172.:
ccxml	JSON Object
local	JSON Object
uri	sip:dialog@138.120.84.47:5070;voicexml=ht

```
sip:dialog@138.120.84.47:5070;voicexml=http://172.21.26.61:
```

Each toolbar button in the Variables view is identified in the table below.

	Show Type Names Select to change if type names should be shown in the view or not. Unavailable when columns are displayed. Hint: Select Layout from View menu and de-select Show Columns.
	Show Logical Structure Select to change if logical structures should be shown in the view or not.
	Collapse All Select to collapse all the currently expanded variables.
	View Menu Select from the following: • Layout: Vertical View Orientation, Horizontal

	View Orientation, Variables view Only, Show Columns, Select Columns. • Java: Show Constants, Show Static Variables, Show Qualified Names, Show Null Array Entries, Show References, Java Preferences.

Breakpoints View

The Breakpoints view and toolbar manage breakpoints within a debugging session.

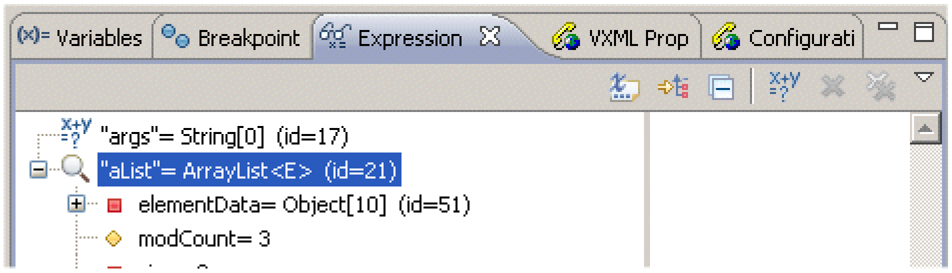
Each toolbar button in the Breakpoints view is identified in the table below.

✖	**Remove Selected Breakpoints** Select to clear all selected breakpoints.
✖	**Remove All Breakpoints** Select to clear all breakpoints.
⚙	**Show Breakpoints Supported by Selected Targets** Select to show all breakpoints supported by the selected targets.
➔	**Go To File For Breakpoint** [Not supported in Composer]
⊘	**Skip All Breakpoints** Select to skip over all breakpoints.
🗋	**Create URL Breakpoint** Select to create a breakpoint that uses a URL.
⊞	**Expand All** Select to expand all the current breakpoints.
⊟	**Collapse All** Select to collapse all the current breakpoints.
⇆	**Link With Debug View** [Not supported in Composer]
J!	**Add Java Exception Breakpoint**

	Select to open a dialog box where you can: • Type a string that is contained in the name of the exception you want to add. You can use wildcards as needed ("* " for any string and "? " for any character). • Select the exception types you want to add. • Select Caught and Uncaught as needed to indicate on which exception type you want to suspend the program. [This option is not relevant to GVP Debugging in Composer.]
▽	**View Menu** Select from the following: • Group By: Breakpoints, Breakpoint Types, Breakpoint Working Sets, Files, Projects, Resource Working Sets, Advanced... • Default Working Set • Deselect Default Working Set • Working Sets • Show Qualified Names

Expressions View

Use the Expressions view to inspect data from a stack frame of a suspended thread, and other places.



```
"args"= String[0]  (id=17)
"aList"= ArrayList<E>  (id=21)
    elementData= Object[10]  (id=51)
    modCount= 3
```

Each toolbar button in the Expressions view is identified in the table below.

	Show Type Names Select to change if type names should be shown in the view or not. Unavailable when columns are displayed. Hint: Select Layout from View menu and de-select Show Columns.

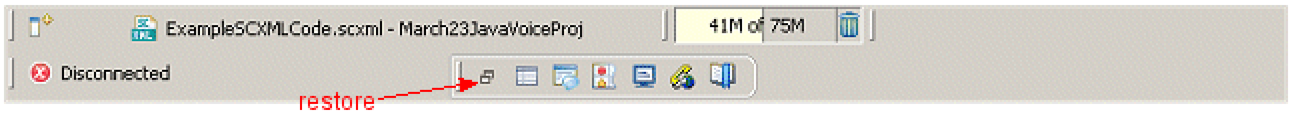
	Show Logical Structure Select to change if logical structures should be shown in the view or not.
⊟	**Collapse All** Select to collapse all the currently expanded expressions.
x+y =?	**Create a New Watch Expression** Select to open the Create New Expression dialog box, which allows you to create a new watch expression based on the selected variable and add it to the Expressions View.
✖	**Remove Selected Expressions** Select to remove the selected expressions.
✖	**Remove All Expressions** Select to remove all expressions.
▽	**View Menu** Select from the following: • Layout: Vertical View Orientation, Horizontal View Orientation, Expressions View Only. • Java: Show Constants, Show Static Variables, Show Qualified Names, Show Null Array Entries, Show References, Java Preferences.

Minimizing and Restoring Views

Panes in the Composer window contain various views. Each view has its own tab. To minimize a pane containing views:

- Click the button to minimize the pane. This causes the views to appear in a toolbar (trim stack toolbar). The toolbar appears in close proximity to where the pane was located.

The toolbar could be on the side or at the bottom of the Composer window depending on the selected perspective. For example, assume you are editing a file in Composer Design perspective and minimize the pane below, which contains Properties, Prompts Manager, Problems, Console, Call Trace, and Bookmark views. In this case, the minimization causes a toolbar to appear at the bottom of the Composer window. Depending on your screen, you may have to maximize the entire Composer window in order to see this toolbar.



The first button on the toolbar is used to restore all views. The remaining buttons represent the minimized views. To restore views:

- Click the button to restore all minimize views.
- Click a single view button to restore an individual view.

Voice Applications and Callflows

This section contains the following:

- Getting Started with Voice Applications
- Preferences for Voice Applications
- Creating Voice Applications for GVP
- Block Palette Reference

Getting Started with Voice Applications

This section contains the following topics:

- Callflow Post Installation Configuration
- Working with Java Composer Projects
- Working with .NET Composer Projects
- Accessing the Editors and Templates

Also see Upgrading Projects/Diagrams.

Callflow Post Installation

After installation of Composer, you need to perform some post-installation configuration tasks. **Note:** If you plan to use IIS as your web server for testing and deployment, you will also need to configure IIS preferences in Composer so that your applications can be auto-deployed to IIS from within the workbench. Composer can work only with IIS installed on the local machine. You can work with both Tomcat and IIS from the same installation of Composer. Also see: Context Services Preferences.

Tomcat

1. Select **Window** > **Preferences**, then expand **Composer** and select **Tomcat**.
2. Provide the same port number that you specified during installation. The default user name and password for the bundled Tomcat is `admin`.
3. To start Tomcat, click the button on the main toolbar.

If you already have Java Composer Projects in the workspace and did not perform the Tomcat configuration earlier, perform the following steps to deploy the project on Tomcat:

4. From the Project Explorer, right-click on the Java Composer Project and select **Propertie**s.
5. Select **Tomcat Deployment** and click the **Deploy** button.

Note: This also needs to be done if a Java Composer Project is imported.

Internet_Information_Services

1. Select **Window** > **Preferences**, then expand **Composer** and select **IIS/.NET.**
2. Provide the IIS website port number where you want to deploy your .NET Composer Project. The IIS Default Website Site port number is `80`.
3. If you plan to use .NET Composer Project builder to compile the server-side files (`.aspx`) in your .NET Composer Project, you will need to configure the location of the `aspnet_compiler`.exe file in the **Microsoft .NET Installed Path** field.

Note: The typical location of the ASP.NET compiler is:`C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_compiler.exe`.

4. Specify the **Web Services Enhancement (WSE)** path. This must be specified before Composer .NET Projects can work.

If you already have .NET Composer Projects in the workspace and did not perform the IIS configuration earlier, perform the following steps to deploy the project on IIS:

5. From the Project Explorer, right-click on the .NET Composer Project and select **Properties**.
6. Select IIS Deployment and click the **Deploy** button.

Note: This also needs to be done if a .NET Composer Project is imported or renamed as well.

GVP_Debugger

1. Select **Window** > **Preferences**, then expand **Composer** and select **Debugging**.
2. Specify the following settings:
    - **Network Interface**. Composer debugging uses this setting to make the socket connection for the Debugger control channel. Select the interface that is applicable to your scenario. The debugging server (GVP or ORS) must be able to access the Tomcat server, bundled as part of Composer, for fetching the Voice or Routing application pages. If you have multiple NIC cards of multiple networks (such as Wireless and LAN) select the interface on which GVP or ORS will communicate to your desktop. In case you are connected over VPN, select the VPN interface (such as PPP if connected via a Windows VPN connection).
    - **Client Port Range**. Enter a port range to be used for connection to ORS for SCXML debugging sessions.
3. Select GVP Debugger and specify:
    - **SIP Phone User Name**. This is the user name or phone number of your SIP Phone.
    - **SIP Phone Hostname/IP** . This is the IP address on which your SIP phone is running. It is possible to send the call to a SIP Phone located on some other machine, but it is generally advisable to have the SIP Phone locally for ease of access. If you have multiple NIC cards or interfaces, make sure you specify the same IP address as corresponds to the Network Interface selected above.
    - **SIP Phone Port**. This is the port on which your SIP phone is running.
    - **Platform IP**. This is the IP address of your GVP Server. Note: Composer 8.1 is compatible with GVP 8.1. Operation with GVP 8.0 is not supported.
    - **Platform Port**. Typically, this will be the default port `5060` or the port that you configured for the Resource Manager (RM) or Media Control Platform (MCP) on your GVP Server. You can make direct calls to MCP from the debugger. However, if using pre-provisioned DNIS, then you will need to make test calls to the RM.
    - **Use Secure Connection**. See Debugging TLS Support.

Composer may display a prompt asking if you wish to propagate these settings to an existing launch configurations.

MIME_Types

MIME (Multipurpose Internet Mail Extensions) refers to a common method for transmitting non-text files via Internet e-mail. By default the SCXML MIME type is already configured in the Tomcat server bundled with Composer. If you are using the Internet Information Services (IIS) Application Server to deploy ASP.NET projects, add the following MIME type extensions through the IIS Manager of your webserver:

.json	text/json
.vxml	text/plain
.scxml	text/plain
.xml	text/xml

Prompt_Resource_Validation

This preference enables diagram validation warnings where prompt audio resources no longer exist in the given file path. If the audio file is no longer present, the diagram block will show a warning icon.

1. Select **Window** > **Preferences**.
2. Select **Composer** > **Composer Diagram**.
3. Select the option **Enable Validation for Prompt Resources**. By default the preference is not enabled.

Media_Control_Platform

GVP 8.1 provides a debugger interface to allow Composer to make direct calls. By default it is turned off and you will have to enable it to allow GVP to accept calls from the debugger interface.

1. Outside of Composer, locate your Media Control Platform (MCP) Application. For example, you can open your MCP Application object in Configuration Manager or in Genesys Administrator for the Configuration environment that is serving the MCP platform.
2. Under the `vxmli` section of the MCP, look for a setting called `debug.enabled`. By default, it is set to `false`. Change the value to `true` and restart your MCP.

Firewall

If you have a local firewall on your machine, open up the following ports:

- Tomcat port (generally, this is set to port 8080). If you installed Tomcat on a different port, open its corresponding port in the firewall.
- IIS port (generally, this is set to port 80). If you installed IIS on a different port, open its corresponding port in the firewall.
- The UDP port on which your SIP phone is running (by default, this will be either 5060 or 5070). Check your SIP phone settings for the exact port number.
- RTP ports on which your SIP phone will get the audio stream. Check your SIP phone Help file for details on this. Some SIP phones will autoconfigure this during installation.

If you continue to run into problems with the firewall and calls are not coming through successfully, consult your network administrator.

Working with Java Composer Projects

A Java Composer Project contains voice application files, callflows, and related server side .jsp / Java files for building an IVR application. A Java Composer Project can also contain routing workflows. It has an associated Java Composer Project builder that will compile source files in the project. Composer ships with a bundled Tomcat and it is used as the web/application server for Java Composer Projects during the development and testing phase. For information on supported operating systems for Java Composer Projects, see the *Composer 8.1 Deployment Guide*.

Getting Started

To start using Java Composer Projects:

1. Create a new Java Composer Project.
2. Use the Project **Properties** tab to deploy the Java Composer Project to Tomcat within Composer. (Right-click the project > **Properties** > **Tomcat Deployment**.)
3. Create callflows and use Run or Debug mode to launch the call with Next Generation Interpreter.

Note: Run as / Debug as will automatically pick the port number from the preferences and form the corresponding Application URL. For example: http://machineIP:portno/JavaVoiceProjectName/src-gen/CallflowName.vxml

Working with .NET Composer Projects

A .NET Composer Project contains voice application files and related server side .aspx / C# / files for building an IVR program. It has an associated .NET Composer Project builder based on Microsoft .NET Framework that can incrementally compile .aspx source files as they are changed.

Prerequisites

Prerequisites for .NET Composer Projects are:

- Microsoft Internet Information Services (IIS) and .NET Framework as described in the Installation chapter of the *Composer 8.1 Deployment Guide*.
- Microsoft Web Services Enhancements (WSE) is also required for creating .NET projects in Composer. However, the WSE installer may not install on Windows 2008. These steps give a workaround:

1. Download the Microsoft WSE 3 "msi" installer bundle.
2. Use 7Zip to extract the contents to a folder.
3. In Composer, select **Window** > **Preferences** > **Composer** > **IIS/.NET**.
4. Set the **Microsoft WSE 3.0 Installed Path** field the `$Folder\Microsoft.Web.Services3.dll` file.
5. Create your Composer .NET Projects.

Getting Started

To prepare for using .NET Composer Projects:

1. Install Microsoft IIS.
2. Install Microsoft .NET and .NET Framework.

Note: Microsoft .NET is required for Composer Server Side blocks.

1. Enable ASP.NET in your IIS.
2. Configure the following MIME settings in your IIS:
    - `.ccxml` - application/ccxml+xml
    - `.vxml` - text/xml
    - `.grxml` - application/srgs+xml
    - `.vox` - audio/basic
    - `.scxml` - application/xml

3. Configure the IIS Website Port number in Composer IIS Preferences (**Window** > **Preferences** > **.NET**)

By default, IIS comes with the DefaultWebSite which runs on port 80. If you want to deploy the .NET Composer Project in your custom website, configure the corresponding port number in the IIS Website Port field.

1. Create a .NET Composer Project.
2. Use the Project **Properties** tab to deploy the .NET Composer Project to IIS within Composer. (Right-click **Properties** > **IIS Deployment**.)
3. Create the diagram callflows and perform Run as / Debug as to launch the call with NGI.

Note: Run as / Debug as will automatically pick the port number from the preferences and form the corresponding application URL. For example: `http://machineIP:portno/NETProjectName/src-gen/CallflowName.vxml` Also see: Request.Form Error Message for .NET Projects

Accessing the Editors and Templates

Composer editors are embedded/integrated within the user interface and are made available to you whenever a .scxml, .vxml, .ccxml, .grxml, or .jsp file is created or accessed within Composer.

Creating a New File

In **Composer** or **Composer Design** perspective, create a new VoiceXML, SCXML, or CallControlXML file as follows:

- Select **File** > **New** > **Other** > **Composer** > **Others**.
- Select the file type.
- Select the parent folder; usually an existing Project.
- Enter a name for the file.
- If applicable, click **Advanced** to link to the file system and use an existing file.
- Click **Finish**.

Using an Existing Template

- Select **File** > **New** > **Other** > **Composer** > **Others**.
- Select the file type.
- Select the parent folder; usually an existing Project.
- Enter a name for the file.
- If applicable, click **Advanced** to link to the file system and use an existing file.
- Click **Next**.
- Select the template.
- Click the **Use SCXML Template** checkbox.
    - Click **Finish**.

The editor opens with your new file. When working with XML files, the view contains **Source** and **Design** tabs. All editor functions described at the top of this topic are available to you. The appropriate Composer editor also opens whenever you open an existing .vxml, .ccxml, .grxml, .aspx, or .jsp file, whether previously created as described above, or previously imported into Composer.

Open an Existing File

Open an existing file as follows:

- Select **File** > **Open File**.
    - Navigate to the file to open, OR

Open a Composer Project's `src` or `src-gen` folder in the Project Explorer, then double-click the file to open it in the editor.

Creating a Custom Code Template

When writing manual SCXML/VXML/CCXML/GRXML code in the file editors, you may run into code that becomes repetitive. You may consider creating a code template to avoid retyping this block of code. Creating templates will improve the speed and consistency for writing code. The following steps show how to create a code template.

- Select **Window** > **Preferences**.
- In the Preferences dialog box, navigate through the Composer category, and expand the file type (VXML Files / CCXML Files / GRXML / SXCML Files) in which you want to add your template. Then select the **Templates** section. For example, select VXML Templates.
- Click the **New** button to add a new code template.
- Fill in the fields for the new template. The Context drop down box specifies at what context level you want the code template to appear as a context sensitive help.
- Click the **OK** button when finished.

XML File Preferences

You can also set XML File Preferences: **Window** > **Preferences** > **XML** > **XML Files**. When specifying Encoding formats in the XML Preference page: encoding formats are applicable only for new File creation using the Template option: (**File** > **New** > **XML** > **XML File** > **Create XML File from an XML Template** > **Select XML Template**). This applies only to new XML, VXML, CCXML and SCXML files. Existing files within the Project will not get impacted.

Creating a Backend JSP File

- Create a new JSP file by selecting **File** > **New** > **Backend JSP file**.
- In the **Create Backend JSP File** folder, navigate to the `src` folder within the Java Composer Project in which the Backend JSP file belongs.
- Type a name In the **File Name** field.

- Click **Finish**.

The Editor opens with a JSP file template. You can see your new file in the `src` folder of your Java Composer Project in the Project Explorer. A template is provided when you create a new Backend JSP file in Composer. You implement a performLogic method as a JSON object, store a result and return it to the voice application if desired. You have the flexibility to enter any valid JSP code that you wish.

Creating a Backend ASP .NET File

- Create a new ASP.NET file by selecting **File** > **New** > **Backend ASPX file**.
- In the ASPX File folder, navigate to the include folder within the .NET Composer Project in which the Backend ASPX file belongs.
- Type a name In the **File Name** field.
- Click **Finish**.

The Editor opens with an ASPX file template. You can see your new file in the include folder of your .NET Composer Project in the Project Explorer. A template is provided when you create a new Backend ASPX file in Composer. You implement a performLogic method as a JSON object, store a result and return it to the voice application if desired. You have the flexibility to enter any valid ASP.NET/C# code that you wish.

Preferences for Voice Applications

Composer Preferences are applicable at a workspace level. They apply to all projects within the workspace. To open the Preferences dialog box for Composer, select **Window** > **Preferences** and expand **Composer**. There are also Refresh automatically and Time zone preferences.

CCXML File Preferences

Select **Window** > **Preferences** > **Composer** > **CCXML File**s. The following preferences for CCXML files can be set in the Preferences dialog box:

Creating Files

1. Under **Creating Files**, select the Suffix to use for CCXML files from the drop-down list: * ccxml
2. Select the **Encoding** type from the drop-down list. The encoding attribute in a CCXML document specifies the encoding scheme. The encoding scheme is the standard character set of a language. The CCXML processor uses this encoding information to know how to work with the data contained in the CCXML document. UTF-8 is the standard character set used to create pages written in English. Select from the following:
    - ISO 10646/Unicode(UTF-8)
    - ISO 10646/Unicode(UTF-16) Big Endian
    - ISO 10646/Unicode(UTF-16BE) Big Endian
    - ISO 10646/Unicode(UTF-16LE) Little Endian
    - US ASCII
    - ISO Latin-1
    - Central/East European (Slavic)
    - Southern European
    - Arabic, Logical
    - Arabic
    - Chinese, National Standard
    - Traditional Chinese, Big5
    - Cyrillic, ISO-8859-4
    - Cyrillic, ISO-8859-5
    - Greek
    - Hebrew, Visual
    - Hebrew
    - Japanese, EUC-JP
    - Japanese, ISO 2022
    - Japanese, Shift-JIS
    - Japanese, Windows-31J

- Korean, EUC-KR
- Korean, ISO 2022
- Thai, TISI
- Turkish

Validating Files

- Select or clear the **Warn when no grammar is specifie**d check box (not selected by default).

Source and Syntax Coloring

Source and Syntax Coloring preferences for CCXML files are set under the XML preferences provided by Eclipse.

- Use **Window** > **Preferences**, expand the **Web and XML** entry, then expand the **XML Files** subentry.

Templates

In addition to previewing templates, you can create, edit, and remove selected templates. There are also buttons to:

- Restore a removed template.
- Revert back to a default template
- Import a template.
- Export a template.

Diagram Preferences

Select **Window**> **Preferences** > **Composer** > **Composer Diagram**. The following preferences for diagrams can be set in the Preferences dialog box:

Global Settings

1. Select or clear the check box for each of the following diagram global settings:

   - **Show Connection Ports**. If enabled, connection ports (both exception ports and out ports) are always displayed on blocks. This makes it convenient to draw links between blocks and to get immediate feedback on how many ports each block provides. However, in this case, the ability to reposition connections on a block is not available. If switched off, connection ports are not displayed by default, but repositioning or finer control over connection link placement becomes available. Note: This preference applies to all projects and is not available for individual projects.)
   - **Show popup bars**. If enabled, this setting displays basic blocks from the blocks palette in a pop-up bar if you hover your mouse on the diagram for one or two seconds without clicking. Note: blocks are shown in icon view only.)
   - **Enable animated layout**. If enabled, causes diagrams to gradually animate to their location when the Diagram \> Arrange \> Arrange All menu option is clicked.
   - **Enable animated zoom**. If enabled, while using the zoom tools, shows a gradual transition between the initial and final state of the diagram on the canvas. If off, the zoom is instantaneous. Similar behavior for animated layout when the Diagram \>\> Arrange \>\> Arrange All menu option is clicked.
   - **Enable anti-aliasing**. If enabled, improves the appearance of curved shapes in the diagram. You can see its effect on the circles in the Entry and Exit blocks.
   - **Show CodeGen success message**. If unchecked, then the confirmation dialog at the completion of code generation will not be shown.)
   - **Prompt to Save Before Generating Code**. If checked, when you generate code for an unsaved diagram, a prompt appears indicating the diagram has been modified and asking if you want to save the changes before generating code. The dialog box also contains a checkbox: Automatically save when generating code and do not show this message again.
   - **Show Validation success message**. If unchecked, then the confirmation dialog at the time of Validation will not be shown.)
   - **Enable Validation for Prompt Resources**. This preference is used for voice applications. If unchecked, then a validation check for missing prompts is not performed at the time of Validation.
   - **Interaction Process Diagram**. If unchecked, Composer will save Interaction Process Diagrams before publishing.
   - **Prompt to delete Published objects when Interaction Process Diagram is deleted**. If unchecked, Composer will attempt to delete any Published objects when an Interaction Process Diagram is deleted. If Composer is not connected to Configuration Server, object

deletion will not work.

2. Click Apply.

Colors and Fonts

1. Select **Appearance** under Composer Diagram.
2. Click **Change** and make selections to change the default font if you wish.
3. Click the appropriate color icon beside any of the following and make selections to change color:
    - **Font color**
    - **Fill color**
    - **Line color**
    - **Note fill color**
    - **Note line color**
4. Click **Apply**.

Connections

1. Select Connections under Composer Diagram.
2. Select a line style from the drop-down list:
    - **Oblique**
    - **Rectilinear**
3. Click **Apply**.

Pathmaps

1. Select **Pathmaps** under Composer Diagram.
2. Click **New** to add a path variable to use in modeling artifacts, or If the list is populated, select the check box of a path variable in the list.
3. Click **Apply**.

Printing

1. Select **Printing** under Composer Diagram.
2. Select **Portrait** or **Landscape** orientation.
3. Select units of Inches or Millimetres.
4. Select a paper size (default is Letter).
5. Select a width and height (for inches, defaults are 8.5 and 11; formillimeters, defaults are 215.9 and 279.4).
6. Select top, left, bottom, and right margin settings (for inches, defaults are 0.5; for millimeters, defaults are 12.7).
7. Click **Apply**

Rulers and Grid

You can make use of rulers and grids when creating diagrams. Rulers and grids can provide a backdrop to assist you in aligning and organizing the elements of your callflow diagrams.

1. Select Rulers and Grid under Studio Diagram.
2. Select or clear the **Show rulers for new diagram** check box (not selected by default).
3. Select ruler units from the drop-down list:
    - **Inches**
    - **Centimeters**
    - **Pixels**
4. Select or clear the Show grid for new diagrams check box (not selected by default).
5. Select or clear the Snap to grid for new diagrams check box (selected by default).
6. Type a value for grid spacing (for inches, the default is 0.125; for centimeters, the default is 0.318; for pixels, the default is 12.019).
7. Click **Apply**.

GAX Server Preferences

Select **Window** > **Preferences** > **Composer** > **GAX Server**. If using the OPM Block for a voice or routing application, you must set GAX Server Preferences. **Note:** *GAX* refers to a Genesys Administrator Extension (GAX) plug-in application used by Genesys EZPulse, which is accessible from a web browser. EZPulse enables at-a-glance views of contact center real-time statistics in the GAX user interface. Composer diagrams connect to GAX using the preference login credentials for fetching the Audio Resource Management (ARM) parameters or IDs list configured for the tenant as described in the Configuration options appendix of the *Genesys Adminstrator Extension Deployment Guide*. The following preferences can be set in the GAX Server Preferences dialog box:

- **Server Host Name/IP**. Enter the hostname or address of the Application server hosting the GAX Server.
- **Port Number**. Enter the port number for the GAX Server used in your environment.
- **Username**. Enter the username defined in the Configuration Database for logging into the GAX server.
- **Password**. Enter the password defined in the Configuration Database for logging into the GAX server.

GRXML File Preferences

Select **Window** > **Preferences** > **Composer** > **GRXML Files**. The following preferences for GRXML files can be set in the Preferences dialog box:

Creating Files

1. Under **Creating Files**, select the Suffix to use for GRXML files from the drop-down list:* grxml
2. Select the **Encoding** type from the drop-down list. The encoding attribute in a GRXML document specifies the encoding scheme. The encoding scheme is the standard character set of a language. The GRXML processor uses this encoding information to know how to work with the data contained in the GRXML document. UTF-8 is the standard character set used to create pages written in English. Select from the following:

    - ISO 10646/Unicode(UTF-8)
    - ISO 10646/Unicode(UTF-16) Big Endian
    - ISO 10646/Unicode(UTF-16BE) Big Endian
    - ISO 10646/Unicode(UTF-16LE) Little Endian
    - US ASCII
    - ISO Latin-1
    - Central/East European (Slavic)
    - Southern European
    - Arabic, Logical
    - Arabic
    - Chinese, National Standard
    - Traditional Chinese, Big5
    - Cyrillic, ISO-8859-4
    - Cyrillic, ISO-8859-5
    - Greek
    - Hebrew, Visual
    - Hebrew
    - Japanese, EUC-JP
    - Japanese, ISO 2022
    - Japanese, Shift-JIS
    - Japanese, Windows-31J

- Korean, EUC-KR
- Korean, ISO 2022
- Thai, TISI
- Turkish

Validating Files

- Select or clear the **Warn when no grammar is specified** check box (not selected by default).

Source and Syntax Coloring

Source, Syntax Coloring, and Template preferences for GRXML files are set under the XML preferences provided by Eclipse.

- Use **Window** > **Preferences**, expand the **Web and XML** entry, then expand the **XML Files** subentry.

Templates

In addition to previewing templates, you can create, edit, and remove selected templates. There are also buttons to:

- Restore a removed template.
- Revert back to a default template
- Import a template.
- Export a template.

VXML File Preferences

Select **Window** > **Preferences** > **Composer** > **VXML Files**.

Note: Composer natively supports VXML 2.1.

The following preferences for VXML files can be set in the Preferences dialog box:

Creating Files

1. Under **Creating Files**, select the Suffix to use for VXML files from the drop-down list: *.vxml
2. Select the **Encoding** type from the drop-down list. The encoding attribute in a VXML document specifies the encoding scheme. The encoding scheme is the standard character set of a language. The VXML processor uses this encoding information to know how to work with the data contained in the VXML document. UTF-8 is the standard character set used to create pages written in English. Select from the following:
    - ISO 10646/Unicode(UTF-8)
    - ISO 10646/Unicode(UTF-16) Big Endian
    - ISO 10646/Unicode(UTF-16BE) Big Endian
    - ISO 10646/Unicode(UTF-16LE) Little Endian
    - US ASCII
    - ISO Latin-1
    - Central/East European (Slavic)
    - Southern European
    - Arabic, Logical
    - Arabic
    - Chinese, National Standard
    - Traditional Chinese, Big5
    - Cyrillic, ISO-8859-4
    - Cyrillic, ISO-8859-5
    - Greek
    - Hebrew, Visual
    - Hebrew
    - Japanese, EUC-JP

- Japanese, ISO 2022
- Japanese, Shift-JIS
- Japanese, Windows-31J
- Korean, EUC-KR
- Korean, ISO 2022
- Thai, TISI
- Turkish

Validating Files

- Select or clear the **Warn when no grammar is specified** check box (not selected by default).

Source, Syntax Coloring, and Templates

Source, Syntax Coloring, and Template preferences for VXML files are set under the XML preferences provided by Eclipse.

- Use **Window** > **Preferences**, expand the **Web and XML** entry, then expand the **XML Files** subentry.

This preference allows you to add custom VXML schemas into Composer to be used in namespaces for new VXML files created through the VXML editor.

GVP Debugger Preferences

Select **Windo**w > **Preferences** > **Composer** > **Debugging** > **GVP Debugger**. GVP Debugger preferences are usually set during callflow post-installation configuration, when you first run Composer. Detailed post-installation configuration instructions are also provided in the Configure Tomcat and Debugger Settings Cheat Sheet (**Help** > **Cheat Sheets** > **Composer** > **Voice Applications**).

IIS.NET Preferences

Select **Window** > **Preferences** > **Composer** > **IIS/.NET**.

IIS/.NET preferences are usually set during post-installation configuration, when you first run Composer. Detailed post-installation configuration instructions are provided in the Setting IIS Preferences Cheat Sheet (**Help** > **Cheat Sheets** > **Composer** > **Building Voice Applications**), and also in Information Services Post-Installation Configuration.

Time Zone Preferences

Composer displays all date/time elements in the user-preferred time zone with the time zone identifier. You can change the preferred time zone in **Window** > **Preferences** > **Composer** > **Context Services**.

Tomcat Preferences

Select **Window** > **Preferences** > **Composer** > **Tomca**t Tomcat preferences are usually set during post-installation configuration, when you first run Composer. Detailed post-installation configuration instructions are provided in the Configure Tomcat and Debugger Settings Cheat Sheet (**Help** > **Cheat Sheets** > **Composer** > **Building Voice Application**s), and also in Callflow Post-Installation Configuration or Workflow Post Installation Configuration.

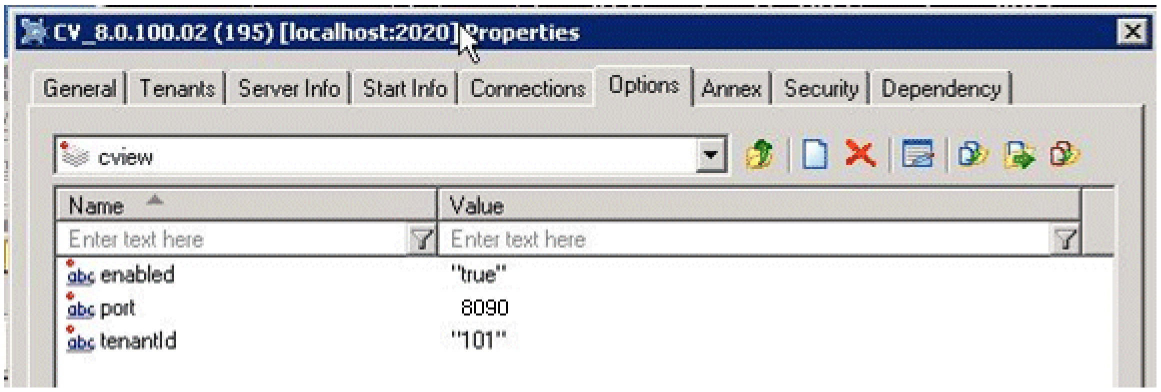
XML Preferences

You can also set XML File Preferences for both routing and voice applications: **Window** > **Preferences** > **XML** > **XML Files**. When specifying Encoding formats in the XML Preference page: encoding formats are applicable only for new File creation using the Template option: (**File** > **New** > **XML** > **XML File** > **Create XML File from an XML Template** > **Select XML Template**). This applies only to new XML, VXML, CCXML and SCXML files. Existing files within the Project will not get impacted.

Setting Context Services Preferences

When working with Context Services blocks, you may wish to use online mode. In this mode, Composer fetches data from Universal Contact Server during design phase to help you configure the blocks. For example, Composer can fetch customer profile attribute names, extension attribute names, and so on. You can enable/disable this behavior in the Context Services Preferences page. If the Context Services capability is enabled at your site, set preferences as follows:

1. Go to **Window** > **Preferences** > **Composer** > **Context Services**.
2. Check the following box to specify online or offline mode when connecting to Context Services: **Connect to the Universal Contact Server when designing diagrams**. This enables the fields below.
3. Under **Universal Contact Server**, enter the server host name in your Configuration Database, which is the name (or IP address) of the Universal Contact Server. Also see the Runtime Configuration topic.
4. Enter the **Server Port number** for Universal Contact Server. **Note:** For the port number, open the Universal Contact Server Application object in your Configuration Database, go to Options tab, select the cview section, and the port option. Example settings are shown below.



Name	Value
enabled	"true"
port	8090
tenantId	"101"

5. Enter the **Base URL** for the Context Services server (UCS).

The GVP Debugger passes all host, port, and base URL parameters to the VXML platform. It uses the parameters to make an url made of: `[http:// http://]<host-parameter>:<port-parameter>[/<base-url>`.

6. Under **Security Settings**, **Use secure connection**, select **never** or **TLS if Transport Layer Security** is implemented as described in the *Genesys 8.1 Security Deployment Guide*. Also see *Debugging Transport Layer Security*.
7. Select **Use Authentication** to require a user name and password when connecting to Universal Contact Server. If selected, enter the User and Password fields.
8. On the Context Services Preferences page, click the **Test Connection** button. Clicking should cause connection successful to appear. If not, check that Universal Contact Server is running and that the entered host/port values are correct. Other sources of error could be:

    - Base URL parameter value is incorrect

- UCS version is not 8.1 or higher

Note: Composer can successfully communicate with UCS at design stage whatever the UCS mode is (production or maintenance). However, UCS needs to be in production mode at runtime stage (when running Context Services SCXML or VXML applications, even when using GVP Debugger).

9. Under **Context Services object Validation,** select one of the following:
    - **No validation**
    - **Validate if connected**
    - **Validate**

Creating Voice Applications for GVP

This section provides key information about using Composer to build VoiceXML-based callflow applications. You should be well-versed in VoiceXML, XML, and HTML before attempting to use Composer. You should also have reviewed Getting Started with Voice Applications. To help you get started:

- From within Composer, select **Help** > **Cheat Sheets** > **Composer** to see how some basic voice applications can be created.
- Your First Application
- Sample Applications & Templates.

To start immediately, see:

- Creating VXML Applications or
- Creating CCXML Applications

What is GVP and How Do Voice Apps Work

The Genesys Voice Platform (GVP) is a VoiceXML-based media server for network service providers and enterprise customers.

What is GVP?

At the most basic level, Genesys Voice Platform (GVP) is Interactive Voice Response software (soft IVR). At a more complex level, GVP is a software suite that integrates a combination of call processing, reporting, management, and application servers with Voice over IP (VoIP) networks, to deliver web-driven dialog and call control services to callers.

Using features such as Automatic Speech Recognition (ASR) and Text-to-Speech (TTS), GVP provides a cost-effective way to implement automated voice interactions from customers calling your contact center. At the technology level, GVP is a collection of software components that complement and work with other Genesys products in order to provide a complete voice self-service solution.

Notes:

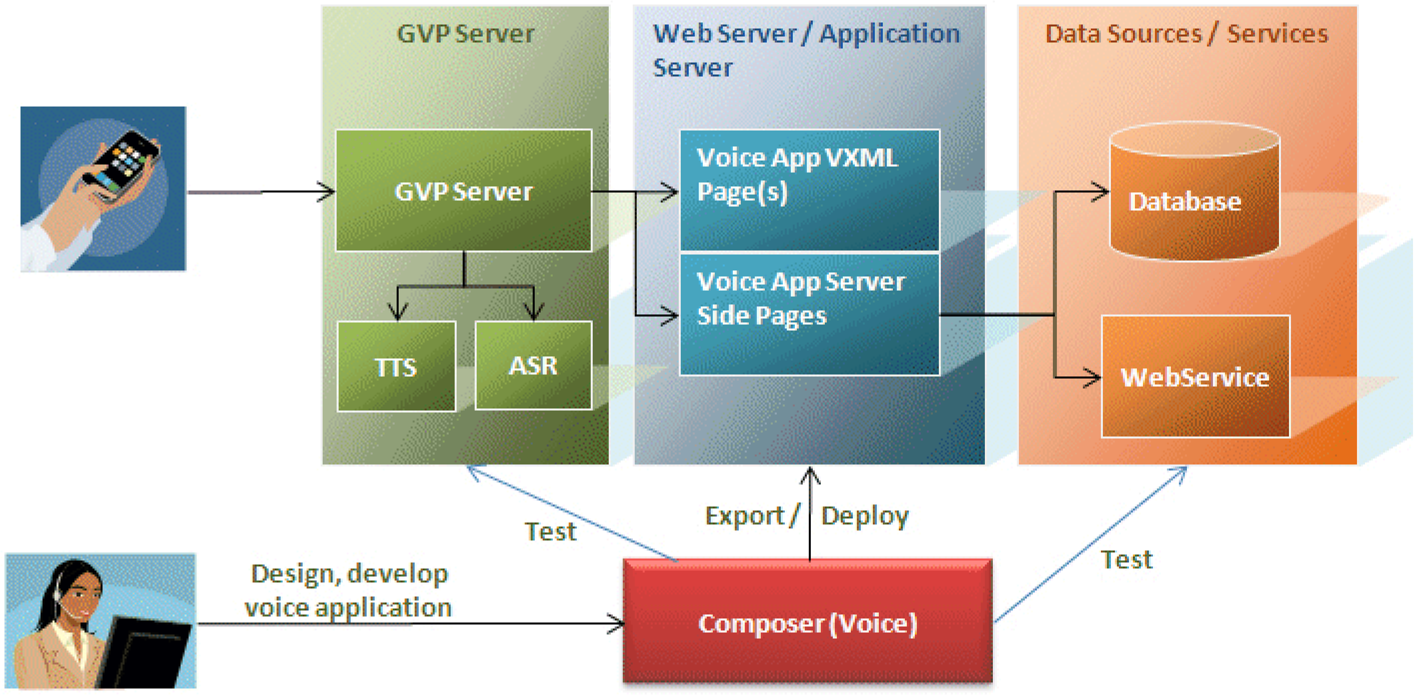
Whereas GVP is commonly used in enterprise self-service environments, many other applications of GVP —including those outside of the contact center—are possible.

A machine on which GVP components are installed is also referred to as a GVP Server in other places in this Help system.

How Do Voice Applications Work?

Just as one uses HTML to create visual applications, VoiceXML is a mark-up language one uses to create voice applications. With a traditional web page, a web browser will make a request to a web server, which in turn will send an HTML document to the browser to be displayed visually to the user. With a voice application, it's the VoiceXML interpreter that sends the request to the web server, which will return a VoiceXML document to be presented as a voice application via a telephone. What makes VoiceXML so powerful is that all of the most popular tools for making web pages are available for making voice applications. Developers can use technologies they are already familiar with such as JavaScript, JSP and ASP.NET/C# to generate exciting new voice applications.

The "Big Picture"



Composer is a fully featured VXML application development tool. Users can develop, debug and test their applications in its Integrated Development Environment (IDE) that provides developer-friendly features to test and debug VXML applications and server side web pages. Once the application is ready, it can be exported or manually deployed using an exported package onto an application server/web server like Tomcat or Microsoft IIS. Once deployed, GVP can access the voice application's VXML pages and any server side pages (JSP/ASP.NET) using HTTP.

When a call comes in to GVP, GVP determines the location of the VXML application through its provisioning data. It then fetches VXML page(s) and uses its VXML engine to execute them. The results are played back to the caller on his/her phone. Any server side pages that access databases or web services or other server side pages are executed on the application server/ web server through server side constructs implemented by Composer.

During development, Composer can use its bundled Tomcat or a local installation of Microsoft IIS as the web server and make test calls to the application right through GVP from within the IDE. This feature provides a quick way to test applications by removing the need to of deploy applications to another server and then point GVP to that location.

Once the application is deployed in production, Composer is no longer in the picture. The application is usually deployed on its own dedicated web server and application server from where it is accessed by GVP. The web/application server provides access to all pages and scripts that make up the application and executes any server side pages of the application.

Creating CCXML Applications

CCXML (Call Control XML) is a specification developed by the Call Control subgroup of the Voice Browser Working Group of the W3C. CCXML provides mechanisms for implementing advanced call control functionality in a standards-based way. It provides the advanced call control features not supported by VoiceXML. You develop CCXML a little differently than VXML or SCXML applications. Rather than creating flow diagrams, you invoke a CCXML text editor and enter the code while Composer performs syntax checking. To create a new CCXML file in Composer perspective:

1. From the menu, select **File** > **New** > **Other** > **CallControlXML** File.
2. In the wizard, select the Project folder, name the file, and click Finish or Next to use a template.
3. If you click Next, select a template and click Finish. Composer opens the view the CCXML editor view.

Creating VXML Applications

When building any application in Composer, you first need to create a Project. A Project contains all the callflows, audio and grammar files, and server side logic for your application. By associating a routing strategy with a Project, you enable Composer to manage all the associated files and resources in the Project Explorer.

Cheat Sheet

Composer provides a cheat sheet to walk you through the steps for building a voice application.

- In the Welcome Screen (**Help** > **Welcome**), click the icon for Tutorials and select the Create a Voice Application tutorial. It will also describe the steps for how to make test calls and debug your application.
- If you are already inside the Workbench and Perspectives, access the same cheat sheet from the Menu bar at the top by selecting Help > Cheat Sheets, then Create Voice Application from the Building Voice Applications category.

Creating a New Project

You can follow the steps below to create a new Project:

1. For a Java Composer Project to be deployed on Tomcat, click the toolbar button to create a **Java Composer Project**. For a .NET Composer Project to be deployed on IIS, click the toolbar icon to create a .**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:
    - **Integrated Voice and Route**. Select to create a Project that contains both callflows and workflows that interact with each other; for example a routing strategy that invokes a GVP voice application. For more information on both voice and routing applications, see What is GVP and How Do Voice Apps Work? and What Is a Routing Strategy, respectively.
    - **Voice**: Select to create a Project associated with the GVP 8.x. This type of Project may include callflows, and related server-side files. For more information on this type of Project, see topic, How Do Voice Applications Work.
    - **Route**: Select to create a Project associated with the URS 8.0 SCXML Engine/Interpreter. For more information on this type of Project, see topic, What Is a Routing Strategy.
5. Click Next.

6. If you want to use templates, expand the appropriate Project type category and select a template for your application. Templates are sample applications for different purposes. If you want to start from scratch, choose the Blank Project template and click Next.
7. Select the default locale and click Next.
8. Optional. If using the in a VoiceXML application, select the **Enable ICM** checkbox to enable integration. When checked, ICM variables will be visible in the Entry block. See the ICM Interaction Data block for more information.
9. Click **Finish**. Composer now creates your new Project. Your new Project folder and its subfolders appear in the Project Explorer.

Note: If the Project Explorer does not display or if this is your first time using Composer, click the large workbench bubble icon on the blank screen to display the Project Explorer. Choose another topic in this workbook to see common steps for designing a Project for voice applications. If you have never created a Composer Project, we recommend starting with Your First Application.

Creating a New Callflow

To add a new callflow diagram to an existing Composer Project:

1. Click the button on the main toolbar to create a new callflow. Or use the keyboard shortcut: Ctrl+Alt+O.
2. In the wizard, select the tab for the type of the callflow. There are two main types of callflows in Composer represented by wizard tabs:
    - Main Callflow: Used for the main application where the call will land or be transferred to from another application.
    - Subcallflow: Used for modularizing your applications. It is useful for structuring large applications into manageable components.

Additionally you will benefit from the automated transaction reports associated with Subcallflows. Action Start and Action End VAR events are auto-generated for Entry and Exit blocks.

3. Select either Main Callflow or Subcallflow.
4. Select the type of diagram.
5. Click Next.
6. Select the Project.
7. Click Finish.
8. Create the callflow.

Validation

Composer can validate your diagram files and other source files for completeness and accuracy. For more information, see Validation.

Code Generation

The process of generating code creates a properly-formatted VoiceXML file from a callflow diagram built with Composer or a SCXML file from a workflow diagram. Static pages (pure VXML or SCXML code) are generated in the src-gen folder of the Composer Project. You can generate code in a couple of ways:

- Select Diagram > Generate Code.
- Click the Generate Code icon on the upper-right of the Composer main window when the callflow/ workflow canvas is selected.

Note: If your project uses the Query Builder or Stored Procedure Helper-generated queries in DB Data blocks, the process of code generation will create one SQL file in the db folder for each such DB Data block. These SQL files will be used at runtime and should not be deleted.

Code Generation for Multiple Callflows

When using the Run as Callflow function, Composer automatically generates the VXML files from the diagram file that you want to run. When generating code, with the generate code function for a Java Composer Project that has multiple callflows, Composer attempts to generate the VXML for all the callflows before running (because the application might move between multiple callflows for subdialogs). However, if one of the callflows has an error, Composer provides the option to continue running the application anyway, because the erroneous callflow may be a callflow that's not used by the one being run (if there are two or more main callflows, for example). When this happens, the VXML files are basically out of sync with the diagram files and this may affect execution. Genesys recommends that you fix all errors before running the application.

Deploying/Testing Your Application

After you have saved your files and generated code for your application, test the application as follows:

1. Deploy the project for testing.
    - If deploying a Java Composer Project, Composer bundles Tomcat 6.0 for running test applications, such as routing applications. If you configured the Tomcat settings prior to creating your Project, it will be auto-deployed on the Tomcat Server. You can double check this by clicking on the name of the project in the Project Explorer, then right-click and select Project Properties. Select the Tomcat deployment category and verify that the project is deployed. If not, click Deploy.
    - If deploying a .NET Composer Project, deploy your project on an IIS Server. Be sure you have configured the IIS settings. Click on the name of the project in the Project Explorer, then right-click and select Project Properties. Select the IIS deployment category and verify that the project is deployed. If not, click Deploy.
2. For Voice Projects, use Run mode to run the application by selecting Run > Run As > Run Callflow, or by right-clicking on the callflow file name in the Project Explorer and selecting Run As > Run Callflow. The code is generated in the src-gen folder and the debugger sends the call to your SIP Phone.
3. Accept the call and you will be connected to the application on GVP. The call traces will become visible in the Call Trace window, and you should hear the voice application run.

Hello World Sample

Here is a simple voice application to help you get started with Composer. This application says Hello World when the call is answered.

Simple Text-to-Speech Application

To build a simple text-to-speech (TTS) application that says Hello World to the caller:

1. Create a new Composer Project called Hello World.
2. Add the following blocks from the Basic Blocks Palette to the canvas area: Entry, Prompt, and Exit, then connect them with Output Links.
3. Select the Entry block, or right-click the Entry block and select **Show Properties View** from the shortcut menu, if you want to set any properties (optional).
4. Select the Prompt block, or right-click the Prompt block and select **Show Properties View** from the shortcut menu.
5. Select the Name property and type a name in the Value field.
6. Select the Prompts property and click the button.
7. Click the **Add** button and type a name in the Name field (optional).
8. Select **Value** in the Type drop-down list (default).
9. Select **Text** in the Interpret-As drop-down list (default).
10. Type **HelloWorld** (one word) in the Value field.
11. Click **OK**.
12. Save the file by selecting **File** > **Save**. You will not be able to generate code if you do not save the file.
13. Generate the code by selecting **Diagram** > **Generate Code**, or by clicking the Generate Code icon on the upper-right of the Composer main window when the callflow canvas is selected.
14. If you get any errors, double-click on the error to get the details and fix the problem. For the Hello World application, typical problems would be forgetting to add the Hello World prompt or forgetting to link the blocks together.
15. If code generation succeeds, click OK at the confirmation dialog box.
16. Make sure the project is deployed for testing. Composer bundles Tomcat 6.0 for running test applications. If you configured the Tomcat settings prior to creating your Composer Project, it will be auto-deployed on the Tomcat Server. You can double check this by clicking on the name of the project in the Project Explorer, then right-click and select Project Properties. Select the Tomcat deployment category and verify that the project is deployed. If not, click Deploy.
17. Select the callflow in the Project callflows folder.
18. Run the application by selecting **Run** > **Run As** > **Run Callflow**, or by right-clicking on the callflow file

name in the Project Explorer and selecting **Run As** > **Run Callflow**.

The code is generated in the `src-gen` folder and the GVP debugger sends the call to your SIP Phone.

19. Accept the call and you will be connected to the application on GVP. The call traces will become visible in the Call Trace view, and you should hear Hello World played through the phone.

Adding Blocks

There are a few ways to add blocks from the Palette to the canvas. The most common methods are as follows:

- Click on the block icon on the palette, release the mouse and click on the target location on the canvas area.
- Double-click a block icon on the palette.
- Click on the block icon on the palette, and while holding down the mouse button, drag and drop the block to the canvas.

Any of these methods will add the new block and you can then type the name of the block on the canvas itself. Click Property here to read about block naming restrictions.

Connecting Blocks

Blocks are connected to each other using connection links. There are two types of connection links:

- Output Links used to connect one block's output port to another block's input port, and
- Exception Links used to indicate error or exception conditions by connecting from a block's exception port to another block's input port.

To add a new Output Link (or Exception Link):

1. Click the Output Link (or Exception Link) icon in the palette.
2. Move the mouse over to the source block. The cursor will change to an upward arrow.
3. Click once on the source block and keep the mouse button pressed. Then drag the mouse onto the target block and release the mouse button.

This will add the connection link between the two blocks. To use an Exception Link, the source block must have an exception port defined. This is done by selecting at least one supported exception within the block's Exceptions property.

Another method for adding an Output Link or Exception Link between two blocks is as follows:

1. Click once on the source block to select it.
2. Hold the Ctrl key and click once on the target block to select it as well.

3. Double-click the Output Link (or Exception Link) icon in the palette to create a connection between the two blocks.

Again, to use an Exception Link, the source block must have an exception port defined.

The preference Show Connection Ports (in Composer Diagram Preferences) affects how connection links can be drawn to connect blocks. If it is switched on, links may be drawn directly by dragging from an outport of a block and dropped onto a block or its inport. This method will work in addition to using the Output link and Exception link tools. If the setting is switched off, connection ports are not displayed and therefore the method of drawing links mentioned above is not available.

Callflow Blocks

A block is the fundamental element of a callflow. Each block defines specific properties and how to handle specific events. You use the Link tools to connect these blocks in the order that the application should follow. A single VXML application is generated per callflow. Each block in a callflow becomes a form in the generated VXML document.

VXML Properties

Each block has custom VoiceXML properties. These properties appear within a Properties view at the bottom of the Composer window when you right-click the block and then select Show Properties View from the shortcut menu. For each block, specific properties determine how events are handled. There are several categories of properties depending on the specific block. The blocks build a callflow or subcallflow. Generate code either from the Toolbar or from the Diagram menu. Static VXML pages (pure VXML code) are generated in the src-gen folder.

Main Versus Subcallflow

There are two types of callflows:

- Main Callflow: This is the starting callflow for any application.
- Subcallflow: This is a component callflow that can be called from the main callflow or another subcallflow.

Each main callflow or subcallflow application should have at least three blocks:

- The Entry block to start the application. This block also specifies the relative file locations of the audio files for the generated application code and default exception handling.
- At least one other block to perform specific functions such as passing a call to an agent, creating a log of an activity, requesting caller input, playing a prompt, and so on.
- The Exit block to end the application, or, for example, the GoTo block to direct the application to another application.

Subcallflows

Subcallflows are used for modularizing applications and for writing components that can be reused by multiple applications (such as a credit card validation subcallflow). The usage of subcallflows within a main callflow is very similar to a function call in a programming language. One or more input parameters can be passed to a subcallflow. Similarly, the subcallflow can return one or more output parameters. Therefore, a subcallflow can be designed to behave differently depending on the input parameter(s) passed.

Variables in Callflows

You can define voice application (session) variables using the Entry block Variables property.

Types of Variables

Composer supports the following types of variables for callflow diagrams:

- **System--Pre-defined application variables** (System category above) hold Project and application-related values. While you cannot delete System variables, you can have your application modify the values.
- **User--User-defined custom variables** that you create by clicking the Add button in the Application Variables dialog box above and selecting User. Your application can delete and modify User variables.
- **Input**--These are variables supplied as input to the called diagram. Created by clicking the Add button in the Application Variables dialog box above and selecting User. During runtime, Input variables get auto-filled from the calling context. Typically Input variables are created on the SubCallflow side to notify the MainCallflow about the Parameter-passing details while designing the application flow. Composer does auto-synchronization of the Input variables in the Subdialog block.Input variables are also used on the MainCallflow while invoking the VoiceXML application from workflows in case of Voice Treatment execution - computer telephony integration (CTI) scenario (Play Application).
- **MainCallflow**--Automatically filled from either session.com.genesyslab.userdata or session.connection.protocol.sip.requesturi based on the Non-CTIC or CTIC flow.
- **SubCallflow**--Automatically filled from the VXML subdialog-invoking methodology.

Variable Versus Static Data

Many blocks enable the use of variables rather than static data. For example, the Prompt block can play the value of a variable as Text-to-Speech. Variables whose values are to be used in other blocks must be declared here so that they appear in the list of available variables in other blocks. The value collected by an Input block or a Menu block is saved as a session variable whose name is the same as the block Name. Also see information on the **AppState variable** used by the DB Data block.

Entry Block Variables

Entry block variables can access User Data (attached data from a routing workflow) from `session.com.genesyslab.userdata` and SIP Request-URI parameters from `session.connection.protocol.sip.requesturi` session variables.

Request URi parameters created in IVR Profiles during the VoiceXML application provisioning are passed to the Composer generated VoiceXML application as request-uri parameters in the `session.connection.protocol.sip.requesturi` session array. An Entry block variable can use these parameters by setting the following expressions to the variable values: `typeof session.connection.protocol.sip.requesturi['var1'] == 'undefined' ? "LocalDefaultValue" : session.connection.protocol.sip.requesturi['var1'].`

If parameters are set as part of IVR Profiles provisioning in the Genesys VoiceXML provisioning system, and if these parameters have the same names as variables set in the Entry block's **Variables** property with the above mentioned `sip.requesturi` expression, then the `SIP-Request-URI` parameters will take precedence over the user variable values set in the Entry block.

Important

For more information on valid values and syntax for the the *gvp.services-parameter* section, refer to page number 121 in the GVP 8.5 User's Guide.

IVR profiles for GVP can be created using the Genesys Administrator. For more information, refer to the Voice Platform Solution Guide and the *gvp.services-parameter* section in the GVP 8.5 User's Guide.

Attaching Results to User Data

While you can assign Classify object results to a variable, Genesys does not recommend this. The recommended way of dealing with the classification results is to attach them to the interaction. Then User Data will have the keys listed in the table below with the corresponding values returned by Classification Server. As an example, User Data would have the following pairs after the attachment:

Parameter	Value
CtgId	00001a05F5U900QW
CtgRelevancy	95
CtgName	Cooking
CtgId_00001a05F5U900QW	95
CtgId_00001a05F5U900QX	85
CtgId_00001a05F5U900QY	75
CtgId_00001a05F5U900QZ	65

VXML Properties

This page provides details about the properties used to manage platform behavior: Note: Properties apply to their parent tag and all the descendants of the parent. A property at a lower level overrides a property at a higher level. If you already have GVP, note that the properties in defaults-ng.vxml will be (re)set as documented below only when a system is newly installed. If you simply upgrade from a previous release, the old values will be preserved. This means that any manual configuration of defaults-ng.vxml will be saved when you upgrade. It also means that when moving to newer versions in which GVP uses different default values, the defaults will not be reset unless you newly install (rather than upgrade).

Receive External Message

Property	*Description*	*Default Value*
com.genesyslab. (GVP extension)	This property specifies whether an external message will be received asynchronously. The valid values are: • True--If the value equals true, external messages will be received asynchronously. • False--If the value equals false, external messages will be received synchronously.	false
com.genesyslab. (GVP extension)	This property specifies whether an external message will be queued or discarded. The valid values are: • True--If the value equals true, external messages will be queued. The external message is reflected to the application in the application.lastmessage$ variable (an ECMAScript object). • False--If the value equals false, external messages will not be delivered as a VoiceXML event (they will be discarded). Note:If no external messages have been received, application.lastmessage$ is ECMAScript undefined. Only the last	false

	received message is available. To preserve a message for future reference during the lifetime of the application, copy the data to an application-scoped variable.	

Speech Recognizer

Property	*Description*	*Default Value*
confidencelevel	Specifies the speech recognition confidence level. Values range from 0.0 (minimum confidence) to 1.0 (maximum confidence). Recognition results are rejected (a nomatch event is thrown) if the confidence level of the results is below this threshold.	0.5
sensitivity	Specifies the level of sensitivity to speech. Values range from 0.0 (least sensitive to noise) to 1.0 (highly sensitive to quiet input).	0.5
speedvsaccuracy	A hint specifying the desired balance between speed versus accuracy when processing a given utterance. Values range from 0.0 (fastest recognition) to 1.0 (best accuracy). Note: The Nuance MRCP engine uses the value of the speedvsaccuracy property to set its proprietary rec.Pruning parameter, using the following algorithm: If x is the speedvsaccuracy value, and x <= 0.5 then rec.Pruning = (x * 400) + 600 else rec.Pruning = (x * 800) + 400	0.5
completetimeout	The length of silence required following user speech before the speech recognizer finalizes a result (either accepting it or throwing a nomatch event). The completetimeout is used when the speech is a complete match of an active grammar and no further words can be spoken.	1s
incompletetimeout	The length of silence required following user speech before the speech recognizer finalizes a result (by either accepting it or throwing a nomatch event). In contrast to completetimeout, the incompletetimeout is used when the speech is an incomplete	1s

	match to an active grammar, or when the speech is a match but it is possible to speak further.	
maxspeechtimeout	The maximum duration of user speech. If this time elapses before the user stops speaking, the maxspeechtimeout event is thrown. Note: Refer to your ASR engine documentation for support details.	60s
	Maximum number of results returned by the recognizer. Also represents the maximum size of the application.lastresult$ array.	1

DTMF Recognizer

Property	*Description*	*Default Value*
interdigittimeout	The timeout period allowed between each digit when recognizing DTMF input.	3s
termtimeout	The terminating timeout to use when recognizing DTMF input.	0s
termchar	The terminating DTMF character for DTMF input recognition.	#
com.genesyslab.dtmf.offboard_recognition (GVP extension)	This property makes it possible to use the DTMF Recognizer that comes with your ASR Engine instead of using the one provided by Genesys. The valid values are: • True--If the value equals true, offboard DTMF recognition is enabled for the call. • False--If the value equals false, offboard DTMF recognition is disabled for the call. Note:If the value is invalid, an error.semantic will be thrown. Note: The recognizer will use the engine specified by the ASR engine property. Note: If you switch engines in mid call, any buffered digits will be lost.	False

Prompt and Collect

Property	*Description*	*Default Value*
inputmodes	Determines which input methods to use. Value is a space separated list of input methods: • dtmf--allows DTMF sequences as input • voice--allows voice as input	dtmf voice
timeout	Once the prompt has finished playing, the length of time to wait, if no speech or dtmf input occurs, before throwing a noinput event.	10s
universals	Specifies universal command grammars to activate. Value is a space-separated list of all or fewer of the following command grammars: • cancel--If this grammar is activated, and the caller says "cancel" (or equivalent phrase configured for another language), the cancel event is thrown. • exit--If this grammar is activated, and the caller says "exit" (or equivalent phrase configured for another language), the exit event is thrown. • help--If this grammar is activated, and the caller says "help" (or equivalent phrase configured for another language), the help event is thrown. A setting of none disables universal commands. A setting of all can be used as a short form for activating all 3 command grammars.	none
com.genesyslab.asrengine (GVP extension)	Specifies the name of the ASR (Automatic Speech Recognition) engine to use. For details about available names, consult with your platform administrator. Note: If this property is not specified, the per call configuration value specified in	platform-specific

	the vxmli.asr.defaultengine property (see the Genesys Voice Platform 8.1 Configuration Options Reference) will be used. The default is empty string (""). Note: It is valid to specify a particular engine only if that engine is installed for the platform running the application. Otherwise, an error.asr.unknownengine event will be thrown. Note: The configured name for SpeechWorks OSR must be speechworks, otherwise a recognition error will occur.	
com.genesyslab.ttsengine (GVP extension)	Specifies the name of the TTS (Text-to-Speech) engine to use (that is, the voice). For details about available names, consult with your platform administrator. Note: If this property is not specified, the per call configuration value specified in the vxmli.asr.defaultengine property (see the Genesys Voice Platform 8.1 Configuration Options Reference) will be used. Note: It is valid to specify a particular engine only if that engine is installed for the platform running the application. Otherwise, an error.tts.unknownengine event will be thrown.	platform-specific
com.genesyslab.endbeep (GVP extension)	Specifies whether a beep should be played at the end of prompts in fields, when bargein is disabled. When bargein is enabled, this attribute has no effect (there is never a beep). Platform owners can access the audio file (endofprompt.vox) in the configured audio path.	false
com.genesyslab.utterancedest (GVP extension)	Specifies the path of the directory to use for saved utterance audio files. The value will be resolved to the configured audio path. This property can be used with the recordutterance property. Note: If you specify the utterancedest and enable the savetmpfiles property, the utterance will only be saved under the utterancedest path. It will not also be saved with the other tmp files.	files are written to the tmp directory (may or may not be saved, depending on whether the savetmpfiles property is enabled)
recordutterance (VoiceXML 2.1 feature)	This property tells the platform to enable recording while simultaneously gathering input from the user. Set to true to enable user utterance to be recorded. Set to false otherwise. Upon completion of user input,	false

	the recording shadow variable will be set. Note: The <vxml> version attribute must be specified as 2.1 (or higher) to use this property. Note: If the recordutterance property has been specified in a VoiceXML 2.0 page, it will behave as if it is a VoiceXML 2.1 page.	
recordutterancetype (VoiceXML 2.1 feature)	This property specifies the audio format to use for recording utterances. Only used with the recordutterance property. GVP currently supports the following types: • audio/basic--Raw (headerless) 8kHz 8-bit mono mu-law [PCM] single channel. (G.711) • audio/x-alaw-basic--Raw (headerless) 8kHz 8-bit mono A-law [PCM] single channel. (G.711) • audio/x-wav--WAV (RIFF header) 8kHz 8-bit mono mu-law [PCM] single channel. • audio/x-wav--WAV (RIFF header) 8kHz 8-bit mono A-law [PCM] single channel.	audio/basic
com.genesyslab.asr.get_swi_literaltimings (GVP extension)	Set to true to allow the special OSR variable, SWI_literalTimings, to be accessed through the application.lastresult$ variable. Requires com.genesyslab.fieldobject to be set to true. Available with SpeechWorks ASR only.	false
com.genesyslab.tts.<Your *vendor specific name*> (GVP extension)	Users will be able to define TTS vendor-specific global properties in the Entry block. The exact set of property names is not known to Composer and therefore no validations will be performed on the names. The general format of these properties will follow this pattern: com.genesyslab.tts.<property_name>	
com.genesyslab.asr.<Your *vendor specific name*> (GVP extension)	When using GVP's MRCP direct integration with an ASR engine, the VoiceXML application can use this property format to specify arbitrary vendor-specific	parameter-specific

	parameters to be sent to the ASR engine. In the property name, <Your *vendor specific name*> is replaced with the actual vendor-specific parameter name; and the value of the property must be a valid value for that vendor-specific parameter. For example, to set Nuance's rec.GrammarWeight parameter to 10: <property name="com.genesyslab.asr.rec.GrammarWeight" value="10"/> Notes: • Vendor parameter names and values could be case-sensitive. Refer to the vendor documentation to ensure you are using valid names and values. • You can only set a vendor parameter using <property> if the parameter can be set by the ASR engine at runtime. Refer to the vendor documentation to confirm which parameters are runtime-settable. • Once a vendor parameter is set using <property>, the setting will stay in effect for the remainder of the call, unless it is set again later in the VoiceXML application.	
swiep_*/swirec_* (GVP extension)	Many of OSR's swiep_*/swirec_* configuration parameters can also be set as VoiceXML properties. To find out whether a particular parameter can be set as a property, look it up in the OSR Reference Manual. If the line under the parameter name includes "API" (and if the description mentions SWIepSetParameter() or SWIrecRecognizerSetParameter()), then it can be set as a property. Some of the parameters that are commonly used are: • swirec_suppress_event_logging • swirec_suppress_waveform_logging • swirec_audio_environment (OSR 2.0+ only) • swirec_backward_compatible_confidence_scores (OSR 2.0+ only)	parameter-specific

	See the OSR Reference Manual for details about the values/usage for each parameter. These properties are specific to Nuance OSR, and are only supported in GVP's *MRCP native* integration with OSR. (They are not supported in GVP's MRCP direct integration with OSR, using SWMS.)	
com.genesyslab.logtoasr (GVP extension)	If set to true, this will enable GVP to log data directly to the ASR engine's log. Note: If this property is true, then the <log> tag's level attribute is ignored.	true

Prompt and Collect--Barge-in

GVP supports Recognition Based Barge-in.

Property	*Description*	*Default Value*
bargein	Controls whether user input can be collected before prompts have finished playing: • true--Any user input can barge in during prompts. • false--No user input can barge in during prompts.	true
bargeinype	Specifies the bargein type: • speech--Any user utterance can barge in the prompt. • hotword (equivalent to recognition)--Only user input that matches a grammar can barge in on the prompt. Note: Not all bargeintypes are supported with all ASR engines.	speech

Prompt and Collect--Wakeup Word Spotting Recognition Mode

In GVP's MRCP native integration with Nuance OSR, OSR's "magic word" feature is exposed through the following properties.

Property	*Description*	*Default Value*
com.genesyslab.wakeupword	Specifies whether Wakeup Word	false

(GVP extension)	Spotting should be used for input in fields, menus, and initials. If set to true, recognition is only performed if input length is between a minimum and maximum length, and (only with Nuance OSR 2.0+) if input matches a grammar.	
com.genesyslab.wakeupwordminimum (GVP extension)	If com.genesyslab.wakeupword is set to true, this specifies the minimum length that input must be in order for recognition to be performed.	
com.genesyslab.wakeupwordmaximum (GVP extension)	If com.genesyslab.wakeupword is set to true, this specifies the maximum length that input may be in order for recognition to be performed.	

Prompt and Collect--Magic Word / Selective Barge-in Recognition Modes

With Nuance SWMS 3.1.4+, OSR's "magic word" and "selective barge-in" features are exposed through the following properties. GVP does not have default values for the following properties. If the application specifies them, GVP passes the specified values through to SWMS. Otherwise, GVP does not pass anything to SWMS - in which case, SWMS would use its own default settings (see the SWMS documentation for these details).

Property	*Description*	*Default Value*
com.genesyslab.ASR.Recognition-Mode (GVP extension)	Set to hotword to enable the OSR *selective barge-in* or *magic word* recognition mode: • Selective Barge-in--Only user input that matches a grammar can barge in on the prompt. (This mode is enabled if com.genesyslab.ASR.Hotword-Max-Duration is set to 0.) • Magic Word--Only user input that matches a grammar, and whose duration is between a minimum and maximum length, can barge in on the prompt. (The minimum and maximum utterance lengths are specified by com.genesyslab.asr.Hotword-	

	Min-Duration and com.genesyslab.asr.Hotword-Max-Duration.) For example: <property name="com.genesyslab.asr.Recognition-Mode" value=""hotword""/> Note: After setting this property, the specified mode will remain in effect for all subsequent recognitions (even if the property is not set in subsequent input fields), unless a new mode is explicitly set. So, to switch back to normal recognition mode after using one of the above hotword modes, the application must explicitly set this property back to normal (and not set any of the three related properties listed below). For example: <property name="com.genesyslab.asr.Recognition-Mode" value=""normal""/> (Available with Nuance SWMS 3.1.4+ only.)	
com.genesyslab.asr.Hotword-Min-Duration (GVP extension)	If com.genesyslab.asr.Recognition-Mode is set to hotword, this specifies the minimum length (in ms) that input must be in order for recognition to be performed. For example: <property name="com.genesyslab.asr.Hotword-Min-Duration" value=""50""/> If com.genesyslab.asr.Hotword-Max-Duration is set to 0, this property will be ignored.	
com.genesyslab.asr.Hotword-Max-Duration (GVP extension)	If com.genesyslab.asr.Recognition-Mode is set to hotword, this specifies the maximum length (in ms) that input may be in order for recognition to be performed. For example: <property name="com.genesyslab.asr.Hotword-Max-Duration" value=""2000""/> If this property is set to 0, the OSR *selective barge-in* mode will be enabled (for example, no minimum and maximum duration constraints are used, so com.genesyslab.asr.Hotword-Min-Duration will be ignored). Otherwise, the OSR *magic word* mode will be enabled (for example, the minimum and maximum duration constraints specified by com.genesyslab.asr.Hotword-Min-Duration and com.genesyslab.asr.Hotword-Max-Duration will be used).	
com.genesyslab.asr.Hotword-Confidence-Threshold	If com.genesyslab.asr.Recognition-	

(GVP extension)	Mode is set to hotword, this specifies the speech recognition confidence level that should be used. Values range from 0 (minimum confidence) to 1000 (maximum confidence). Recognition results are rejected (a nomatch event is thrown) if the confidence level of the results is below this threshold. For this property to take effect, you must also set the standard confidencelevel property to an equivalent decimal percentage. For example: <property name="com.genesyslab.asr.Hotword-Confidence-Threshold" value=""100""/> <property name="confidencelevel" value="0.1"/>	

Fetching

Property	*Description*	*Default Value*
audiofetchhint	Defines when audio files can be fetched: • prefetch--audio file may be downloaded when the page is loaded • safe--only load the audio file when needed Currently, all audio is fetched when needed.	prefetch
audiomaxage	Defines maximum acceptable age, in seconds, of cached audio resources.	undefined
audiomaxstale	Defines maximum staleness, in seconds, of expired cached audio resources.	undefined
datafetchhint	Defines when XML data files can be fetched: • safe--only load the XML data file when needed Currently, all data files are fetched when needed.	safe
datamaxage	Defines maximum acceptable age, in seconds, of cached XML resources.	undefined

datamaxstale	Defines maximum staleness, in seconds, of expired cached XML resources.	undefined
documentfetchhint	Defines when next document can be fetched: • safe--only load the next document when needed Currently, all documents are fetched when needed.	safe
documentmaxage	Defines maximum acceptable age, in seconds, of cached documents.	undefined
documentmaxstale	Defines maximum staleness, in seconds, of expired cached documents.	undefined
grammarfetchhint	Defines when grammar files can be fetched: • prefetch--grammar file may be downloaded when the page is loaded • safe--only load the grammar file when needed Currently, all grammars are fetched when needed.	prefetch
grammarmaxage	Defines maximum acceptable age, in seconds, of cached grammar resources. SpeechWorks OSR 1.x does not support this.	undefined
grammarmaxstale	Defines maximum staleness, in seconds, of expired cached grammar resources. SpeechWorks OSR 1.x does not support this.	undefined
objectfetchhint	Defines when objects can be fetched: • prefetch--object may be downloaded when the page is loaded • safe--only load the object when needed	prefetch
objectmaxage	Defines maximum acceptable age, in seconds, of cached object	undefined

	resources.	
objectmaxstale	Defines maximum staleness, in seconds, of expired cached object resources.	undefined
scriptfetchhint	Defines when scripts can be fetched: • prefetch--script may be downloaded when the page is loaded • safe--only load the script when needed Currently, all scripts are fetched when needed.	prefetch
scriptmaxage	Defines maximum acceptable age, in seconds, of cached script resources.	undefined
scriptmaxstale	Defines maximum staleness, in seconds, of expired cached script resources.	undefined
fetchaudio	The URI of audio to play while waiting for documents to be fetched.	builtin:background_audio.wav
fetchaudiodelay	The length of time to wait at the start of a fetch delay before playing fetchaudio.	1s
fetchaudiominimum	The minimum length of time to play fetchaudio, once started, even if the fetch result arrives in the meantime.	0s
fetchtimeout	Timeout for fetches. This is not supported when using Nuance(MRCP). An error.badfetch is thrown when a fetch duration exceeds fetchtimeout.	30s

Audio Control

The Audio Control Feature is an extension to VoiceXML. **Note**: Audio control functions are only applied to the currently playing prompt, and not across the queued prompt list. Note: These properties may not work properly for TTS. <tbody></tbody>

Property	*Description*	*Default Value*
com.genesyslab.noaudiocontrol (GVP extension)	If this property is set (to any value), the com.genesyslab.audiocontrol property is disabled.	undefined

com.genesyslab.audiocontrol (GVP extension)	(Only used if com.genesyslab.noaudiocontrol is undefined.) Set to true to enable Audio Control during playing of audio. Set to false to disable the feature.	true
com.genesyslab.audio.skipduration (GVP extension)	Sets the duration of audio to be skipped when using the skipahead/skipback features. Note: Time units (s or ms) must be provided.	6000ms
com.genesyslab.audio.skipahead (GVP extension)	Sets the DTMF button for skipping ahead in the audio file/ TTS. The duration skipped depends on the value of the com.genesyslab.audio.skipduration property. If set to "-" or undefined, this feature is disabled.	undefined
com.genesyslab.audio.skipback (GVP extension)	Sets the DTMF button for rewinding the audio file/TTS. The duration rewound depends on the value of the com.genesyslab.audio.skipduration property. If set to - or undefined, this feature is disabled.	undefined
com.genesyslab.audio.louder (GVP extension)	Sets the DTMF button for turning volume up. If set to - or undefined, this feature is disabled. This is not supported with VoIP.	undefined
com.genesyslab.audio.softer (GVP extension)	Sets the DTMF button for turning volume down. If set to - or undefined, this feature is disabled. This is not supported with VoIP.	undefined
com.genesyslab.audio.pause (GVP extension)	Sets the DTMF button for pausing playback temporarily, until the pause button is pressed a second time. If set to- or undefined, this feature is disabled.	undefined
com.genesyslab.audio.stop (GVP extension)	Sets the DTMF button for stopping all queued audio playback. If set to - or undefined, this feature is disabled.	undefined
com.genesyslab.audio.next (GVP extension)	Sets the DTMF button for interrupting the current audio playback, and starting the next audio playback in the queue. If set to - or undefined, this feature is disabled.	undefined
com.genesyslab.audio.faster	Sets the DTMF button for increasing the rate of audio	undefined

(GVP extension)	playback. If set to - or undefined, this feature is disabled. This is not supported with VoIP.	
com.genesyslab.audio.slower (GVP extension)	Sets the DTMF button for decreasing the rate of audio playback. If set to - or undefined, this feature is disabled. This is not supported with VoIP.	undefined

Miscellaneous

Property	*Description*	*Default Value*
com.genesyslab.loglevel (GVP extension)	The loglevel limits execution of <log> tags to the ones whose level attribute have a value up to (including) the loglevel value.	1
com.genesyslab.private	This property enables data masking. This means that private data like credit card numbers, social insurance numbers, and so on are converted to asterisks (for example, 123 would be converted to ***). The valid values are: • True--If com.genesyslab. equals true, data masking is enabled. The data that is masked includes: - asr_trace (result) - dtmf (digit) - input_end (phrase) - prompt _play (all) - subdialog_start (param_value and URL query string) - eval_cond - eval_expr (expression and value) - eval_var (expression and value) - submit (namelist and URL query string) - link (URL query string) - parse_error (URL query string) - wf_arrived (URL query string) - wf_lookup (URL query string) - event_handler_enter (URL query string) - filling (value) - filled_enter (namelist) • False--If com.genesyslab. equals false, data masking is not enabled.	

	Note: The default value is false. Note: This attribute is overridden by the gvp:private attribute (in the <block>, <field>, <transfer>, <record>, <subdialog>, and <initial> tags).	

Platform

The following properties are specific to GVP. The first three are useful for debugging purposes.

Property	*Description*	*Default Value*
com.genesyslab.maintainer.sendwhen	This property indicates if the maintainer email message should be sent. Valid values are: always, never, on_message.	on_message
com.genesyslab.savetmpfiles	The value is interpreted as a string with a list of words. The words may be: all, none, prompts, inputs, pages, recordings. When a list of keywords is specified, the superset of all the keywords are saved. In particular, this means if someone specifies <property name= "com.genesyslab.savetmpfiles" value="none inputs" /> it is equivalent to specifying <property name= "com.genesyslab.savetmpfiles" value="inputs"/>.	none
com.genesyslab.savetmpfilesmode	This property two valid values:immediate or delayed. This property only takes effect when com.genesyslab.savetmpfiles is enabled. If set to immediate the files are written to disk immediately. If set to delayed the files are stored in memory.	immediate
com.genesyslab.onexit.keeptmpfiles	This property specifies whether or not keep temp files around after the VoiceXML session has ended. This property will only have meaning if at least one temp files has been saved. If this value is false, all temp files on the disk will be erased, and any files in memory will be discarded. If this value is true, all temp files on disk will be kept, and files in	true

	memory will be flushed to disk.	
com.genesyslab.maxrecordtime	Defines the default (also the upper limit) for the maxtime attribute of the <record> tag.	10 minutes

Order of Precedence

To find the property value that will take effect at a particular point in an application, the current form item is checked first (to see if the property is defined there), and enclosing scopes are checked as necessary. Here is the full order of precedence for properties:

1. First, look for a property in the current form item (for example, in <field>, <record>, <transfer>, and so on.). If found, use its value.
2. If not found, check the current form (for example, lookdirectly under <form> or <menu>). If the property is found, use its value.
3. If not found, check the current document (for example, look directly under <vxml>). If the property is found, use its value.
4. If not found, check the current document's application root document (if specified by <vxml application="..."> in the current document). If the property is found, use its value.
5. Finally, if not found in any of the above, use the setting from the interpreter context for the current call, which includes the settings in the defaults file (for example, defaults.vxml) and hard-coded default values that are used if no value is configured anywhere else.

Voice Block Palette Reference

When you create a voice application, You use the callflow diagram blocks, located on the palette, to develop voice applications. The palette contains the link tools, and various categories of blocks used to build callflow diagrams:

- **Basic Blocks**
- **Database Blocks**
- **Computer Telephony Integration (CTI) Blocks**
- **External Message Block**s
- **Reporting Blocks**
- **Server-Side Blocks**
- **Outbound Blocks**

The Common Blocks section describes blocks that can be used by both routing and voice applications.

Voice Blocks Basic

The Basic Blocks provide the VoiceXML element functionalities used to perform IVR activities and GVP platform extensible object elements:

Basic Blocks

Block Name	Usage
Assign	Assign a computed value/expression or an entered value to a variable
Branching	Specify multiple application routes based on a branching condition
Disconnect	Explicitly hang-up a phone call
End FCR	Indicate the end of a recording segment
Entry	Begin an application. Only one Entry block can be present in each application.Sets global error (exception) handlers. Defines all global application-level properties, global variables (which appear in the list of available variables for other blocks in the diagram), and global commands. Sets default application scripts and parameters.
Exit	End the application
Go To	Direct the application to a specific URL
Grammar Menu	Uses Grammar Builder files to determine the input options
Input	Accepts DTMF or speech input from callers
Log	Record information about an application
Looping	Iterate over a sequence of blocks multiple times
Menu	Collects DTMF and/or speech input from the caller
Prompt	Play specific data to the caller
Raise Event	Throw custom events
Record	Record voice input from the caller
Release ASR Engine	Control when the ASR engine(s) being used in the current session will be released
Set Language	Changes the current active language from that set in the Entry block or a previous Set Language block
SNMP	Send SNMP traps from the application using the NGI ‘dest’ extension attribute of the <log> tag
Start FCR	Indicate the start of a recorded audio file
Subdialog	Invoke VoiceXML subdialogs, which are a mechanism for reusing common dialogs and building libraries of reusable applications.
Transfer	Transfer the call to another destination

Use the Link tools to connect the blocks.

Assign Block Common

Use the Assign common block to assign a computed value/expression or an entered value to a variable.

See the Query Services block Service Data property for an example of using the Assign block and Expression Builder to parse a JSON string and assign the service data to a variable.

Note: Function getSIPHeaderValue(headername) returns the SIP header value associated with the given SIP headername. You may wish to use this function with the Assign block.

The Assign block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Assign Data Property

This property assigns a value (expression) to a variable. You select the variable and then enter an expression, either a literal or one created in Expression Builder.

To select a variable and assign a value:

1. Click the **Assign Data** row in the block's property table.
2. Click the ⋯ button to open the Assign Data to Variables dialog box.
3. Click in the **Variable** field to display a down arrow.
4. Click the down arrow and select a variable whose value will be evaluated to determine the branching condition. Default application variables are described in the Property Entry block for voice applications and the Property Entry block for routing applications. You can also use a custom variable.
5. Click under Expression to display the ⋯ button.
6. Click the ⋯ button to open Expression Builder. For examples of how to use Expression Builder, see the Expression Builder topic.

7. Select an operator for the branching condition.Your variable's value will be equal to (==), less than (<), greater than (>). less than or equal to (<=), greater than or equal to (>=) or not equal to (!=) to value you enter in the Expression field.
8. In the Expression field, create a value to compare to the variable's value. Enclose the value in single quotes (' ').
9. Click the button to validate the expression. Syntax messages appear under the Expression Builder title.
10. Click **OK** to close Expression Builder and return to the Assign Data to Variables dialog box.
11. You can make multiple variable/value assignments. Click the **Add** button if you wish to add more assignments and repeat the steps above.

Editing Expressions

To edit an expression:

1. Click its row under Expression in the Assign Data to Variables dialog box. This causes the button to appear.
2. Click the button to re-open Expression Builder where you can edit the expression.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

- For callflows, invalid ECMAScript expressions may raise the following exception event: `error.semantic`.
- For workflows, invalid ECMAScript expressions may raise the following exception events: `error.script.SyntaxError`, and `error.script.ReferenceError`.

You can use custom events to define the ECMAScript exception event handling.

Condition Property

Find this property's details under Common Properties for Callflow Blocksor Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Branching Common Block

The Branching block is used for both routing and voice applications. For an example of using the branching block in a strategy, see the Example Workflow Diagram section in the Creating a New Workflow tutorial. Use the Branching block as a decision point in a callflow or workflow. It enables you to specify multiple application routes based on a branching condition. Depending on which condition is satisfied, the call follows the corresponding application route. A default path is automatically created once the conditions have been defined. If the application cannot find a matching condition, it routes the call to the default flow. **Note:** To support creating multiple views per interaction queue, the Branching block is available when creating an IPD.

Date/Time Functions

You can open Expression Builder from the Condition property and access the following date/time URS functions (**Data Category=URS Functions** > **Data Subcategory=genesys**):

- `_genesys.session.timeInZone(tzone)`
- `_genesys.session.dayInZone(tzone)`
- `_genesys.session.dateInZone(tzone)`
- `_genesys.session.day.Wednesday`
- `_genesys.session.day.Tuesday`
- `_genesys.session.day.Thursday`
- `_genesys.session.day.Sunday`
- `_genesys.session.day.Saturday`
- `_genesys.session.day.Monday`
- `_genesys.session.day.Friday`

The Branching block has the following properties:

Exceptions Property

The Branching block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties for Callflow Blocks Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Disconnect Block

Use the Disconnect block to explicitly hang-up a phone call. It differs from the Exit block as follows:

- When an Exit block is used, if the application was called from a CCXML or CTI application, control is sent back to the calling application.
- In the case of the Disconnect block, the entire call is terminated.

Notes

- The Disconnect block returns values (a list of variables) back to the calling context, such as a CCXML application.
- The Disconnect block has no page exceptions.
- There is also a Disconnect Block for use in routing workflows as described below.
- Use the routing Disconnect block and not this Disconnect block when invoking a callflow as part of a Play Application treatment. *GVP 8.x Integration Guide* states the following: For a URS-centric application, the incoming call arrives at a Routing Point DN configured in the SIP Server switch. A routing strategy loading on the Routing Point executes a Play Application treatment to collect customer input. SIP Server sends an INVITE specifying the URI for the voice application. Media Control Platform executes the application. Customer data is collected, then returned to SIP Server in the BYE message. The routing strategy receives the attached data and determines the next action for the call. The call will return to URS where the call can be disconnected in the strategy.

The Disconnect block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.   Use this Property to specify a reason for the disconnect. The content can be either an ECMAScript expression created in Expression Builder or free-form text. The string should conform to the standard specified in RFC 3326 (http://www.ietf.org/rfc/rfc3326.txt), Reason Header Field for the Session Initiation Protocol (SIP). To use Expression Builder to create the reason:

1. Click under Value to display the button.
2. Click the button to open Expression Builder. For examples of how to use Expression Builder, see the

Expression Builder topic.

Return Values Property

Use this property to specify the variable(s) whose value(s) will be returned once the Disconnect block is executed. To select return variables:

1. Click the Return Values row in the block's property table.
2. Click the button to open the Return Values dialog box.
3. Select individual variables, or click Select all or Deselect all as needed.
4. Click OK to close the Return Values dialog box.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

End FCR Block

Use the End FCR block to indicate the end of a recording segment. There must be a matching End FCR block for each Start FCR block used.

Note: Starting and stopping at tapped points (as marked by the Start FCR block and either EndFCR block or the end of call) depends on the Prompt Queuing feature. For this reason, all Prompts between Start FCR and End FCR should have their Immediate Playback property set to `true`.

The End FCR block has the following properties:

The End FCR block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Entry Block

Use an Entry block to begin an application. Only one Entry block can be present in each application. The Entry block:

- Sets global error (exception) handlers.
- Defines all global application-level properties, global variables (which appear in the list of available variables for other blocks in the diagram), and global commands. See topic Variables in Callflows.
- Sets default application scripts and parameters.
- Accesses Expression Builder.

The Entry block is used as the entry point for a main callflow or a sub-callflow. It contains the list of all the variables associated with the callflow (referred to as global variables). Note: Outlinks starting from the Entry block cannot be renamed or assigned a name through the Properties view. The Entry block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The Entry block has all global exception events, with the defaults of all, connection.disconnect.hangup, and error. Also see Exception Events.

Note on No Input and No Match Events

When selecting exceptions for the Entry block, use both `com.genesyslab.composer.toomanynoinputs` / `com.genesyslab.composer.toomanynomatches` and `noinput/nomatch` to catch all the possible no input and no match events. The selection of com.genesyslab.composer.toomanynoinputs / com.genesyslab.composer.toomanynomatches is required when noinput / nomatch exceeds the maximum retries in the lower block. The selection of noinput / nomatch is required when the lower block does not retry at all.

- `com.genesyslab.composer.toomanynoinputs` occurs when the number of no inputs exceeds the maximum retries in the Menu, Input, DBInput, and Record blocks, and the blocks do not have local noinput exception ports.

- `com.genesyslab.composer.toomanynomatches` occurs when the number of no matches exceeds the maximum retries in the Menu, Input, DBInput, and Record blocks, and the blocks do not have a local nomatch exception port.

Note on error.badfetch.badxmlpage

NGI no longer supports this event. If upgrading an application from an earlier version of Composer that supported this event in its Entry object, you will need to modify that object via the Exceptions property dialog box.

Application Root Property

You have the option to specify a VXML file to be used as an application root document allowing multiple callflows to share variables. Background: Starting with 8.1.1, each Composer Project can have (at most) one root document (VXML file). If a Project has no root document, each callflow is its own stand-alone application. If a Project contains a root document, the set of callflows with Entry blocks that reference that root document make up the application.

- If a callflow or sub-callflow references an application root document, the variables specified in the application root become available for selection in all dialogs in that diagram.
- Variables defined in the application root directly under the <vxml> tag become available as global variables to callflows and sub-callflows that access them.

To select an application root document:

1. Click the Application Root row in the block property table.
2. Click the ⋯ button to open the Select Resource dialog box.
3. Select the VXML file in the Project src folder and click OK.

Global Commands Property

The Global Commands property sets rootmap elements for the entire application. A rootmap element is a phrase (user-defined phrase or external grammar) and/or tone the application reacts to at any time the application is running. Use the Global Commands property to set rootmap elements for the entire application. The application uses these rootmap elements as global grammars (subsets of a spoken language that callers are expected to use) in each Input block. Composer creates one outport for each rootmap element; the outport specifies the application path in the event to which the rootmap element is matched. Use the Entry block Global Commands property to set rootmap elements for a subcallflow as well. Note: The RootMap elements defined in the Entry block do not apply to blocks inside a subcallflow. To add, delete, or arrange global phrases, DTMF keys, and grammars:

1. Click the Global Commands row in the block's property table.

2. Click the button to open the Set Rootmap Commands dialog box.

Fields in Set Rootmap Commands Dialog Box

- Name-- Displays the name of the command.
- DTMF Option--Displays the DTMF key to recognize.
- Phrase-- Displays the phrase to recognize.
- Grammar--Displays the built-in or custom grammar used.

Genesys recommends that you use only the GRXML grammar. Otherwise, GSL support--which is not a part of the VoiceXML 2.1 specification--deprecates over time. **Note:** Built-in grammar support for languages other than U.S. English is dependent on the ASR vendor. Before using this feature, make sure that your ASR Engine supports built-in grammars for your language.

Add Button

Use the Add button to enter global phrases, DTMF keys, and grammars.

1. Click Add to enable Command Details fields.
2. In the Name* box, accept the default name or change it.
3. From the DTMF Option drop-down list, select the global DTMF key.
4. In the Phrase box, type the phrase.
5. In the Grammar drop-down list, select a grammar. The grammar source is the custom or built-in grammar for recognition.

Up/Down Buttons

Use the Up and Down buttons to reorder your rootmap elements. Select the element you want to reposition, and then click Up or Down, as necessary.

Delete Button

To delete a phrase, DTMF key, or grammar entry:

1. Select an entry from the list.
2. Click Delete.

Global Properties Property

This property allows suppression of data within the Nuance 9 platform ASR logs. For more information on this property, see the Properties topic on the Genesys Voice Platform wiki. Use Global Properties to select global settings for VXML properties, Automatic Speech Recognition vendor-specific properties or Text-to-Speech vendor-specific properties. To enter properties and values:

1. Click the Global Properties row in the block's property table.
2. Click the button to open the Global Property Settings dialog box.
3. Click Add to enable the Property Name and Property Value fields.
4. Enter or select a Property Name by doing one of the following:
    - Select the Property Name from the drop-down list, or
    - Type the Property Name in the Property Name field.
5. Enter or select a Property Value by doing one of the following:
    - Select the Property Value from the drop-down list, or
    - Type the Property Value in the Property Value field.
6. Click OK.

Scripts Property

Use the Scripts property for including custom JavaScript includes into the application. The JavaScript functions in the specified .js file can then be used in the Assign or Branching blocks in the expression.

1. For this property, enter the filename of your file (for example: script.js). If there are multiple files to be loaded, you can delimit by using the | character; for example: script1.js|script2.js.
2. Then place the custom ECMAScript file in the Scripts subfolder of your project.

There is also a Global Variable SCRIPTSDIR, which specifies the default folder for the scripts files (and works very similar to VOXFILESDIR for audio files).

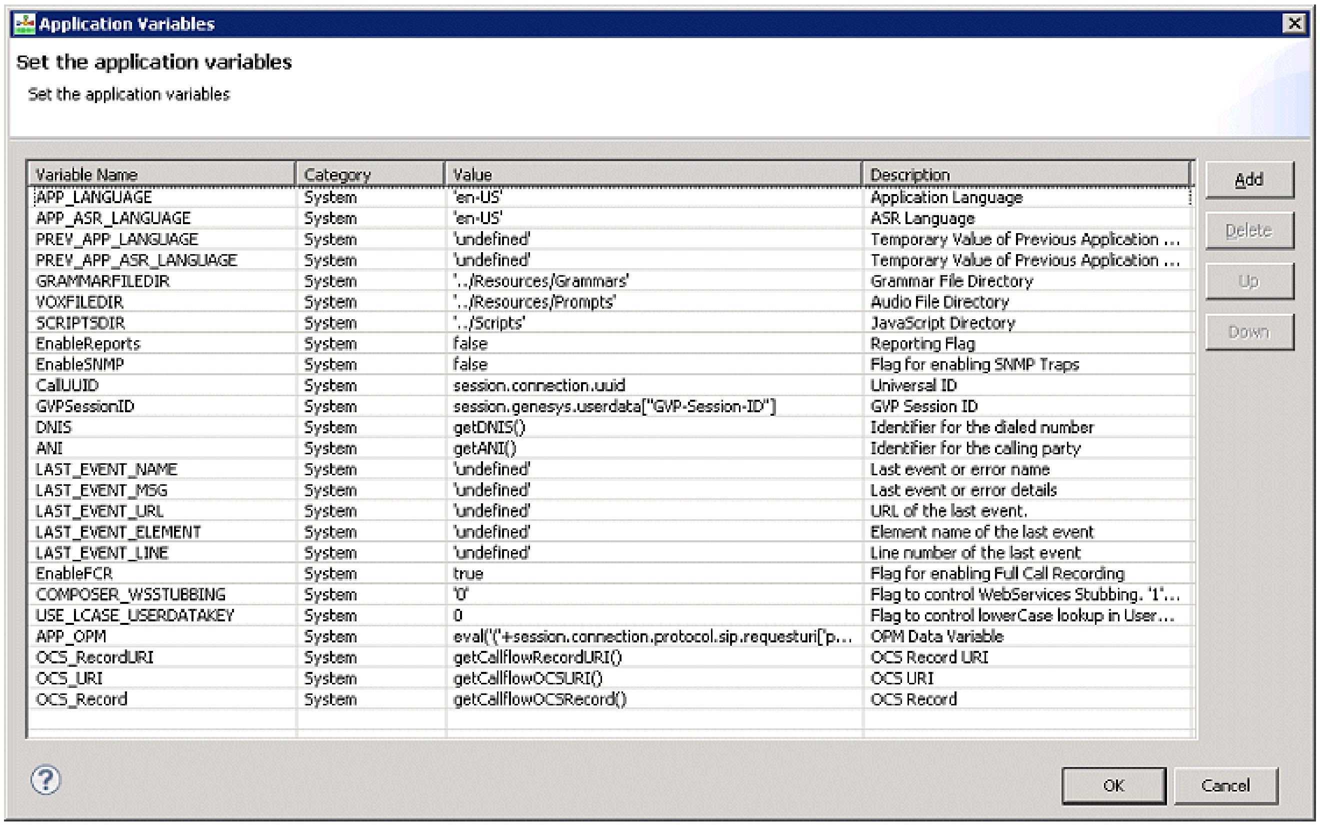
Variables Property

Variables can be predefined system variables (provided by Composer, which you cannot delete) or user-defined variables. See the Variables in Callflows topic for more information. Many Composer blocks have properties that require you to select a variable. **Examples:**

- The following callflow blocks contain a mandatory Output Result property: Menu, Record, DB Input, Grammar Menu, Input, Get Access Number Block, Transfer, and Statistics. After defining variables in the Entry block, you supply this property by selecting the variable to contain the output result.
- When creating a new voice project, a Project-level flag, Enable_ICM, controls whether ICM variables are available for selection and assignment to variables within Composer's Entry block.

To declare for the application or subcallflow:

1. Click the Variables row in the block's property table.
2. Click the button to open the Application Variables dialog box.



Variable Name	Category	Value	Description
APP_LANGUAGE	System	'en-US'	Application Language
APP_ASR_LANGUAGE	System	'en-US'	ASR Language
PREV_APP_LANGUAGE	System	'undefined'	Temporary Value of Previous Application ...
PREV_APP_ASR_LANGUAGE	System	'undefined'	Temporary Value of Previous Application ...
GRAMMARFILEDIR	System	'../Resources/Grammars'	Grammar File Directory
VOXFILEDIR	System	'../Resources/Prompts'	Audio File Directory
SCRIPTSDIR	System	'../Scripts'	JavaScript Directory
EnableReports	System	false	Reporting Flag
EnableSNMP	System	false	Flag for enabling SNMP Traps
CallUUID	System	session.connection.uuid	Universal ID
GVPSessionID	System	session.genesys.userdata["GVP-Session-ID"]	GVP Session ID
DNIS	System	getDNIS()	Identifier for the dialed number
ANI	System	getANI()	Identifier for the calling party
LAST_EVENT_NAME	System	'undefined'	Last event or error name
LAST_EVENT_MSG	System	'undefined'	Last event or error details
LAST_EVENT_URL	System	'undefined'	URL of the last event.
LAST_EVENT_ELEMENT	System	'undefined'	Element name of the last event
LAST_EVENT_LINE	System	'undefined'	Line number of the last event
EnableFCR	System	true	Flag for enabling Full Call Recording
COMPOSER_WSSTUBBING	System	'0'	Flag to control WebServices Stubbing. '1'...
USE_LCASE_USERDATAKEY	System	0	Flag to control lowerCase lookup in User...
APP_OPM	System	eval('('+session.connection.protocol.sip.requesturi['p...	OPM Data Variable
OCS_RecordURI	System	getCallflowRecordURI()	OCS Record URI
OCS_URI	System	getCallflowOCSURI()	OCS URI
OCS_Record	System	getCallflowOCSRecord()	OCS Record

The above figure shows the dialog box after click the Add button. The Value field for the new variable (Var0) contains a button to access Expression Builder. Important! When defining a variable name, the name:

- Must not start with a number or underscore.
- May consist of letters, numbers, or underscores.

When you define and initialize a variable that is expected to be played as a date later on in the callflow, define the value using the following format: yyyyymmdd. Example: MyDate=20090618. You must use this format; Composer does not perform any conversions in this case. When you define and initialize a variable that is expected to be played as a time later on in the callflow, define a 12 hour-based value using the following format: hhmmssa or hhmmssp. Example: MyTime=115900a or MyTime=063700p. Define a 24 hour-based value using the following format: hhmmssh Example: MyTime=192000h. You must use this format; Composer does not perform any conversions in this case. If variables are set as part of provisioning by the Genesys VoiceXML provisioning system, and if these variables have the same names as variables set in the Variables property dialog box, the VoiceXML provisioning system values take precedence over the global variables set here. Many blocks enable the use of variables rather than static data. For example, the Prompt block can play the value of a variable as Text-to-Speech. Variables whose values are to be used in other blocks must be declared here so that they appear in the list of available variables in other blocks. The value collected by an Input block or a Menu block is saved as a session variable whose name is the same as the block Name.

System Variables

These variables apply only to the Entry block, unless otherwise indicated.

- **APP_LANGUAGE**--Holds the application language setting. The value should be the RFC 3066 language tag of an installed language pack. Examples of valid RFC 3066 language tags include en-US and fr-FR. This setting also acts as a default language for the application. This variable may be set using the Set Language block for a multilingual application.
- **APP ASR LANGUAGE**--Holds the language locale for ASR resources. You must define this variable if the application needs to use a different language locale for ASR from TTS resources.
- **GRAMMARFILEDIR**--Gives the relative path from the application to the directory that contains the grammar files. By default, it is set to ../Resources/Grammars. If a voice application supports multiple languages, you can enable the application to switch between them, by changing the value of this variable. In the Subcallflow_Start block, the GRAMMARFILEDIR global variables are not defined by default. This allows the subcallflows to inherit the value of this variable from the main callflow. If the subcallflow overrides this value, the variable can be defined in the Subcallflow_Start block.
- **VOXFILEDIR**--Gives the relative path in the application to the directory that contains the audio files (.vox/.wav). By default, it is set to ../Resources/Prompts. If a voice application supports multiple languages, you can enable the application to switch between them, by changing the value of this variable. In the Subcallflow_Start block, the VOXFILEDIR global variables are not defined by default. This allows the subcallflows to inherit the value of this variable from the main callflow. If the subcallflow overrides this value, the variable can be defined in the Subcallflow_Start block.
- **SCRIPTSDIR**--Default location for JavaScript files
- **EnableReports**--Enables VAR reporting. (Reporting blocks)
- **EnableSNMP**--Enables the SNMP block, if present in the application
- **CallUUID**--Session connection Universal ID
- GVPSessionID--The Genesys Userdata Session ID
- **LAST_EVENT_NAME**--Stores the name of the last event or error that was handled in the Entry block.
- **LAST_EVENT_MSG**--Stores the message of the last event or error that was handled in the Entry block
- **LAST_EVENT_URL**--Stores the URL of the last event or error that was handled in the Entry block.
- **LAST_EVENT_ELEMENT**--Stores the element name of the last event or error that was handled in the Entry block
- **LAST_EVENT_LINE**--Stores the line number of the last event or error that was handled in the Entry block
- **EnableFCR**--A flag for enabling Full Call Recording
- **COMPOSER WSSTUBBING**
- **App_OPM**--Used for fetching OPM parameters. Stores JSON content passed by GVP in session variables. Available throughout the callflow diagram. The OPM block works with this variable by extracting values from it into application variables. Available for main callflows only.
- **OCS_RecordURI**--Used by Outbound blocks. Its default value will be set from userdata passed into the application. For workflows (SCXML): _genesys.ixn.interactions[InteractionID].udata.GSW_RECORD_URI.For callflows: (VXML) session.com.genesyslab.userdata.GSW_RECORD_URI.
- **OCS_URI**--Used by Outbound blocks. Holds the OCS resource path ([http|https]://<host>:<port>). Its

default value will be deduced from OCS_Record_URI. You may change this variable value in order to use a different OCS application for all Outbound blocks in the workflow.

- **OCS_Record**--Used by Outbound blocks. Holds the Record Handle value deduced from OCS_Record_URI.

Note: Request URi parameters created in IVR Profiles during the VoiceXML application provisioning are passed to the Composer generated VoiceXML application as request-uri parameters in the `session.connection.protocol.sip.requesturi` session array. An Entry block variable can use these parameters by setting the following expressions to the variable values: `typeof session.connection.protocol.sip.requesturi['var1'] == 'undefined' ? "LocalDefaultValue" : session.connection.protocol.sip.requesturi['var1']`. If parameters are set as part of IVR Profiles provisioning in the Genesys VoiceXML provisioning system, and if these parameters have the same names as variables set in the Entry block's **Variables** property with the above mentioned `sip.requesturi` expression, then the `SIP-Request-URI` parameters will take precedence over the user variable values set in the Entry block.

Many blocks enable the use of variables rather than static data. For example, the Prompt block can play the value of a variable as Text-to-Speech. Variables whose values are to be used in other blocks must be declared here so that they appear in the list of available variables in other blocks. The value collected by an Input block or a Menu block is saved as a session variable whose name is the same as the block name.

Variable Name

You can use the Variable name field for either of the following purposes:

- To enter the name of a new variable.
- To change the name of an existing variable. To do this, select an existing variable from the list of variables. The variable's name appears in the Variable box, and you can the change its value in the Value box.

Excluded Characters

The Variable name field will not accept the following special characters:

- less-than sign (<)
- greater-than sign (>)
- double quotation mark ()
- apostrophe (‘)
- asterisk (*)
- ampersand (&)
- pound (#)
- percentage (%)
- semi colon (;)
- question mark (?)
- period (.)

The variable Value field will not accept the following special characters:

- less-than sign (<)
- greater-than sign (>)
- double quotation mark ()
- apostrophe (‘)
- ampersand (&)
- plus sign (+)
- minus sign (-)
- asterisk (*)
- percentage (%)

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exit Block

Use the Exit block to end the application. There will usually be an Exit block in every main callflow, unless you use a GoTo block, blind transfer, or other mechanism to end a callflow. Return Mode should be set to false in the main callflow's Exit block. The Exit block is typically connected to the connection.disconnect.hangup exception handler. It is also connected to the last block in the application (for example, when the application wants to play an error message and terminate the call). You can have multiple Exit blocks inside a callflow. The Exit block has no page exceptions.

Using an Exit Block Inside a Subcallflow

The Subdialog block is used to create subcallflows, which are VoiceXML subdialogs. An Exit block terminates the subcallflow application. If the control has to be returned to the main application, then the Return Mode property should be set to true and the user can send a list of parameters to the main call flow as the output parameters. Name

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Reason Property

This property is visible only for subcallflows. Enter a reason for the implicit ActionEnd to be used for VAR reporting.

Return Mode Property

This property is visible only for subcallflows. Click the down arrow under Value and select one of the following:

- true to return control back to the calling callflow.
- false to exit the application.

Return Values Property

Use this property to specify the variable(s) whose value(s) will be returned once the Exit block is executed. To select return variables:

1. Click the Return Values row in the block's property table.
2. Click the ⋯ button to open the Return Values dialog box.
3. Select individual variables (including ICM variables if applicable), or click Select all or Deselect all as needed.
4. Click OK to close the Return Values dialog box.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Result Property

This property is visible only for subcallflows. Click the down arrow and select one of the following to be used for VAR reporting:

- **UNKNOWN**
- **SUCCESS**
- **FAILED**

GoTo Block

Use this block to direct the application to a specific URL. This block enables you to pass parameters in the current application to the URL by selecting them from the User Parameters list. This block is normally used to transfer to another voice application. Genesys recommends that you use subcallflows for modularizing the application and the GoTo block for calling an external application. Note: If an application enables Voice Application Reporting, Genesys recommends that you use subcallflows instead of a GoTo block. The GoTo block has no page exceptions. The GoTo block has the following properties:

Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Parameters

Use to select variables/parameters to pass to the target application. Note: If the Type property is set to ProjectFile, the Parameters property does not apply. To select parameters (Type property is set to URL):

1. Click the Parameters row in the block's property table.
2. Click the button to open the Parameters dialog box.
3. Select individual parameters, or click Select all or Deselect all as needed.
4. Click OK to close the Parameters dialog box.

Type

Sets the type of the destination application. There are two options:

- URL--The destination application can be found at the location specified in the Uri property.
- ProjectFile--The destination can be another location inside the same Composer Project.

To select a value for the Type property:

1. Select the Type row in the block's property table.
2. In the Value field, select URL or ProjectFile from the drop-down list.

URI

Specifies the destination (URL or Composer Project) depending on the value of the Type property. To set a URL destination for the Uri property (Type property is set to URL):

1. Select the Uri row in the block's property table.
2. In the Value field:
    1. - Type a valid URL, which can be specified as a relative path if the file is in the same project (for example, .../src/WSJNews.vxml).
    - Or select a variable from the drop-down list.

To set a Composer Project destination for the Uri property (Type property is set to ProjectFile):

1. Click the Uri row in the block's property table.
2. Click the button to open the Uri dialog box.
3. Select a Voice Project file in the list.
4. Click OK to close the Uri dialog box.

Grammar Menu Block

Creating a Simple Grammar Video

Below is a video tutorial on building a simple grammar with the Grammar Menu block.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```

Link to video

The Grammar Menu block uses Grammar Builder files to determine the input options.

Menu Block Exception Events

The Menu block has eight local exception events.

- error
- error.noresource
- maxspeechtimeout
- noinput
- nomatch
- error.badfetch.grammar.uri
- error.badfetch.grammar.syntax
- error.badfetch.grammar.load

The Grammar Menu block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes

Can be used for both callflow and workflow blocks to add comments.

Exceptions

Find this property's details under Common Properties.

Gbuilder File

A Gbuilder file is created using Grammar Builder and may contain grammar-related information for multiple locales in a proprietary format. The Grammar Menu block can work with the Gbuilder file directly. The Gbuilder File property is used to select a Gbuilder file in the project. This step also selects the particular rule Rule ID to use for the Grammar Menu block. Once specified, the Grammar Menu block creates menu options based on the information contained in the specified Rule ID in the selected Gbuilder file. To select a grammar builder file and rule:

1. Select the Gbuilder File row in the block's property view.
2. Click the ⋯ button to open the GBuilder File dialog box.

Grammar builder files that are defined for this Composer Project are shown in the GBuilder Files pane on the left. These files are usually located in the project folder path: `[VoiceProject] > Resources > Grammars > [ locale] > [gbuilderfile].gbuilder` . Note: Gbuilder files also contain DTMF information.

1. Select a grammar builder file in the left pane.
2. Rules defined for the selected grammar builder file are displayed in the Rules in selected file pane to the right. Select the rule you want to use in this Grammar Menu block, then click OK.

Your selection automatically populates the information for the following three properties: Rule IdRule TagsMenu Options Note: The Grammar Menu block does not pick up changes automatically if you change your Gbuilder file. To synchronize the block with the latest changes, click on the Gbuilder File property. In the popup make sure that the correct Gbuilder file and RuleID are selected. Click OK to close the dialog box. Your diagram will now reflect any menu options changes made in the Gbuilder file.

Rule ID

The Rule Id property is automatically populated with the rule you selected from the Rules in selected file pane in the GBuilder File dialog box. (Refer to the Gbuilder File property.) This is a read-only property in the properties view.

Rule Tags

The Rule Tags property is automatically populated with the specific rule tags that have been defined for the rule you selected from the Rules in selected file pane in the GBuilder File dialog box. (Refer to the Gbuilder File property and Rule Id property.) This is a read-only property in the properties view.

Language

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Grammar Menu block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Menu Mode

To assign a value to the Menu Mode property:

1. Select the Menu Mode row in the block's property table.
2. In the Value field, select DTMF, Voice, or Hybrid from the drop-down list.

The DTMF format indicates the menu option mode of input will be via the telephone keypad. Note: Grammar Builder treats DTMF as another locale. The Voice format indicates the menu option mode of input will be a voice phrase. The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase. Note: If you select the Hybrid menu mode, you will have to provide both voice and DTMF values for all menu options.

Menu Options

The Menu Options property is automatically populated with generated menu items (options) that apply to the selected rule tags in the grammar builder file. You do not modify this property. (Refer to the Gbuilder File property, Rule Id property, and Rule Tags property.) This is a read-only property in the properties view.

Clear Buffer

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications with multiple menus, enabling the caller to key ahead the DTMF digits corresponding to the menu choices. To assign a value to the Clear Buffer property:

1. Select the Clear Buffer row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Interruptible

The Interruptible property does not apply to the Record block. This property specifies whether the caller can interrupt the prompt before it has finished playing. To assign a value to the Interruptible property:

1. Select the Interruptible row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Prompts

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Security

When the Security property is set to true, data for this block is treated as private. GVP will consider the data entered by the caller for this block as sensitive and will suppress it in platform logs and metrics. To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Output Result

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Get Shadow Variables

Shadow variables provide a way to retrieve further information regarding the value of an input item. By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show GrammarMenu1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: GrammarMenu1$.duration. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Number of Retries Allowed

This property determines how many opportunities the user will be provided to re-enter the value. If

Use Last Prompt Indefinitely is set to true, this property has no effect; otherwise, the error.com.genesyslab.composer.toomanynomatches or error.com.genesyslab.composer.toomanynoinputs errors will be raised on reaching the maximum retry limit. To provide a value for the number of retries allowed:

1. Select the Number Of Retries Allowed row in the block's property table.
2. In the Value field, type a value for the number of retries that will be allowed.

Retry Prompts

Find this property's details under Common Properties. A selection can only be made if the Number Of Retries Allowed Property is greater than 0. U

se Last Reprompt Indefinitely

If you set the Use Last Reprompt Indefinitely property to true, the application uses your last re-prompt as the prompt for all further retries. Therefore, the exception handlers that come out for nomatch and noinput are redundant--even if you set the default exceptions that come out as red dots on the left-side of the block. To assign a value to the Use Last Reprompt Indefinitely property:

1. Select the Use Last Reprompt Indefinitely row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Original Prompts

If you set the Use Original Prompts property to true, in the event of an error requiring a retry, the application first plays back the retry error prompt, and then plays back the original prompt for the block (as specified in the Prompts property). To assign a value to the Use Original Prompts property:

1. Select the Use Original Prompts row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Single Counter For Nomatch And Noinput

If you set the Use Single Counter For Nomatch And Noinput property to true, the application maintains a single combined counter for the nomatch and noinput errors. For example, if the block has three nomatch retry messages and three noinput retry messages, the user gets three retry attempts. If you do not select this option, the application generates a total of six retries; and the user gets up to six retry attempts while not exceeding three of each type -- noinput or nomatch. Note: This property not available on the Record block. To assign a value to the Use Single Counter For Nomatch And Noinput property:

1. Select the Use Single Counter For Nomatch And Noinput row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Input Block

The Input block accepts DTMF or speech input from callers. It differs from the Menu block in that it enables taking input that might not belong to a simple choice list (as for the Menu block). It can be used to collect numerical data; for example, phone numbers, account numbers, amounts, or speech data, such as a Stock name. It uses speech or DTMF grammars to define the allowable input values for the user responses. Built-in system grammars are available for data, such as dates and amount. *Note:* Built-in grammars may not be available for all languages. If you specify a language other than U.S. English and refer to an unsupported built-in grammar, a parse error error.unsupported.builtin will be thrown. In case of user input blocks (Menu, Input, Record, Transfer), Composer adds a global variable of type "Block" to the variables list. You can conveniently use this variable for accessing the user input value. Also see Menu Block, Number of Allowed Attempts Exceeded. The Input block has the following properties:

Input Block Exception Events

The Input block has eight exception events:

- error
- error.noresource
- maxspeechtimeout
- noinput
- nomatch
- error.badfetch.grammar.uri
- error.badfetch.grammar.syntax
- error.badfetch.grammar.load

Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Input block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Clear Buffer Property

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications with multiple menus, enabling the caller to key ahead the DTMF digits corresponding to the menu choices. To assign a value to the Clear Buffer property:

1. Select the Clear Buffer row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Interruptible Property

This property specifies whether the caller can interrupt the prompt before it has finished playing. To assign a value to the Interruptible property:

1. Select the Interruptible row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Prompts Property

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Security Property

When the Security property is set to true, data for this block is treated as private. GVP will consider the data associated with this block as sensitive and will suppress it in platform logs and metrics. To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Grammar Type Property

To assign a value to the Grammar Type property:

1. Select the Grammar Type row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:
    - builtinBoolean
    - builtinCurrency
    - builtinDate
    - builtinDigits
    - builtinNumber
    - builtinPhone
    - builtinTime
    - externalGrammar

Note: All the builtinXXX selections are grammars that are provide by the platform or the ASR Engine. Note: Built-in grammar support for locales other than U.S. English is dependent on the ASR vendor. Before using this feature, make sure that your ASR Engine supports built-in grammars for your locale. This feature has the following critical prerequisites:

1. The ASR Engine must support built-in grammars for that language. Contact your ASR Vendor for details.
2. If the ASR Engine supports the language you want to use, then you must install the Language Pack for that language on the GVP Server.

builtinBoolean

Valid inputs include affirmative and negative phrases appropriate to the current locale. DTMF 1 represents " yes," and 2 represents "no." The result is ECMAScript true for yes or false for no. The value is submitted as the string true or the string false.

builtinCurrency

Valid spoken inputs include phrases that specify a currency amount. For DTMF input, the asterisk (*)

character acts as the decimal point. The result is either a string with the format UUUmm.nn, where UUU is the three-character currency indicator according to ISO standard 4217:1995, or null if not spoken by the caller.

builtinDate

Valid spoken inputs include phrases that specify a date, including a month, day, and year. DTMF inputs are: four digits for the year, followed by two digits for the month, and then two digits for the day. The result is a fixed-length date string with format yyyymmdd--for example, 20000704. If the year is not specified, yyyy is returned as ????; if the month is not specified mm is returned as ??; and if the day is not specified dd is returned as ??.

builtinDigits

Valid spoken or DTMF inputs include one or more digits, 0--9. The result is a string of digits.

builtinNumber

Valid spoken inputs include phrases that specify numbers--for example, one hundred twenty-three, or five point three. Valid DTMF input includes positive numbers entered using digits and the star (*) character (to represent a decimal point). The result is a string of digits from 0-9 and that can optionally include a decimal point (.), and/or a plus sign (+) or minus sign (-).

builtinPhone

Valid spoken inputs include phrases that specify a phone number. DTMF star (*) represents x. The result is a string that contains a telephone number consisting of a string of digits and optionally, the character x to indicate a phone number with an extension--for example, 8005551234x789."

builtinTime

Valid spoken inputs include phrases that specify a time, including hours and minutes. The result is a five-character string in the format hhmmx, where x is either a for AM, p for PM, h to indicate a time specified according to the 24-hour clock, or ? to indicate an ambiguous time. Because there is no DTMF convention for specifying AM/PM, in the case of DTMF input, the result is always end with h or ?. If the field value is subsequently used in a prompt, the value is spoken as a time appropriate to the current locale.

externalGrammar

The application can also define an external grammar. The grammars can be written using the GRXML Editor, or GRXML files can be imported into the Composer Project. Note: Look at the User Input Project voice application template in Composer for an example of the use of an external grammar file. Note for Voice Application Developers When developing a VoiceXML application, you must set the web server connection timeout so that it is appropriate to the task that the application performs. It should be greater than one or all of the following callflow applications:

- Maximum talk time
- Maximum recording time

- Maximum wait time for a user input

Input Grammar Dtmf Property

Use the Input Grammar Dtmf (Dual Tone Multi-Frequency as described below) property to specify the DTMF Grammar for the Input Block. The DTMF Grammar is processed and handled by GVP. In the case of external grammars, this specifies the actual path of the grammar file / resource for DTMF Grammars. This is only valid when the Grammar Type is externalGrammar and Input Mode is dtmf or hybrid. To assign a value to the Input Grammar Dtmf property:

1. Select the Input Grammar Dtmf row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables You can specify multiple grammars by separating the grammars with the "|" character. About Dual Tone Multi Frequency (DTMF) Signaling DTMF signaling is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communications devices and the switching center. The version of DTMF used for telephone tone dialing is known by the trademarked term, Touch-Tone. There are some situations where the interpreter (NGI) cannot accept DTMF keypresses immediately as input. In these situations, the keypresses are stored in the DTMF input buffer, for possible later use as input. Throughout the execution of the application, the interpreter must decide whether to save the current contents of the DTMF input buffer (and use them at the next input state), or to discard them. Buffering DTMF input can be useful in allowing typeahead, where users input DTMF for multiple fields rapidly, separated by the termchar. Whatever input is left after the first termchar, may be used in subsequent fields, meaning that the user does not have to wait to hear each prompt.

Input Grammar Voice Property

Use the Input Grammar Voice property to specify the Voice Grammar for the Input block. If you are writing hybrid applications that allow both DTMF and Speech input, specify both the DTMF and Voice grammars. The Voice Grammar is sent to the ASR Engine for processing, whereas the DTMF grammar is processed by GVP. As a result, you need two separate grammars for Voice and DTMF in the case of hybrid applications that allow both Voice and DTMF inputs. In the case of external grammars, this specifies the actual path of the grammar file / resource for ASR Grammars.. This is only valid when Grammar Type is externalGrammar and Input Mode is voice or hybrid. To assign a value to the Input Grammar Voice property:

1. Select the Input Grammar Voice row in the block's property table.
2. In the Value field, select a value from the drop-down list. You can specify multiple grammars by separating the grammars with the "|" character.

Values are the Voice Application Variables described under the Variables Property.

Input Mode Property

To assign a value to the Input Mode property:

1. Select the Input Mode row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:
    - dtmf
    - voice
    - hybrid

DTMF

The DTMF format indicates the menu option mode of input will be via the telephone keypad.

Voice

The Voice format indicates the menu option mode of input will be a voice phrase.

Hybrid

The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase.

Slot Property

The Slot property enables you to specify the slot name of the return value from the grammar. If the slot name is not specified, it is assumed that the grammar will return the value of a slot having the same name as the INPUT block itself. To provide a slot name:

1. Select the Slot row in the block's property table.
2. In the Value field, type a slot name that conforms to the restrictions above.

Input Termination Character Property

The Input Termination Character property defines any character that callers can input in order to indicate that they have finished entering data. For example, the prompt given to the caller may say "Enter your account number, and then press the pound key." The pound key is the input-ending character. To provide a value for the input termination character:

1. Select the Input Termination Character row in the block's property table.
2. In the Value field, type a value for a character to represent the end of the input string.

A typical value that is often used, as indicated above, is: # Example:

- To use # or * then type the value as # or *

Note: Only 1 character can be used as the termination character.

Inter Digit Timeout Property

The Inter Digit Timeout property defines the longest wait time between input characters before a timeout is generated. This is mandatory if dtmf is selected as the Input Mode. Note: Inter Digit Timeout property is applicable only for DTMF input. To provide an Inter Digit timeout value:

1. Select the Inter Digit Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Maximum Input Digits Property

Note: This property only applies if the builtinDigits grammar is selected. The Maximum Input Digits property defines the maximum number of characters that the caller may input. If the input is variable, an input character such as pound sign (#) should be used to terminate the input. This is mandatory if dtmf is selected as the Input Mode. To provide a value for the maximum number of input digits:

1. Select the Maximum Input Digits row in the block's property table.
2. In the Value field, type a value for the maximum number of input digits.

Minimum Input Digits Property

Note: This property only applies if the builtinDigits grammar is selected. The Minimum Input Digits property defines the minimum number of characters that the caller must input. This is mandatory if dtmf is selected as the Input Mode. To provide a value for the minimum number of input digits:

1. Select the Minimum Input Digits row in the block's property table.
2. In the Value field, type a value for the minimum number of input digits.

Get Shadow Variables Property

Shadow variables (optional) provide a way to retrieve further information regarding the value of an input item. By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show Input1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: Input1$.duration. Shadow variables can provide platform-related information about the interaction/input. For example, for speech recognition, this may be the confidence level the platform receives from the ASR engine about how closely the engine could match the user utterance to specified grammar. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Number Of Retries Allowed Property

The Number Of Retries Allowed property determines how many opportunities the user will be provided to re-enter the value. If Use Last Prompt Indefinitely is set to true, this property has no effect; otherwise, the error.com.genesyslab.composer.toomanynomatches or error.com.genesyslab.composer.toomanynoinputs errors will be raised on reaching the maximum retry limit. To provide a value for the number of retries allowed:

1. Select the Number Of Retries Allowed row in the block's property table.
2. In the Value field, type a value for the number of retries that will be allowed.

Retry Prompts Property

Find this property's details under Common Properties.

Use Last Reprompt Indefinitely Property

If you set the Use Last Reprompt Indefinitely property to true, the application uses your last reprompt as the prompt for all further retries. In this case the NoMatch and NoInput exception handlers will never get executed, as the retry loop keeps executing forever. To assign a value to the Use Last Reprompt Indefinitely property:

1. Select the Use Last Reprompt Indefinitely row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Original Prompts Property

If you set the Use Original Prompts property to true, in the event of an error requiring a retry, the application first plays back the retry error prompt, and then plays back the original prompt for the block (as specified in the Prompts property). To assign a value to the Use Original Prompts property:

1. Select the Use Original Prompts row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Single Counter For Nomatch And Noinput Property

If you set the Use Single Counter For Nomatch And Noinput property to true, the application maintains a single combined counter for the nomatch and noinput errors. For example, if the block has three nomatch retry messages and three noinput retry messages, the user gets three retry attempts. If you do not select this option, the application generates a total of six retries; and the user gets up to six retry attempts while not exceeding three of each type -- noinput or nomatch. Note: This property not available on the Record block. To assign a value to the Use Single Counter For Nomatch And Noinput property:

1. Select the Use Single Counter For Nomatch And Noinput row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Log Common Block

Use a Log block to record information about an application. For example, you can log caller-recorded input collected while an application is running or error messages. You can use the Log block for any of the following purposes:

1. Informational – To log the application specific data
2. Error – for logging error details
3. Warning – to flag any warnings
4. Debug – for debugging

The categories in the Log Level property correspond to the above.

When used for a callflow, the Log block writes the log to the Genesys Voice Platform logs (Media Control Platform) using the VoiceXML <log> tag.

The Log block has the following properties:

The Log block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

For callflows, invalid ECMAScript expressions may raise the following exception events: error.semantic. For workflows, invalid ECMAScript expressions may raise the following exception events:

- `error.log.ReferenceError`
- `error.illegalcond.SyntaxError`
- `error.illegalcond.ReferenceError`

You can use custom events to define the ECMAScript exception event handling.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Label Property

This property applies to workflows only. It provides meta-data for the logging details.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Looping Common Block

Use this block to iterate over a sequence of blocks multiple times in the following scenarios:

1. Iterate over a sequence of blocks based on a self-incrementing counter (FOR).
2. Iterate indefinitely until an exit condition is met (WHILE).
3. Iterate over records/data returned by the DB Data block (CURSOR/FOREACH). Also, populate variables if variables mapping is defined.
4. Iterate over data returned by Context Services blocks (FOREACH). Also, populate variables if Variables Mapping is defined.
5. Iterate over JSON Array defined in the application.

For scenarios 1 and 2 above, use the Looping block with a reference to the block retrieving the data. Scenarios 3 and/or 4 can be used in conjunction with 1 or 2, in which case the loop will exit when either of the exit conditions is met.

Prerequisite

You must perform the following steps in order for the Looping block to be used to iterate over data returned by the DB Data block:

1. Create a folder named Scripts in the Project folder.
2. In the Entry block, specify a value for the Scripts property such as ../include/DBRecordSetAccess.js

The Looping block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Counter Initial Value Property

A Counter variable controls the number of loops. Specify the initial value by entering a positive integer (including zero) or selecting the variable that contains the initial value. Composer will increment the Counter variable after each iteration. The value of the Counter variable is available after the looping has exited. This is a mandatory property if the Counter Variable property is specified.

Counter Variable Property

Enter a name for the variable used to store the Counter value or select the variable that contains the name. This is a mandatory property if the Counter Initial Value property is specified.

Current Record Variable Property

Select a variable to be used to store the current record when iterating over records. Composer will assign the current record after each iteration. This property is ignored if the Data Source Property is not set

Data Source Property

Specify a block reference to the DB Data or a Context Services block (with Variables Mapping support) that provides the data to be iterated or select the variable that contains a JSON Array. This is a mandatory property if Counter Initial Value and Counter Variable are not specified.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Counter Max Value Property

Specify the maximum value by entering a positive integer greater than the initial value or selecting the variable that contains the maximum value. When the Counter variable reaches the maximum value, then the block connected to the Exit port is executed. This is a mandatory property if the Counter Variable property is specified or if the Data Source property is not specified.

Exit Expression Property

This property is optional. If specified, prior to each iteration the exit expression is evaluated. If true, the flow goes out via the Exit port of the block. This condition is used in conjunction with max records (if Data Source is specified) or Counter Max Value (if Counter Variable is specified). To enter an exit expression

1. Opposite the **Exit Expression** property, click under **Value** to display the button.
2. Click the button to open Expression Builder. For examples of how to use Expression Builder, see the Expression Builder topic.
3. Create the exit expression and click **OK**.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

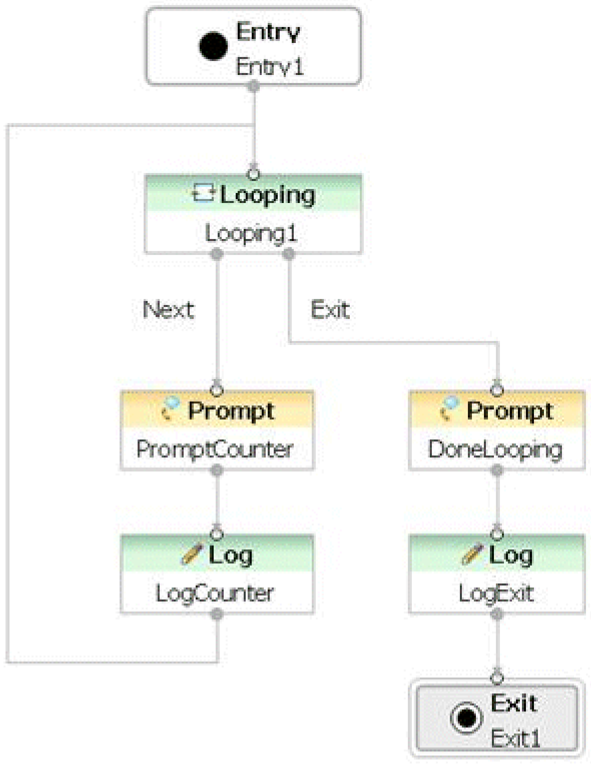
Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Using the Looping Block (Counter-based without a Data Source)

1. Add a Looping block and connect the previous block outport to the Looping block.
2. Connect the Next outport to the sequence of connected blocks.

3. Connect the outport of the last block in the sequence in step 2 back to the looping block to form a loop.
4. Connect the Exit outport of the looping block to the block(s) to continue processing after the loop has exited. The diagram when a looping block is used should appear as follows:



FOR loop: To iterate over the PromptCounter block 10 times, the following properties are set:

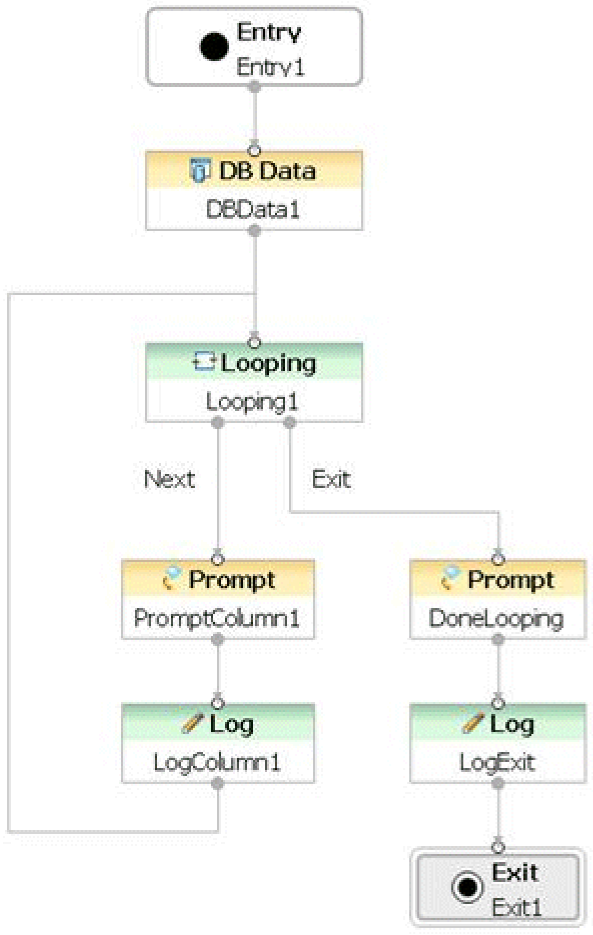
1. Counter Initial Value is 1.
2. Counter Variable Name is Variable(MyCounterVariable).
3. Counter Max Value is 10.

WHILE loop: To iterate over the PromptCounter block until a condition is satisfied, the following property is set: Exit expression is loginSuccessful != true.

Using the Looping Block with a DB Data/Context Services Block

1. Add a Looping block and connect the DB Data/Context Services block outport to the Looping block.
2. Connect the Next outport to the sequence of connected blocks.
3. Connect the outport of the last block in the sequence in step 2 back to the looping block to form a loop.
4. Connect the Exit outport of the looping block to the block(s) to continue processing after the loop has

exited. The diagram when a looping block is used should appear as follows:



CURSOR/FOREACH loop: To iterate over the PromptColumn1 block for each record returned by the DBData1 block, the following property is set: Data Source = Block Reference (DBData1) This example assumes variables were mapped for Column1 in DB Data1. If variables were not mapped, then another Assign block would be needed to store the value into a variable and the variable is then specified in the PromptColumn1 block.

Menu Block

The Menu block collects DTMF and/or speech input from the caller. Typically, you use it for directed input choices (such as selecting to pay bills, get account balances, and so on) so that users are directed to the correct place in the application to perform their transactions, talk to an operator, or other options. In case of speech applications, tones can be associated with phrases to allow either speech or DTMF input from the caller. The phrases and tones are defined in the Menu tab. In case of user input blocks (Menu, Input, Record, Transfer), Composer adds a global variable of type "Block" to the variables list. You can conveniently use this variable for accessing the user input value. The Menu block has the following properties:

Menu Block Exception Events

The Menu block has four local exception events as described in Exception Events.

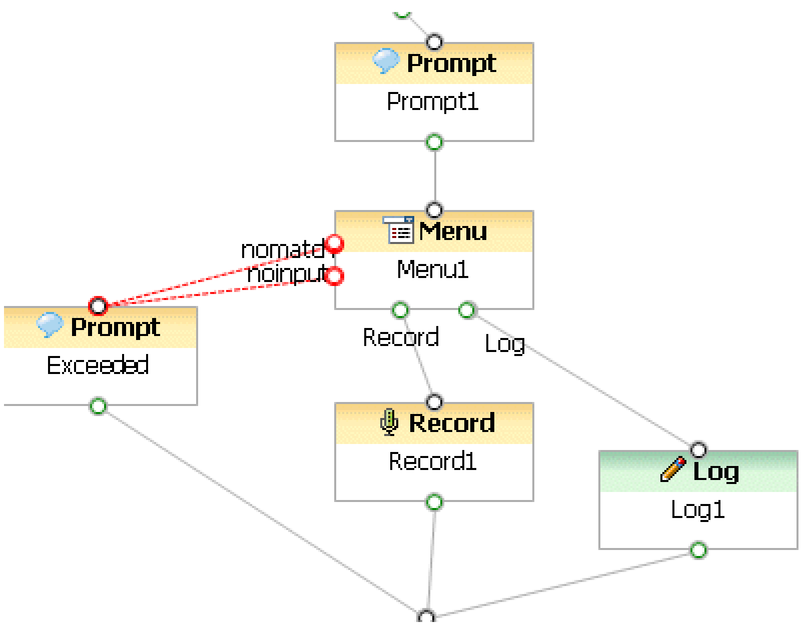
- error
- error.noresource
- noinput
- nomatch

Number of Allowed Retries Exceeded

Assume you need to configure the following use case:

1. User is allowed one invalid entry attempt and one no input attempt. User will then be re-prompted and given a chance to repeat the attempt.
2. When all allowed attempts are exceeded, the user hears a prompt (something like You have exceeded the number of possible retries; please call us later when you have correct information. Good bye).
3. At this point, the call should be terminated (or transferred to an agent or some other action taken.

To handle step 2 during application design: In Menu/Input blocks, move exceptions (nomatch, noinput) to supported events. You can then define the flow path(s) for the case when number of attempts is exceeded. The callflow below illustrates this configuration:



Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Menu block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the

language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Menu Mode Property

To assign a value to the Menu Mode property:

1. Select the Menu Mode row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

DTMF

The DTMF format indicates the menu option mode of input will be via the telephone keypad.

Voice

The Voice format indicates the menu option mode of input will be a voice phrase.

Hybrid

The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase. Note: If you select the Hybrid menu mode, you will have to provide both voice and

DTMF values for all menu options.

Menu Options Property

Use the Menu Options property to add phrases and/or tones to the VoiceMap. To add, delete, or arrange menu options:

1. Click the Menu Options row in the block's property table.
2. Click the button to open the Menu Options dialog box.

Menu Details Fields

Available Menu Details fields depend on the option selected in the Menu Mode property.

For DTMF mode:

- Name*-- Displays the name of the menu option.
- DTMF-Option*--Indicates where the option appears on the menu (1 for first option, 2 for second option, and so on).
- Return Value--Displays the menu option's return value.

For Voice mode:

- Name*-- Displays the name of the menu option.
- Voice-Option*--Allows input of a voice phrase that will be played for the menu option.
- Return Value--Displays the menu option's return value.

For Hybrid mode:

- Name*-- Displays the name of the menu option.
- DTMF-Option*--Indicates where the option appears on the menu (1 for first option, 2 for second option, and so on).
- Voice-Option*--Allows input of a voice phrase that will be played for the menu option.
- Return Value--Displays the menu option's return value.

Menu Options Table

In a new Menu block, four menu options populate the Menu Options table by default. To set or change one of the existing menu options:

1. Select a menu option in the Menu Options table to enable Menu Options fields.
2. In the Name* box, change the default name to a more descriptive name.

3. From the DTMF-Option* drop-down list, select a numeric value to indicate the order that this option will appear in the menu.
4. In the Return Value box, type a return value for this menu option.

Add Button

Use the Add button to add a new menu option.

1. In the Name* box, change the default name to a more descriptive name.
2. From the DTMF-Option* drop-down list, select a numeric value to indicate the order that this option will appear in the menu.
3. In the Return Value box, type a return value for this menu option.

Up/Down Buttons

Use the Up and Down buttons to reorder your menu option elements. Select the element you want to reposition, and then click Up or Down, as necessary.

Delete Button

To delete a menu option:

1. Select an entry from the list.
2. Click Delete.

Repeat Menu Options

Use for specifying a Repeat DTMF key that will cause the menu to be replayed to the caller, from the beginning. The generated VXML will use a <reprompt/> when this DTMF is entered by the caller. Composer's variable support and application root document support allows specifying the same key across blocks. To enable the re-prompting functionality for both DTMF and ASR, you can connect a Menu block outport back to the Menu block itself. To specify:

1. Click the Repeat Menu Options row in the block's property table.
2. Click the ... button to open the Repeat Menu Options dialog box.
3. Click Add.
4. Name the option.
5. Click the down arrow and select a number to indicate where the option appears on the menu (1 for first option, 2 for second option, and so on).
6. Specify the menu option's return value.
7. Click OK.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Security Property

When the Security property is set to true, data for this block is treated as private. GVP will consider the data associated with this block as sensitive and will suppress it in platform logs and metrics. To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Clear Buffer Property

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications the caller is familiar with. For example, the caller hears a welcome prompt but knows the next prompt will solicit the caller's input or menu selection. The caller may start inputting with dtmf while the welcome prompt plays and expect the input to carry forward. To assign a value to the Clear Buffer property:

1. Select the Clear Buffer row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Interruptible Property

This property specifies whether the caller can interrupt the prompt before it has finished playing. To assign a value to the Interruptible property:

1. Select the Interruptible row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Prompts Property

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Get Shadow Variables Property

Shadow variables (optional) provide a way to retrieve further information regarding the value of an input item. By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show Menu1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: Menu1$.duration. Shadow variables can provide platform-related information about the interaction/input. For example, for speech recognition, this may be the confidence level the platform receives from the ASR engine about how closely the engine could match the user utterance to specified grammar. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Number Of Retries Allowed Property

The Number Of Retries Allowed property determines how many opportunities the user will be provided to re-enter the value. If Use Last Prompt Indefinitely is set to true, this property has no effect; otherwise, the error.com.genesyslab.composer.toomanynomatches or error.com.genesyslab.composer.toomanynoinputs errors will be raised on reaching the maximum retry limit. To provide a value for the number of retries allowed:

1. Select the Number Of Retries Allowed row in the block's property table.
2. In the Value field, type a value for the number of retries that will be allowed.

Retry Prompts Property

Find this property's details under Common Properties.

Use Last Reprompt Indefinitely Property

If you set the Use Last Reprompt Indefinitely property to true, the application uses your last reprompt as the prompt for all further retries. Therefore, the exception handlers that come out for nomatch and noinput are redundant--even if you set the default exceptions that come out as red dots on the left-side of the block. To assign a value to the Use Last Reprompt Indefinitely property:

1. Select the Use Last Reprompt Indefinitely row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Original Prompts Property

If you set the Use Original Prompts property to true, in the event of an error requiring a retry, the application first plays back the retry error prompt, and then plays back the original prompt for the block (as specified in the Prompts property). To assign a value to the Use Original Prompts property:

1. Select the Use Original Prompts row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Single Counter For Nomatch And Noinput Property

If you set the Use Single Counter For Nomatch And Noinput property to true, the application maintains a single combined counter for the nomatch and noinput errors. For example, if the block has three nomatch retry messages and three noinput retry messages, the user gets three retry attempts. If you do not select this option, the application generates a total of six retries; and the user gets up to six retry attempts while not exceeding three of each type -- noinput or nomatch. Note: This property not available on the Record block. To assign a value to the Use Single Counter For Nomatch And Noinput property:

1. Select the Use Single Counter For Nomatch And Noinput row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Prompt Block

Use the Prompt block to play specific data to the caller. The Prompt block has no page exceptions. The Prompt block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Prompt block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Clear Buffer Property

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications the caller is familiar with. For example, the caller hears a welcome prompt but knows the next prompt will solicit the caller's input or menu selection. The caller may start inputting with dtmf while the welcome prompt plays and expect the input to carry forward. To assign a value to the Clear Buffer property:

1. Select the Clear Buffer row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Immediate Playback Property

Important! See Note in Timeout section below.

- When Immediate Playback is set to true, prompts are played immediately on the execution of the prompt without queuing them.
- When Immediate Playback is set to false (default), the interpreter goes to the transitioning state and queues the TTS Prompt until the interpreter waits for an input (such as the Menu, Input, Record,and Transfer blocks.

To assign a value to the Immediate Playback property:

1. Select the Immediate Playback row in the block's property table.
2. In the Value field, select true or false from the drop-down list. Selecting false will causes prompts only to be played when waiting for input. Set to false if you want prompts to be played consistent with the VXML default behavior as described below. Otherwise select true to have Composer force immediate playback.

VXML Behavior and Queueing of Prompts

A prompt gets played only when the platform is waiting for input. As described in Voice Extensible Markup Language (VoiceXML) Version 2.0, section 4.1.8, a VoiceXML interpreter is at all times in one of two states:

- waiting for input in an input item (such as <field>, <record>, or <transfer>), or
- transitioning between input items in response to an input (including spoken utterances, dtmf key presses, and input-related events such as a noinput or nomatch event) received while in the waiting state. While in the transitioning state no speech input is collected, accepted or interpreted...

The waiting and transitioning states are related to the phases of the Form Interpretation Algorithm as follows:

- the waiting state is eventually entered in the collect phase of an input item (at the point at which the interpreter waits for input), and
- the transitioning state encompasses the process and select phases, the collect phase for control items (such as <block>s), and the collect phase for input items up until the point at which the interpreter waits for input.

An important consequence of this model is that the VoiceXML application designer can rely on all executable content (such as the content of <filled> and <block> elements) being run to completion, because it is executed while in the transitioning state, which may not be interrupted by input. While in the transitioning state, various prompts are queued, either by the <prompt> element in executable content or by the <prompt> element in form items. In addition, audio may be queued by the fetchaudio attribute. The queued prompts and audio are played either

- when the interpreter reaches the waiting state, at which point the prompts are played and the interpreter listens for input that matches one of the active grammars, or
- when the interpreter begins fetching a resource (such as a document) for which fetchaudio was specified. In this case the prompts queued before the fetchaudio are played to completion, and then, if the resource actually needs to be fetched (i.e. it is not unexpired in the cache), the fetchaudio is played until the fetch completes. The interpreter remains in the transitioning state and no input is accepted during the fetch.

Interruptible Property

This property specifies whether the caller can interrupt the prompt before it has finished playing. To assign a value to the Interruptible property:

1. Select the Interruptible row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Note: For Prompts to be interruptible, there must be a an input block (Menu, Input, etc.) in the execution path. If there are no such blocks further down in the execution path, the Interruptible property has no effect.

Prompts Property

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-

As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Note: Composer does not honor the Timeout setting if you keep the Immediate Playback default setting (=false); for example, where sequential prompts are used. In order for Composer to honor the timeout, you must set Immediate Playback to true.

Raise Event Block

Use the Raise Event block for Composer to throw custom events. You specify the event name and a message, which is selection of a dynamic variable. It is a terminating block (can end an application instead of an Exit block). Orchestration Server 8.1.2+ versions are required for Raise and Cancel Event blocks.

Also see Custom Events.

The Raise Event block has the following properties:

The Raise Event block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Delay Property

Enter the timeout or select a variable. Maps to <send delay>.

Unit Property

Select seconds or milliseconds for the delay. Maps to <send delay>.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Event Property

Select the variable or enter a value. Maps to <send event>.

Parameters Property

Add a list of key-values. Maps to <param>.

Enable Status

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later.

Record Block

The Record block records voice input from the caller. Also see Number of Allowed Attempts Exceeded Property. In case of user input blocks (Menu, Input, Record, Transfer), Composer adds a global variable of type "Block" to the variables list. You can conveniently use this variable for accessing the user input value. The Record block has the following properties: Record Block Exception Events The Record block has four exception events as described in Exception Event Descriptions:

- error
- error.badfetch
- noinput]] (supported by default)
- error.com.genesyslab.composer.recordCapture.failure

Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Record block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Web Server Record File Name Property

User-defined variable (to be assigned) containing the file name of the recorded file located in the folder as specified in the Capture Location property.

1. Select the Web Server Record File Name row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

Prompts Property

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Audio Format Property

This property specifies the audio format for the recording.

1. Select the Audio Format row in the block's property table.
2. In the Value field, select a format value from the drop-down list.

You can modify this value in order to specify enhanced format information such as the codec and the rate as in the following: audio/x-wav;codec=g726;rate=<rate>

Note: You can specify a bit rate as shown in the above example only for the g726 codec.

Beep Before Recording Property

The Beep Before Recording property indicates whether a beep sound will be played for the caller just before recording begins. When set to true, a beep sound will be played; when set to false, no beep will be played. To assign a value to the Beep Before Recording property:

1. Select the Beep Before Recording row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Dtmf Term Character Property

The Dtmf Term Character property defines any character that callers can input in order to indicate that they have finished entering data. For example, the prompt given to the caller may say "Enter your account number, and then press the pound key." The pound key is the Dtmf-ending character. To provide a value for the Dtmf Term Character:

1. Select the Dtmf Term Character row in the block's property table.
2. In the Value field, type a value for a character to represent the end of the Dtmf string.

A typical value that is often used, as indicated above, is: # If several different DTMF tones could be used to indicate the end of data entry, type all values for the supported tones. No separation signs or characters are required. Examples:

- To use # or * then type the value as #*
- If any numeric key could be used for termination, type the value as 1234567890*#

Final Silence Property

The value supplied for the Final Silence property indicates the amount of silence (in seconds) that is allowed to elapse before recording will be stopped. The default value is 3 seconds. To provide a value for the Final Silence property:

1. Select the Final Silence row in the block's property table.
2. In the Value field, type a value for the allowable final silence before recording is stopped.

Max Duration Property

In the context of a Record block, the Max Duration property specifies the maximum recording duration. The default is 60 seconds.

To provide a value for the maximum recording duration:

1. Select the Max Duration row in the block's property table.
2. In the Value field, type a value for the maximum recording duration.

For more information on this property, refer to the Record VXML tag topic in GVP 8.1 Voice XML Help.

Min Duration Property

In the context of a Record block, the Min Duration property specifies the minimum allowed recording duration. The default is 1 second. To provide a value for the minimum recording duration:

1. Select the Min Duration row in the block's property table.
2. In the Value field, type a value for the minimum recording duration.

Capture Filename Property

A value for the Capture Filename property is required when the Capture Filename Type property is set to the value useSpecified. To provide a filename for the captured recording:

1. Select the Capture Filename row in the block's property table.
2. In the Value field, you can:
    - Type a name for the recording file.
    - Click the down arrow and select a variable.

Capture Filename Prefix Property

A value for the Capture Filename Prefix property is required when the Capture Filename Type property is set to the value usePrefix. To provide a prefix for the captured recording filename:

1. Select the Capture Filename Prefix row in the block's property table.
2. In the Value field, you can:
    - Type a value for the recording file prefix.
    - Click the down arrow and select a variable.

Capture Filename Type Property

The Capture Filename Type property indicates the type of the filename for saving the recording. To assign a value to the Capture Filename Type property:

1. Select the Capture Filename Type row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

- **autoGenerate** to auto-generate a recording filename.
- **usePrefix** to add the prefix value specified in the Capture Filename Prefix property to the default name that is generated for the recording.
- **useSpecified** to use the value specified in the Capture Filename property as the filename for the recording. In this case, the file will be overwritten for each call.

Capture Location Property

The Capture Location property specifies the destination path on the Web Application server where the recording is to be saved. If no location is specified, the recordings are saved in the working directory the web application server process. This location may change depending on the web server environment, and therefore, it is recommended that a fixed location is always specified in the Capture Location property. To specify a capture (recording) location:

1. Click the Capture Location row in the block's property table.
2. Type a file path where the recording is to be saved that is located on the web server hosting the application. If the web server is running on Linux, a UNIX-style path can be entered. Composer will not validate the path.

Security Property

When the Security property is set to true, data for this block is treated as private. GVP will consider the data associated with this block as sensitive and will suppress it in platform logs and metrics. To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Last Reprompt Indefinitely Property

If you set the Use Last Reprompt Indefinitely property to true, the application uses your last reprompt as the prompt for all further retries. Therefore, the exception handlers that come out for nomatch and noinput are redundant--even if you set the default exceptions that come out as red dots on the left-side of the block. To assign a value to the Use Last Reprompt Indefinitely property:

1. Select the Use Last Reprompt Indefinitely row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Get Shadow Variables Property

Shadow variables (optional) provide a way to retrieve further information regarding the value of an input item. They can provide platform-related information about the interaction/input. For example, for speech recognition, this may be the confidence level the platform receives from the ASR engine about how closely the engine could match the user utterance to specified grammar. By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show Record1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: Record1$.duration. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Number of Retries Allowed Property

The Number Of Retries Allowed property determines how many opportunities the user will be provided to re-enter the value. If Use Last Prompt Indefinitely is set to true, this property has no effect; otherwise, the error.com.genesyslab.composer.toomanynomatches or error.com.genesyslab.composer.toomanynoinputs errors will be raised on reaching the maximum retry limit. To provide a value for the number of retries allowed:

1. Select the Number Of Retries Allowed row in the block's property table.
2. In the Value field, type a value for the number of retries that will be allowed.

Retry Prompts Property

Find this property's details under Common Properties.

Use Original Prompts Property

If you set the Use Original Prompts property to true, in the event of an error requiring a retry, the application first plays back the retry error prompt, and then plays back the original prompt for the block (as specified in the Prompts property). To assign a value to the Use Original Prompts property:

1. Select the Use Original Prompts row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Release ASR Engine Block

Use the Release ASR Engine block to control when the ASR engine(s) being used in the current session will be released. The Release ASR Engine block has the following properties: The Release ASR Engine block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Engine Name Property

The optional Engine Name property specifies the name(s) of the ASR engine(s) to release. If no engine is specified, all open ASR engines will be released. To specify an ASR engine to release:

1. Select the Engine Name row in the block's property table.
2. In the Value field, type the name of the ASR engine to release.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Set Language Block

The Set Language block changes the current active language from that set in the Entry block or a previous Set Language block. The language specified will be used for all subsequent prompts and grammars. This updates the APP LANGUAGE and APP ASR LANGUAGE global variables to the specified values. All audio and grammar resources will get picked from the specified language folder under the Resource folder of the Composer Project. Set Language is only applicable for audio and grammar files in Composer. Note: Locales that are not defined in Composer must be manually set in the diagram's Assign block. Example: ASR LANGUAGE='te-IN' Also see topic Developing Multi-Lingual Applications. The Set Language block has the following properties: The Set Language block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Language

To set the active language for prompts and grammars:

1. Select the Language row in the block's property table.
2. In the Value field, select one of the following:

- A language from the list of locales defined in the Project settings.
- A variable that contains the active language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

SNMP Block

Use the SNMP block to send SNMP traps from the application. This uses the NGI ‘dest’ extension attribute of the <log> tag. All application-generated SNMP traps are mapped to a single TrapID as defined by the MCP. The EnableSNMP voice application variable is a flag to turn SNMP traps on or off from the SNMP block. The SNMP block has the following properties: The SNMP block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Message Property

The Message property uses a dynamic variable as the message for the SNMP trap. To assign a variable as an SNMP trap:

1. Select the Message row in the block's property table.
2. In the Value field, enter the name of the variable containing the message for the SNMP trap.

The SNMP block will append the following information to the log message:

- session-id
- block name

The format will be : <session-id>::<block-name>::<log message>

Start FCR Block

Use the Start FCR (Start Full Call Recording) block to indicate the start of a recorded audio file. You specify the audio format of the recorded file, which is saved in the MCP folder specified in the Capture Location property. Once recording has started, all interactions will be recorded the End FCR block is reached or the call is terminated Notes:

- Starting and stopping at tapped points (as marked by the Start FCR block and either EndFCR block or the end of call) depends on the Prompt Queuing feature. For this reason, all Prompts between Start FCR and End FCR should have their Immediate Playback property set to true.
- The enableFCR system variable in the Entry block must be set to true in order to use this block.

The Start FCR block has the following properties: The Start FCR block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Audio Format Property

This property specifies the audio format for the recording.

1. Select the Audio Format row in the block's property table.
2. In the Value field, select an audio format value from the drop-down list.

The following audio formats are currently supported:

- audio/vox
- audio/basic
- audio/x-alaw-basic
- audio/x-g726-24
- audio/x-g726
- audio/x-adpcm

- audio/adpcm
- audio/x-adpcm8
- audio/x-g726-40
- audio/L8
- audio/L16
- audio/x-wav
- audio/wav
- audio/x-wav;codec=ulaw
- audio/wav;codec=ulaw
- audio/x-wav;codec=alaw
- audio/wav;codec=alaw
- audio/x-vox
- audio/x-wav;codec=pcm
- audio/wav;codec=pcm
- audio/x-wav;codec=pcm16
- audio/wav;codec=pcm16
- audio/x-wav;codec=g726
- audio/wav;codec=g726
- audio/x-gsm
- audio/x-g729

You can modify this value in order to specify enhanced format information such as the codec and the rate as in the following: audio/x-wav;codec=g729;rate=<rate>

Capture Location Property

The Capture Location property specifies the location for the FCR files on MCP. The default value is ..\callrec, but this value can be changed. To specify a capture (recording) location for the FCR files:

1. Click the Capture Location row in the block's property table.
2. Select the Value field and type a directory path, or keep the default ..\callrec path.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Subdialog Block

Use the Subdialog block for invoking VoiceXML subdialogs, which are a mechanism for reusing common dialogs and building libraries of reusable applications. Subcallflows called from a main callflow encapsulate VXML subdialogs and provide modularization for large VXML applications.   An application can specify the URI of the subdialog to be invoked, pass parameters, and receive output results. Parameters of type In, Out and InOut are supported. You have the option to select how the parameters are to be passed to the invoked subdialog. In the case of Dynamic pages (like JSPs) you can specify the method for sending Get / Post and Use Namelist to indicate the parameters are to be passed as Query String values.

Note: These two choices do not apply in the case of static subdialogs (such as those generated by Composer Voice). The Subdialog block also has the ability to invoke subcallflows created by Composer Voice. In this case, auto-synchronization of input and output parameters is provided. A developer will be able to select a subcallflow to invoke from the current Composer Project.

Also see Using Composer Shared Subroutines.

The Subdialog block has the following properties: The Subdialog block has no page exceptions.

Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Type Property

The Type property sets the type of the invoked subdialog. There are two options:

- URL--The invoked subdialog can be found at the location specified in the Uri property.
- ProjectFile--The invoked subdialog is a subcallflow in the Composer Project.

To select a value for the Type property:

1. Select the Type row in the block's property table.
2. In the Value field, select URL or ProjectFile from the drop-down list.

Uri Property

The Uri property specifies the destination (URL or Composer Project) depending on the value of the Type property. To set a URL destination for the Uri property (Type property is set to URL):

1. Select the Uri row in the block's property table.
2. In the Value field, type a valid URL, or select a variable from the drop-down list.

To set a Composer Project destination for the Uri property (Type property is set to ProjectFile):

1. Click the Uri row in the block's property table.
2. Click the ... button to open the Uri dialog box.
3. Select a callflow in the list.
4. Click OK to close the dialog box.

Composer automatically synchronizes the Input and return variables of the selected sub-callflow with the main callflow by adding them as Input/Output parameters in the corresponding Subdialog Block. Open the Parameters Property of the Subdialog Block to assign the desired value. Note: For a selected studio diagram file, right-click the block's context menu and select the Open Sub Callflow Diagram option to open the chosen Subcallflow diagram file in the Workbench window.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Security Property

When the Security property is set to true, data for this block is treated as private (for example, credit card account numbers, Social Security numbers, date of birth information, and so on). GVP will consider the data associated with this block as sensitive and will suppress it in platform logs and metrics. To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Method Property

This property Indicates the method for invoking the subdialog:

- get--Invoked using HTTP Get
- post--Invoked using HTTP Post. This option is valid only when the parameters are passed as a namelist (Use Namelist property is set to true). This is generally used when a large amount of data needs to be sent as an input value for a subdialog.

To select a value for the Method property:

1. Select the Method row in the block's property table.
2. In the Value field, select get or post from the drop-down list.

Parameters Property

Use the Parameters property to specify parameters to pass to the invoked subdialog. To specify parameters:

1. Click the Parameters row in the block's property table.
2. Click the ⋯ button to open the Parameter Setting dialog box.

If the Type Property is ProjectFile,all the Input/Output parameters are automatically synchronized between the sub-callflow and the main callflow. The Input/Output parameters are automatically added based on the sub-callflow Input/Output parameters. In this case, there are no Add and Delete buttons in the Parameter Setting dialog box as described below. You must fill in the Variables column.

Add Button

Use the Add button to enter parameter details.

1. Click Add to add an entry to SubDialog Parameters.
2. In the Parameter Name field, accept the default name or change it.

3. From the Parameter Type drop-down list, select In, Out, or InOut:

In	Input parameters are variables submitted to the subdialog.
Out	Output parameters are variables that the subdialog returns and will be reassigned back to the current callflow.
InOut	InOut parameters are parameters that act as both input and output.

1. In the Expression drop-down list, select from among the variables shown, type your own expression, or click the button to use Skill Expression Builder.
2. In the Definition field, type a description for this parameter.
3. Click Add again to enter another parameter, or click OK to finish.

Delete Button

To delete a parameter:

1. Select an entry from the list.
2. Click Delete.

Use Namelist

Indicates whether the subdialog parameters need to submitted as a namelist (if set to true) to the called subdialog. To select a value for the Use Namelist property:

1. Select the Use Namelist row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Transfer Block

Use the Transfer block to transfer the call to another destination. By default, blind transfer is enabled, and it has no outports. However, if you enable bridging, the block will have one or more outports. In case of user input blocks (Menu, Input, Record, Transfer), Composer adds a global variable of type "Block" to the variables list. You can conveniently use this variable for accessing the user input value. The Transfer block has the following properties:

Transfer Block Exception Events

The Transfer block has the following exception events as described in Exception Event Descriptions:

- connection.disconect.hangup
- connection.disconnect.transfer (supported by default)
- error (supported by default)
- error.connection.baddestination (supported by default)
- error.connection.noauthorization
- error.connection.noresource
- error.connection.noroute
- error.connection
- error.unsupported.transfer.blind
- error.unsupported.transfer.consultation
- error.unsupported.uri

Name Property

Please find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Transfer block, this property affects the language of grammars used for ASR input:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note: This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Transfer Audio Property

The optional Transfer Audio property plays a prompt to the end user while the number is being dialed out. You provide the URI of the audio source to play while the transfer attempt is in progress (before the other end answers). If the callee answers, the interpreter terminates playback of the recorded audio immediately. If the end of the audio file is reached and the callee has not yet answered, the interpreter plays the audio tones from the far end of the call (ringing, busy). If the resource cannot be fetched, the error is ignored and the transfer continues. To provide a Transfer Audio value:

1. Select the Transfer Audio row in the block's property table.
2. In the Value field, type a Transfer Audio value URI (HTTP or RTSP) specifying the location of the audio file to play.

Aai Property

Use the optional Application-to-Application Information (the Aai property) for the data that is to be transferred from the current application to another application. Use this option to transfer the call to a number that initiates another voice application. To assign a value to the Aai property:

1. Select the Aai row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Authorization Code Property

GVP supports dialing of an authorization code as part of an outbound call on a two-leg transfer. Use free form text to specify the authorization code in the application.

Connect Timeout Property

Use the Connect Timeout property for the connection timeout value. The default is 15 seconds. For information on what happens if a timeout occurs, select Help > Contents and see the GVP 8.1Voice XML 2.1 Reference Help. Specifically see Standard VoiceXML > Variables > Transfer, attribute connecttimeout.   To provide a timeout value:

1. Select the Connect Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Connect When Property

This property controls when the call is connected to the end point. To assign a value:

1. Select the Connect When row in the block's property table.
2. In the Value field, select answered or immediate from the drop-down list.

Destination Property

The Destination property contains the destination phone number. The destination number can be one of the following:

- A Virtual Route point number on which the IRD Strategy is loaded
- Extension number of an Agent
- External number

The value must be specified in one of the formats below:

- sip:[user@]host[:port]
- tel:phonenumber e.g., tel:+358-555-1234567

For information on this property, select Help > Contents and see the GVP 8.1Voice XML 2.1 Reference Help. Specifically see Standard VoiceXML > Variables > Transfer, attribute dest.   To assign a value to the Destination property:

1. Select the Destination row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Max Call Duration Property

Use the Max Call Duration property for the maximum call duration. The default is 3600 seconds. (This is not supported for Consultation Transfer Type.) Note: If this is set to 0 (zero), an infinite value is supplied, and there is no upper limit to the call duration. To provide a value for the maximum call duration:

1. Select the Max Call Duration row in the block's property table.
2. In the Value field, type a value for the maximum call duration.

Transfer Type Property

Specifies the type of the Transfer, which determines whether or not the caller’s session with the VoiceXML interpeter resumes after the call initiated by the transfer ends. Note: Composer also supports AT&T blind transfers with the following options: Out of Band Courtesy, Out of Band Consult, and Out of Band Conference. For more information on these options, start with the GVP 8.1 Voice XML Reference Help (Help > Contents). Search for ATTOOBCOURTESY, ATTOOBCONSULT, and ATTOOBCONFERENCE (Transfer topic). Also see the Genesys Voice Platform 8.1 Deployment Guide. To assign a value to the Transfer Type property:

1. Select the Transfer Type row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

Blind is the default setting. The platform redirects the caller to the agent without remaining in the connection, and it does not monitor the outcome. Once the caller is handed off to the network, the caller's session with the VoiceXML application cannot be resumed. The VoiceXML interpeter throws a connection.disconnect.transfer immediately, regardless of whether the transfer was successful or not. **Bridge** causes the platform add the agent to the connection. Document interpretation suspends until the transferred call terminates. The platform remains in the connection for the duration of the transferred call; listening during transfer is controlled by any included <grammar>s.If the caller disconnects by going onhook or if the network disconnects the caller, the platform throws a connection.disconnect.hangup event. If the agent disconnects, then transfer outcome is set to near_end_disconnect and the original caller resumes her session with the VoiceXML application. **Consultation** causes the consultation transfer to be similar to a blind transfer except that the outcome of the transfer call setup is known and the caller is not dropped as a result of an unsuccessful transfer attempt. When performing a consultation transfer, the platform monitors the progress of the transfer until the connection is established between caller and agent. If the connection cannot be established (e.g. no answer, line busy, etc.), the session remains active and returns control to the application. As in the case of a blind transfer, if the connection is established, the interpreter disconnects from the session, connection.disconnect.transfer is thrown, and document interpretation continues normally. Any connection between the caller and the agent remains in place regardless of document execution. Note: The selected transfer type will work only if the platform is provisioned to support that type of transfer.

Variables Property

This is the list of variables that can be optionally sent by the application as part of the Transfer Request to the far end. It corresponds to the signalvars extension attribute of the NGI VXML Interpreter. Check the NGI VXML Reference Guide for more details. All variables that are selected (checked) will be sent as part of the signalvars . The name of the variable will be used as the key name and the actual value will be the corresponding value. Note: Refer to the GVP Documentation for details on the signalvars attribute. The variable name must match the name of the key that will be sent as signalvars . To declare session variables for the application or subcallflow:

1. Click the Variables row in the block's property table.
2. Click the ... button to open the Variables dialog box.
3. Select individual variables, or click Select all or Deselect all.
4. Click OK.

Method Property

The Method property specifies the type of SIP transfer method that the Media Control Platform (MCP) uses. To assign a value to the Method property:

1. Select the Method row in the block's property table.
2. In the Value field, select one of the following from the drop-down list (descriptions below):

Bridge

A Bridge method indicates that the Media Control Platform (MCP) bridges the media path.

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The transfer fails if a non-2xx final response is received for the INVITE request.

This is a two-leg transfer (in other words, it occupies two channels on the platform). The platform stays in the signaling path and is responsible for bridging the two call legs.

Hkf (Hookflash)

A Hookflash method indicates a transfer using DTMF digits (RFC 2833).

1. The Media Control Platform (MCP) sends DTMF digits on the media channel. The platform leaves it to the media gateway or switch to perform the transfer on the network.
2. Configurable options enable you to specify whether the call will be disconnected by the platform or by the remote end. Otherwise, the call is disconnected after a configured timeout.

This is a one-leg transfer (in other words, it occupies only one channel on the platform).

Refer

A Refer method indicates that the transfer is based on a SIP REFER message (RFC 3515).

1. The platform sends a REFER request to the caller, with the callee (as specified in the VoiceXML application) in the Refer-To: header.
2. The transfer fails if a non-2xx final response is received for the REFER.

This is a one-leg transfer (in other words, it occupies only one channel on the platform).

Referjoin

A Referjoin method indicates a consultative REFER transfer (RFC 3891).

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The platform also sends a REFER request to the caller, with the callee’s information in the Replaces header.
3. The platform considers the transfer to be successful if it receives a BYE from the caller after a 2xx response for the REFER.
4. The transfer fails if a non-2xx final response is received for the INVITE request or for the REFER request.

This is a two-leg, or join-style, transfer (in other words, it occupies two channels on the platform).

Mediaredirect

A Mediaredirect method indicates a media redirection transfer. The Media Control Platform (MCP) uses SIP to handle call control between the caller and the callee, and the RTP media channel is connected directly between the caller and callee.

1. The platform sends an INVITE request to the callee without SDP.
2. If the transfer is proceeding, the callee responds with a 200 OK that includes an SDP offer.
3. The platform forwards the SDP offer in a re-INVITE request to the caller.
4. The caller responds with a 200 OK that includes the SDP answer.
5. The platform forwards the SDP answer to the callee in an ACK response.
6. The transfer fails if a non-2xx final response is received for the initial INVITE request.

This is a two-leg transfer (in other words, it occupies two channels on the platform). attcourtesy attconsult attconference attoobcourtesy attoobconsult attoobconference For information on these methods, consult the section on how the Media Control Platform works in the Genesys Voice Platform 8.1 Deployment Guide.

Disconnect on Answering Machine Property

This property indicates whether or not the FAX / Answering machine has to be detected. To assign a value:

1. Select the Disconnect on Answering Machine row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Do CPA Analysis Property

This property indicates whether or not the platform is enabled to detect who/what answered the call. To assign a value:

1. Select the Do CPA Analysis row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Get Shadow Variables Property

Shadow variables (optional) provide a way to retrieve further information regarding the value of an input item. They can provide platform-related information about the interaction/input. For example, for speech recognition, this may be the confidence level the platform receives from the ASR engine about how closely the engine could match the user utterance to specified grammar. By setting this property to true, it will expose the block's shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block's Logging Details will show Transfer1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: Transfer1$.duration. To assign a value:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Transfer Results Property

There are several types of transfer results supported for applications. When you select a transfer result, a corresponding outport node is added to the block to allow specific actions to be taken for that condition. Please note that a default outport is always present. The default path is executed if none of the selected transfer results are set. The available transfer results are:

- far_end_disconnect (selected by default)
- noanswer (selected by default)
- busy (selected by default)
- near_end_disconnect

Note: Consultation Transfer supports only noanswer, busy, and near_end_disconnect transfer results. To select transfer results:

1. Click the Transfer Results row in the block's property table.
2. Click the ... button to open the Transfer Results dialog box.
3. Select items from the list of available CPA results, or click Select all or Deselect all as needed, then click OK.

Input Grammar Dtmf Property

Use the Input Grammar Dtmf (Dual Tone Multi-Frequency) property to specify the DTMF Grammar for the Transfer block, which accepts DTMF signals or speech input from callers. The DTMF Grammar is processed and handled by GVP. In the case of external grammars, this specifies the actual path of the grammar file / resource for DTMF Grammars. This is only valid when the Grammar Type is externalGrammar and Input Mode is dtmf or hybrid. To assign a value to the Input Grammar Dtmf property:

1. Select the Input Grammar Dtmf row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property. Section 2.3.7.2.1, of the Voice Extensible Markup Language (VoiceXML) Version 2.0 specification (http://www.w3.org/TR/voicexml20/#dml2.3.7.2.1), contains the following information on listening for user input during a transfer (interrupting a transfer): Platforms may optionally support listening for caller commands to terminate the transfer by specifying one or more grammars inside the <transfer> element. The <transfer> element is modal in that no grammar defined outside its scope is active. The platform will monitor during playing of prompts and during the entire length of the transfer connecting and talking phases:

- DTMF input from the caller matching an included DTMF grammar
- an utterance from the caller matching an included speech grammar

A successful match will terminate the transfer (the connection to the callee); document interpretation continues normally. An unsuccessful match is ignored. If no grammars are specified, the platform will not listen to input from the caller. The platform does not monitor in-band signals or voice input from the callee.

Input Grammar Voice Property

Use the Input Grammar Voice property to specify the Voice Grammar for the Input block, which accepts DTMF or speech input from callers. If you are writing hybrid applications that allow both DTMF and Speech input, specify both the DTMF and Voice grammars. The Voice grammar is sent to the ASR Engine for processing, whereas the DTMF grammar is processed by GVP. As a result, you need two separate grammars for Voice and DTMF in the case of hybrid applications that allow both Voice and DTMF inputs. In the case of external grammars, this specifies the actual path of the grammar file / resource for ASR Grammars.. This is only valid when Grammar Type is externalGrammar and Input Mode is voice or hybrid. To assign a value to the Input Grammar Voice property:

1. Select the Input Grammar Voice row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Input Mode Property

To assign a value to the Input Mode property:

1. Select the Input Mode row in the block's property table.
2. In the Value field, select one of the following from the drop-down list (descriptions below):

DTMF

The DTMF format indicates the menu option mode of input will be via the telephone keypad.

Voice

The Voice format indicates the menu option mode of input will be a voice phrase.

Hybrid

The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase.

VXML Form Block

Use this block to embed VXML code directly into a callflow diagram with using <subdialog>.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Body Property

This property contains all the executable content of the <form> element before directing to a block or external application.

1. Click opposite **Body** under **Value**. This brings up the ... button.
2. Click the ... button to bring up the Configure Body dialog box.
3. Enter the executable content of the <form> element. .
4. When through, click **OK**. Note: The editor does not validate against the VXML schema.

Gotostatements Property

This property allows the you to configure the output nodes of the blocks. An output port is created for every GOTOStatement item with target enabled.

1. Click opposite **Gotostatements** under **Value.** This brings up the ▭ button.
2. Click the ▭ button to bring up the Gotostatements dialog box.
3. Click **Add**.
4. When Target is disabled, select **ProjectFile** or **URL** to indicate the destination application type. When ProjectFile is selected, you can click the button to enter the URI. When URL is selected, you can click the URI button and specify a literal or a variable.
5. When URL is selected, you can also click the Parameters button to select a system variable.
6. For each goto statement, specify at least one event, condition, or target (you are not required to complete all three fields). An output port is created for every goto statement.

    - **Name**--Composer uses the name of the goto statement to label the outport.
    - **Event**--Use to select the event that will trigger the goto statement.
    - **Condition**--The guard condition for this goto statement. The goto statement is selected only if the condition evaluates to true.
    - **Target**--If a target is set, an outport for that Goto Statement will appear and you can connect it to other blocks. If a target is not set, an outport for that Goto Statement does not appear; in this case, you can add some VXML code to handle the event.

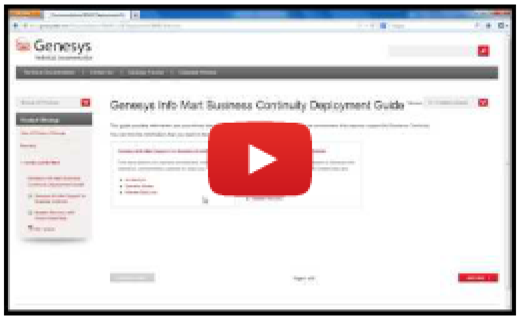
Voice Database Blocks

The Database palette provides blocks that enable VXML applications to use databases.

Video Tutorial

Below is a video tutorial on using the Database Blocks.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```



Using the Database Blocks

Using these blocks, VXML applications can connect to databases and query data from them. It also provides blocks that consume this retrieved data and perform high level operations on it like speaking out the returned data or accepting user input against a grammar generated from the returned data.

Types of Blocks

There are three Database blocks:

- DB Data Block for connecting to a database and retrieving/manipulating information from/in a database.
- DB Prompt Block for speaking out prompts generated using TTS based on the data returned by an associated DB Data block.
- DB Input Block for accepting a DB Data block as its data source and acting as an input field that accepts input based on a grammar created from the results returned from the database.

Also see Working with Database Blocks for an overview of database support in Composer including a high level description of how it works as well as level of support for various databases.

DB Data Block

The DB Data block is used for both routing and voice applications. See the DB Data Block topic in the Common Blocks book. Also see Working with Database Blocks.

Database Input Block

The DB Input block 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.

It accepts DTMF or speech input. This block differs from the Menu block in that it enables taking input that might not belong to a simple choice list (as for the Menu block). The DB Input block can be used to collect numerical data; for example, phone numbers, account numbers, amounts, or speech data, such as a Stock name. It uses speech or DTMF grammars to define the allowable input values for the user responses. Built-in system grammars are available for data, such as dates and amount.

The user input result will be stored in a block name variable in the VXML application.

Note: If the DB Input block uses a DB Data block as its data source, it uses only the first column from returned results to generate the grammar.

The DB Input block can also use a variable as a data source instead of a DB Data block. In this case, grammar for the input is generated based on data in the array. The variable should represent a JSON array similar to the sample below:

myVariable=”[[""Google""],[""Apple""],[""Motorola""],[""Samsung""],[""Nokia""]]”

```
The DB Input block has the following properties:
```

DB Input Block Exception Events

The DB Input block has four exception events as described in Event Descriptions Exception Event Descriptions:

- error
- error.noresource
- noinput
- nomatch

Name Property

Please find this property's details under Property Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Data Source Property

The Data Source property allows you to select the DB Data block that contains a previously-defined database query. This is used when DBDataBlock is selected as the Data Source Type property value. The results of this database query will be used to create the input field.

To select the data source (a DB Data block):

1. Select the **Data Source** row in the block's property table.
2. In the **Value** field, select the appropriate DB Data block from the drop-down list.

Data Source Type Property

The Data Source Type property allows you to select whether your data source is the contents of a DB Data block or a variable.

To select the data source type:

1. Select the **Data Source Type** row in the block's property table.
2. In the **Value** field, select DBDataBlock or Variable from the drop-down list.

Data Source Variables Property

The Data Source Variables property allows you to select the contents of a Property variable as your data source. This is used when Variable is selected as the Data Source Type property value.

To select the variable that serves as your data source:

1. Select the Data Source Variable row in the block's property table.
2. In the Value field, select one of the available variables from the drop-down list. This can also be a custom variable you assigned in the Entry block.

Exceptions Property

Find this property's details under Property Common Properties.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the DB Input block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Clear Buffer Property

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications that the caller is familiar with. For example, the caller hears a welcome prompt but knows the next prompt will solicit the caller's input or menu selection. The caller may start inputting with DTMF while the welcome prompt plays and expect the input to carry forward.

To assign a value to the Clear Buffer property:

1. Select the **Clear Buffer** row in the block's property table.
2. In the **Value** field, select true or false from the drop-down list.

Interruptible Property

This property specifies whether the caller can interrupt the prompt before it has finished playing.

To assign a value to the Interruptible property:

1. Select the **Interruptible** row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Prompts Property

Find this property's details under Property Common Properties.

Note: When Type is set to Value and Interpret-As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block.

To provide a timeout value:

1. Select the **Timeout** row in the block's property table.
2. In the **Value** field, type a timeout value, in seconds.

Security Property

When the Security property is set to true, data for this block is treated as private. GVP will consider the data associated with this block as sensitive and will suppress it in platform logs and metrics.

To assign a value to the Security property:

1. Select the Security row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined Property variable for further processing.

- Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Get Shadow Variables Property

Shadow variables provide a way to retrieve further information regarding the value of an input item.

By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show DBInput1$.)

A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: DBInput1$.duration.

To assign a value to the Get Shadow Variables property:

1. Select the **Get Shadow Variables** row in the block's property table.

2. In the Value field, select true or false from the drop-down list.

Number of Retries Allowed Property

The Number Of Retries Allowed property determines how many opportunities the user will be provided to re-enter the value. If Use Last Prompt Indefinitely is set to true, this property has no effect; otherwise, the error.com.genesyslab.composer.toomanynomatches or error.com.genesyslab.composer.toomanynoinputs errors will be raised on reaching the maximum retry limit.

To provide a value for the number of retries allowed:

1. Select the Number Of Retries Allowed row in the block's property table.
2. In the Value field, type a value for the number of retries that will be allowed.

Retry Prompts Property

Find this property's details under Property Common Properties.

Use Last Reprompt Indefinitely Property

If you set the Use Last Reprompt Indefinitely property to true, the application uses your last reprompt as the prompt for all further retries. In this case the NoMatch and NoInput exception handlers will never get executed, as the retry loop keeps executing forever.

To assign a value to the Use Last Reprompt Indefinitely property:

1. Select the **Use Last Reprompt** Indefinitely row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Use Original Prompts Property

If you set the Use Original Prompts property to true, in the event of an error requiring a retry, the application first plays back the retry error prompt, and then plays back the original prompt for the block (as specified in the Prompts property).

To assign a value to the Use Original Prompts property:

1. Select the **Use Original Prompts** row in the block's property table.
2. In the Value field, select **true** or **false** from the drop-down list.

Use Single Counter for Nomatch And Noinput Property

If you set the Use Single Counter For Nomatch And Noinput property to true, the application maintains a single combined counter for the nomatch and noinput errors. For example, if the block has three nomatch retry messages and three noinput retry messages, the user gets three retry attempts. If you do not select this option, the application generates a total of six retries; and the user gets up to six retry attempts while not exceeding three of each type – noinput or nomatch.

Note: This property not available on the Record block.

To assign a value to the Use Single Counter For Nomatch And Noinput property:

1. Select the Use Single Counter For Nomatch And Noinput row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

DB Prompt Block

The DB Prompt block speaks out prompts generated using TTS based on the data returned by an associated DB Data block. The DB Prompt block will speak each row of the data result set as a sentence. To speak data returned by a DB Data block in a specific format, Genesys recommends using the Prompt block along with ECMA script. A template application (Database Query Result Access Project) is provided which demonstrates the use of ECMA script to allow Prompting of currency and data formats as an example. **DB Prompt Block Tip:** The DB Prompt block speaks out all columns for each record returned by the database as the result of a query. The ordering of columns and of the records is controlled by the query itself and DB Prompt plays them all in the same order without any breaks. To introduce breaks or to add prefix or suffix text to individual columns, you can use a custom query and introduce these features in that query. For example: SELECT ‘name ‘ + employee.firstname + employee.lastname + ‘. . .‘ FROM employee WHERE employee.emp_id < 10 This query will speak out the text name with a small gap before speaking out each name of each employee returned from the database. After each record, it will pause for a small period due to the ‘. . .’ literal in the query. The DB Prompt block has no page exceptions. The DB Prompt block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Data Source Property

The Data Source property allows you to select the DB Data block that contains a previously-defined database query. The results of this database query will be used to create the voice prompt. To select the data source (a DB Data block):

1. Select the Data Source row in the block's property table.
2. In the Value field, select the appropriate DB Data block from the drop-down list.

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only fr the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the DB Prompt block, this property affects the language of grammars of TTS output:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Clear Buffer Property

Use the Clear Buffer property for clearing the DTMF digits in the key-ahead buffer. If it is not set to true, the DTMF digits entered are carried forward to the next block. It is commonly used for applications the caller is familiar with. For example, the caller hears a welcome prompt but knows the next prompt will solicit the caller's input or menu selection. The caller may start inputting with DTMF while the welcome prompt plays and expect the input to carry forward. To assign a value to the Clear Buffer property:

1. Select the Clear Buffer row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Immediate Playback Property

When Immediate Playback is set to true, prompts are played immediately on the execution of the prompt without queuing them. When Immediate Playback is set to false, the interpreter goes to the transitioning state and queues the TTS Prompt until the interpreter waits for an input (such as the Menu, Input, Record,and Transfer blocks). To assign a value to the Immediate Playback property:

1. Select the Immediate Playback row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Interruptible Property

This property specifies whether the caller can interrupt the prompt before it has finished playing. To assign a value to the Interruptible property:

1. Select the Interruptible row in the block's property table.
2. In the Value field, select true,false,or DTMF (for DTMF barge-in mode support) from the drop-down list.

Prompts Property

Find this property's details under Common Properties. Note: When Type is set to Value and Interpret-

As is set to Audio, you can specify an HTTP or RTSP URL. When Type is set to Variable and Interpret-As is set to Audio, you can specify a variable that contains an HTTP or RTSP URL.

Timeout Property

The Timeout property defines the length of the pause between when the voice application plays the last data in the list, and when it moves to the next block. To provide a timeout value:

1. Select the Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

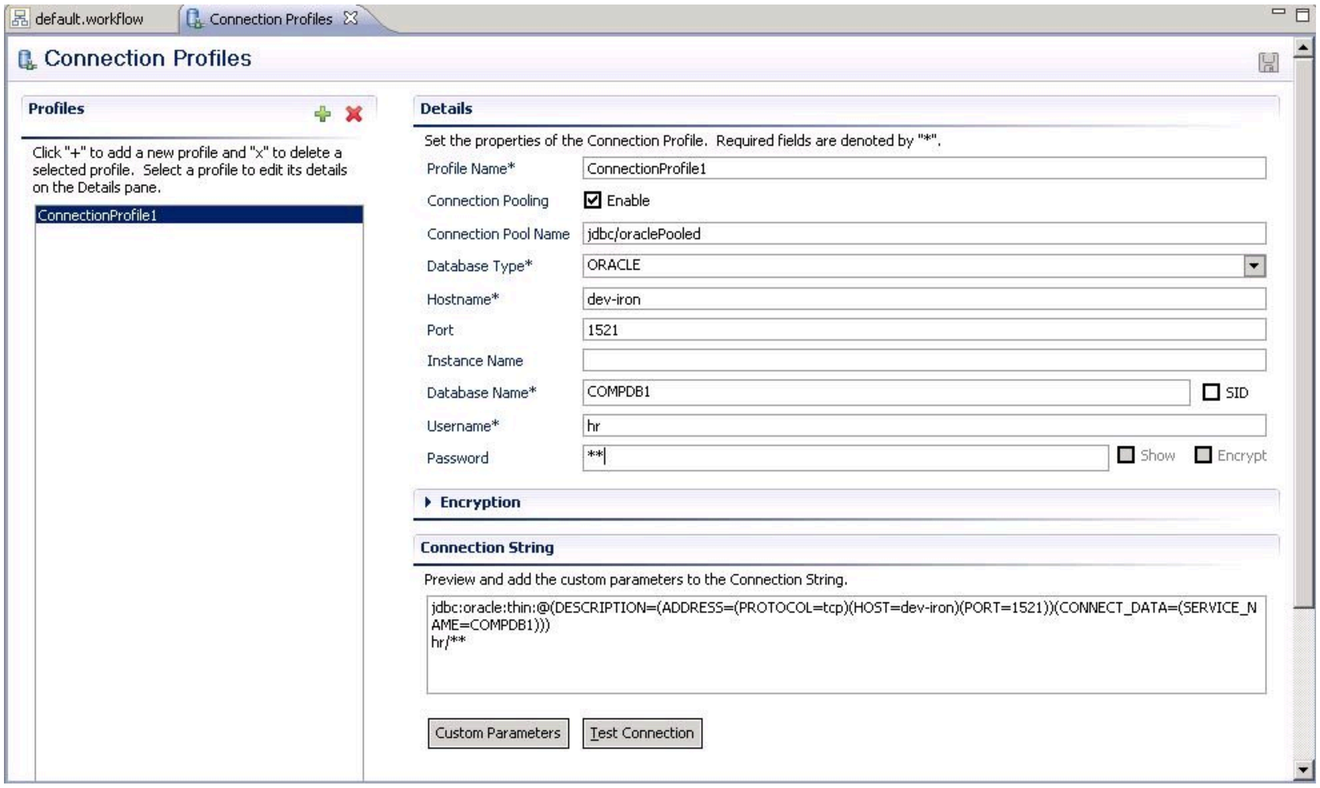
Find this property's details under Common Properties for Callflow Blocks.

Working with Database Blocks

This page contains general information on working with the Database blocks.

Database Connection Profiles

Before you can connect to a database in your application, you need to define a database connection profile that will maintain all information necessary to connect to a particular instance of a database. The figure below shows an example connection profile.



Click "+" to add a new profile and "x" to delete a selected profile. Select a profile to edit its details on the Details pane.

ConnectionProfile1

Set the properties of the Connection Profile. Required fields are denoted by "*".

Profile Name*	ConnectionProfile1
Connection Pooling	☑ Enable
Connection Pool Name	jdbc/oraclePooled
Database Type*	ORACLE
Hostname*	dev-iron
Port	1521
Instance Name	
Database Name*	COMPDB1 ☐ SID
Username*	hr
Password	** ☐ Show ☐ Encrypt

Preview and add the custom parameters to the Connection String.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=dev-iron)(PORT=1521))(CONNECT_DATA=(SERVICE_N
AME=COMPDB1)))
hr/**
```

The DB Data block requires that you specify the name of a connection profile in its properties so that it can use that information to connect to the database at runtime. Multiple connections profiles can be defined in one Project and these profiles can be shared by multiple DB Data blocks even if they are in different callflows. A connection profile consists of the basic information required to connect to a database. The information provided in a connection profile includes the following:

- **Profile Name**. The internal name that Composer uses to identify connections uniquely.
- **Connection Pooling**. Select to enable connection pooling, which maintains a set of database connections that can be reused for requests to databases. You can use this feature to enhance performance by avoiding time-consuming re-establishment of connections to databases.

- **Connection Pool Name**. Specify a Java Naming and Directory Interface (JNDI) name for the pooled data source. Composer applications can use any JNDI data source exposed by the web server. The .war files exported by Composer contain configuration files to support connection pooling with JBoss and WebSphere; other configuration changes to the web application may be required for other web servers
- **Database Type**. The type of database from the list of supported databases
- **Hostname**. The host on which the database server is running. In case of Database Cluster, Virtual IP/ Cluster Alias/SCAN Name is specified here.
- **Port**. The TCP port on which the database server is listening for connections. The most commonly used defaults for supported database types are pre-populated by Composer. If your database server uses custom ports, you will need to specify them here.
- **Instance Name**. The MSSQL Instance that need to connect in SQL Server. Port will take precedence if specified. This field is disabled when Database Type is selected as ORACLE.
- **Database Name**. The name of the database/catalog for SQLServer and the SID in case of Oracle.
- **SID**. The check box to specify if value provided in "Database Name" is SID. This check box is disabled when "Database Type" is MSSQL
- **Username**. The username that should be used to access the database
- **Password**. The password that should be used to access the database
- **Encrypt**. Select the encrypt the password.
- **Show**. Select to show the password
- **Custom Parameters**. The supported custom parameters can be included in connection string along with other parameters. To define custom parameters click on the button "Custom Parameters". In the dialog opened add the parameter name and value, in the order that need to be appended to connection string.

Configuration for Database Cluster:

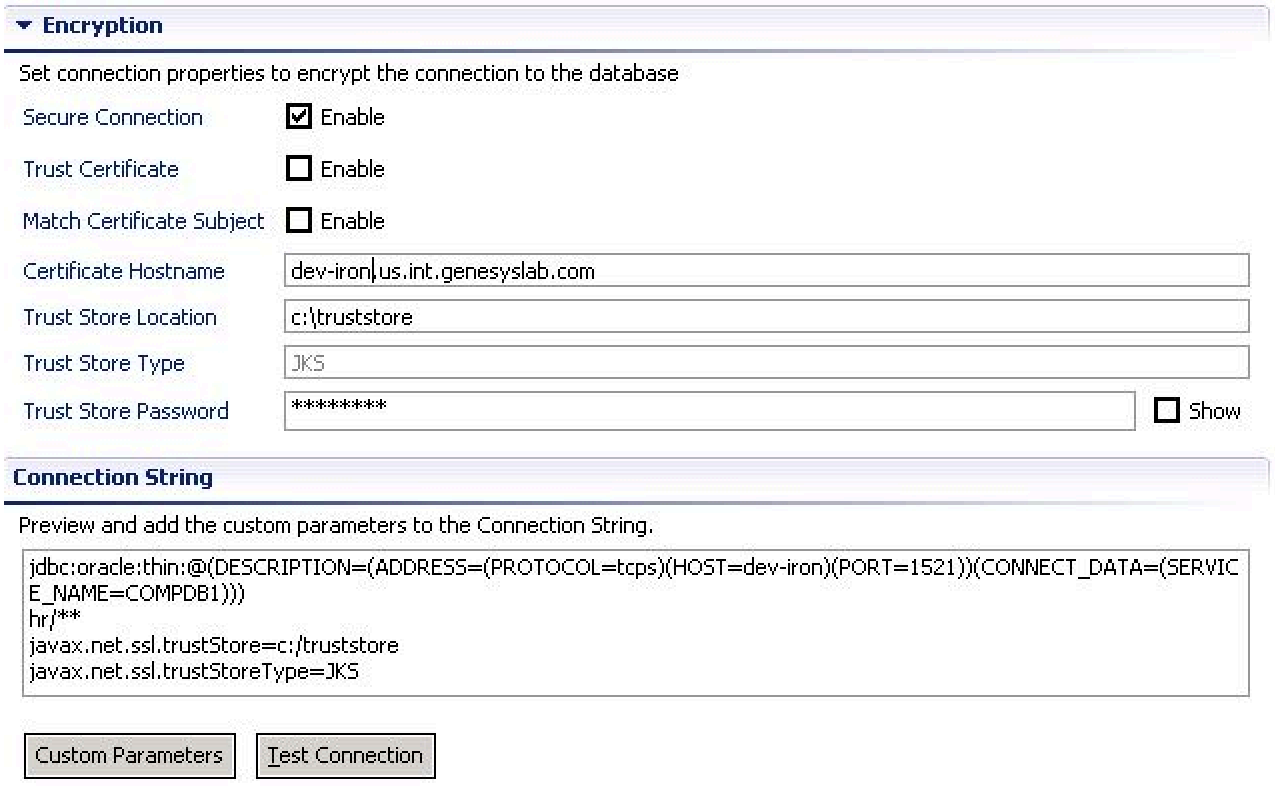
- For MSSQL Cluster, Virtual IP/Cluster Alias is specified in Hostname field of Connection Profile. To connect to particular named instance in cluster, Instance parameter is configured.
- For ORACLE Cluster, Cluster Alias/SCAN Name is specified in Hostname field of Connection Profile.

Additionally, to enable TAF functionality in ORACLE clusters, connection pool is created similar to pooling capability in other application servers. Connection pool can be created as the example below (This need to be added in Tomcat server.xml present in Composer installed path) <Resource name="jdbc/oraclePooled" auth="Container"

```
 type="com.mchange.v2.c3p0.ComboPooledDataSource"
factory="org.apache.naming.factory.BeanFactory"
driverClass="oracle.jdbc.driver.OracleDriver"
user="scott"
password="tiger" jdbcUrl="jdbc:oracle:oci:@(DESCRIPTION=(LOAD_BALANCE=on)(FAILOVER=on)
(ADDRESS=(PROTOCOL=tcp)(HOST=172.21.184.70)(PORT=1521))(ADDRESS=(PROTOCOL=tcp)
(HOST=172.21.184.71)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=rac.genesyslab.com)
(FAILOVER_MODE=(TYPE=session)(METHOD=basic))))" />
```

Encryption:

Parameters under "Encryption" tab allows you to configure SSL encryption and server authentication for Database connections made during Design time (Query Builder, Stored Procedure) and Runtime. When security is enabled, SSL encryption is used for all data sent between composer and SQLServer, if the SQL server has a certificate installed.



Encryption

Set connection properties to encrypt the connection to the database

Secure Connection	[✓] Enable
Trust Certificate	[ ] Enable
Match Certificate Subject	[ ] Enable
Certificate Hostname	dev-iron.us.int.genesyslab.com
Trust Store Location	c:\truststore
Trust Store Type	JKS
Trust Store Password	******** [ ] Show

Connection String

Preview and add the custom parameters to the Connection String.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=dev-iron)(PORT=1521))(CONNECT_DATA=(SERVIC
E_NAME=COMPDB1)))
hr/**
javax.net.ssl.trustStore=c:/truststore
javax.net.ssl.trustStoreType=JKS
```

To establish a Secure Database connection from Composer, following parameters are to be configured under encryption tab:

- **Secure Connection**. Enabling this check box will make all connections from Composer to Database Server encrypted with a choice of server authentication
- **Trust Certificate**. Enabling "Secure Connection" and "Trust Certificate" will be sufficient to establish SSL Connection. When "Trust Certificate" is disabled, other optional attributes are enabled to validate server certificate,
- **Match Certificate Subject**. This is enabled in order to force the matching of the certificate subject available in Server Certificate and client's trusted copy.
- **Certificate Hostname**. This parameter is specified in case the client certificate carries a different subject name than the server certificate and user wishes to ignore the difference by providing the subject name expected in the server certificate explicitly.
- **Trust Store Location**. Location where the Trust Store file is present. The trust store file contains all the certificates trusted by the client, including the certificate that the server uses to autheticate itself.
- **Trust Store Type**. JKS truststore is supported when Database Type is ORACLE. This parameter is not editable. This is not applicable when Database Type is MSSQL
- **Trust Store Password**. Password to access the trust store.

Certificate configuration for Secure Connection:

- For Java Composer Projects, when "Secure Connection" is enabled and "Trust Certificate" is disabled, certificates are placed in "TrustStore Location" specified in connection profile.
- For .NET Composer Projects Design time (i.e. for Query Builder and Stored Procedure Builder), certificates are placed in "TrustStore Location" specified in connection profile.
- For .NET Composer Projects Runtime and MSSQL database, certificates are installed in "Certificate Windows Snap-In" accessed from MMC console in Windows.
- For .NET Composer Projects Runtime and ORACLE database, certificates are installed in Oracle wallet both in client and server. tnsnames.ora configuration will have service name with TCPS protocol. Example is given below.

SSLTEST =

```
  (DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCPS)(HOST = dev-rose.us.int.genesyslab.com)(PORT = 2484))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = SSLTEST)
  )
)
```

Notes:

To establish a connection profile, you must be working with a Project file that was upgraded to Composer 8.0.2 or higher from an earlier Composer release. Connection profiles are not available in Projects created using Composer 8.0. They become available after the Project is upgraded. The method for specifying additional pooling parameters varies based on the database being used and the Project type. Java Composer Projects use the c3p0 library for both SQLServer and Oracle databases. Otherwise, in the case of Oracle databases, Composer uses the c3p0 library and the library exposes its own configuration parameters for pooling via an XML file. In case of SQLServer, additional pooling parameters can be specified in the connection string.

Creating/Editing a Connection Profile

To create (or edit) a connection profile:

1. Select the Project for which you are creating a connection profile in the Project Explorer, and expand your project folder set.
2. Expand the db folder.
3. Double-click the connection.properties file. The Connection Profiles view opens.
4. To create a new profile, click the **Add Profile** icon in the Profiles pane. (If you wish to edit an existing profile, you can select an existing profile in the Profiles pane.)
5. In the Details pane, enter (or update) the appropriate information in each field (fields containing the *

character are required).

6. Click the **Save Profile** icon in the upper-right of the Connection Profiles window. You must save the profile in order for it to be available for selection in the Select Connection Profile dialog box.
7. Test the connection profile by clicking the **Test Connection** button to connect to the database.
    - The message `Database connection was successful` indicates your connection profile successfully connected to the intended database.
    - The message `Database connection failed` followed by additional details indicates a problem with your connection profile. Update the profile, save it, and test it again.

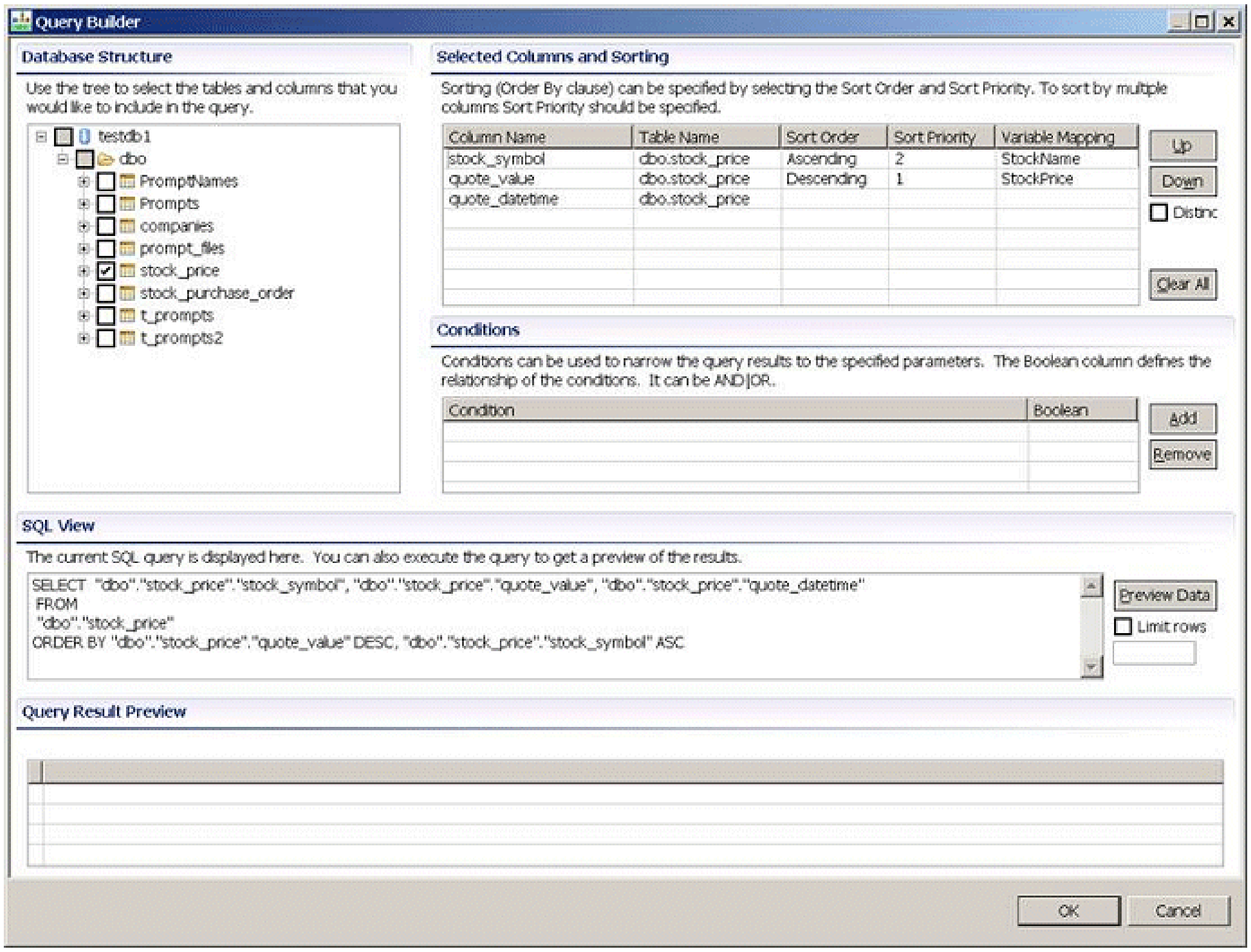
Note: For information on creating the configuration for the connection pool on the application server side, see Connection Pooling.

Preview Connection Strings

The connection to the database with the specified parameters in the connection profile can be previewed and tested in the Connection profile editor. In case of Java project as the design and runtime connections use JDBC connection , JDBC connection string is available to preview and test. In case of Dotnet projects as the design time uses JDBC connection and runtime uses OLEDB conneciton, both strings are available to preview and test. **Note:** The Dotnet project must be deployed correctly in IIS to preview the OLEDB connection string. The parameters apart from ones explicitly collected in the editor can be added using the **custom parameters** dialog which takes the parameters as a name value pair.

Using the Query Builder

The Composer Query Builder provides a visual method of building a database query without the need to type SQL code. The Query Builder is accessed through the Query Type property in the DB Data block. It can be used for both voice callflows and routing workflows. **Note:** The Query Builder can only be accessed when a valid connection profile has been created and selected in the Connection Profile property of the DB Data block. The Query Builder with an example query is shown below.


Query Builder

Database Structure

Use the tree to select the tables and columns that you would like to include in the query.

- testdb1
  - dbo
    - PromptNames
    - Prompts
    - companies
    - prompt_files
    - [✓] stock_price
    - stock_purchase_order
    - t_prompts
    - t_prompts2

Selected Columns and Sorting

Sorting (Order By clause) can be specified by selecting the Sort Order and Sort Priority. To sort by multiple columns Sort Priority should be specified.

Column Name	Table Name	Sort Order	Sort Priority	Variable Mapping
stock_symbol	dbo.stock_price	Ascending	2	StockName
quote_value	dbo.stock_price	Descending	1	StockPrice
quote_datetime	dbo.stock_price			

Conditions

Conditions can be used to narrow the query results to the specified parameters. The Boolean column defines the relationship of the conditions. It can be AND|OR.

Condition	Boolean

SQL View

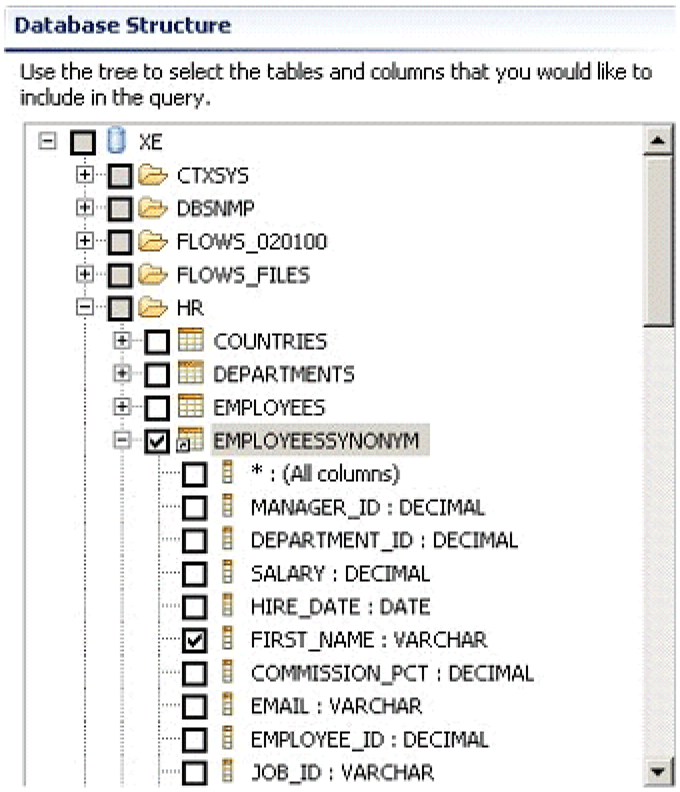
The current SQL query is displayed here. You can also execute the query to get a preview of the results.

```
SELECT  "dbo"."stock_price"."stock_symbol", "dbo"."stock_price"."quote_value", "dbo"."stock_price"."quote_datetime"
 FROM
 "dbo"."stock_price"
ORDER BY "dbo"."stock_price"."quote_value" DESC, "dbo"."stock_price"."stock_symbol" ASC
```

Query Result Preview

Building a Database Query

The Query Builder opens when Composer is successfully able to connect to the database specified in your connection profile. Any schemas, tables (and table synonyms) and columns of the database accessible from the specified user account are shown in hierarchical format in the Database Structure pane of the Query Builder. In the example below, `EMPLOYEESSYNONYM` is a table synonym.


Database Structure

Use the tree to select the tables and columns that you would like to include in the query.

```
XE
  CTXSYS
  DBSNMP
  FLOWS_020100
  FLOWS_FILES
  HR
    COUNTRIES
    DEPARTMENTS
    EMPLOYEES
    [✓] EMPLOYEESSYNONYM
      * : (All columns)
      MANAGER_ID : DECIMAL
      DEPARTMENT_ID : DECIMAL
      SALARY : DECIMAL
      HIRE_DATE : DATE
      [✓] FIRST_NAME : VARCHAR
      COMMISSION_PCT : DECIMAL
      EMAIL : VARCHAR
      EMPLOYEE_ID : DECIMAL
      JOB_ID : VARCHAR
```

TableSyn.gif

Note: MSSQLServer table synonyms are read from the system table sys.synonyms. Oracle table synonyms are read from the system table user_synonyms. To build a query:

1. Specify which table columns are returned as query results.

- Select the tables and columns to include in your query by checking appropriate items in the **Database Structure** pane. Expand table entries to see the columns. To select all columns in a table, select the appropriate (**All columns**) check box under the appropriate table.
- Selected columns and tables appear in the **Selected Columns** pane. To alter the order in which selected columns are returned in query results, use the **Up** and **Down** buttons to reorder columns within the list.
- To specify the order in which query results should be sorted, click on the **Sort Order** field for a column and select a Sort option (**ascending** or **descending**). This will automatically fill in the Sort Order, which indicates the sequence in which multiple sort criteria will be applied. It is possible to sort by multiple columns and you can change the sorting sequence by clicking on the **Sort Priority** column. For example, you might sort a query of names by last name and then sort by first name for those people with the same last name. In that case, last name has Sort Order 1, and first name has Sort Order 2.

Note: The order in which columns appear in the Selected Columns list does not affect the sort order.

- To specify the variables into which the column values need to be copied, click on the **Variable Mapping** field for a column and select a variable. If a variable is specified for a column, DB Data block execution will result in the column values of the first record being copied into the specified variable. If more than one record is returned by the query, then use the Looping block along with the DB Data block to iterate over records and populate the variables specified for the columns.

2. Specify filter criteria. In the **Conditions** pane, you build the search or filter criteria to identify the data you want to retrieve from the database. You can can specify multiple conditions.

    - Click **Add** to create a new condition. A new row will be added to the Conditions list. Click on the **Condition** column, and then click the ... to open the Condition Builder.
    - Select a column from the **Select Column** drop-down list which the search condition will operate on.
    - Select the operator (=, <>, <, >, and so on) from the **Operator** drop-down list. This operator will be used to compare the specified column with the value specified in the next step.</li>
    - In the **Value** field, type or select your value for the condition depending on the value type option:
        - **Column Reference**: a table column that you can select from a drop-down list. This option will compare the two selected columns based on the specified operator.
        - **Application Variable**: a variable defined in your application that can be selected from a drop-down list. At runtime the current value of the selected variable will be used for comparing the column's value based on the specified operator.
        - **Custom Value**: a value that is not validated by the query builder and is added directly to the query. It can be used to specify SQL functions or more complex expression.
        - **Literal**: a value that is interpreted as a string or a number. Type in the literal value. The value will be enclosed in quotes automatically if it is a string. If the literal value represents a number, you will need to enclose it in quotes depending on the data type of the selected column. This option will compare the selected column's value to the specified literal using the specified operator.
    - Click **OK** to complete the condition.
    - Using the above steps, you can define multiple conditions. These conditions can be combined using logical operators to further refine your search criteria. You can select **AND** or **OR** in the **Boolean** field to specify the logical operator.

3. Test your query.

    - To test the query, you can click the **Preview Data** button. This executes the query against the appropriate database. If the database tables contain data and if any records match the specified conditions, they will be displayed in the Query Results Preview pane. A message will also show the number of records returned as a result of the query.
    - If you expect that the number of matching records will be large and want to preview a subset of returned data, click the **Limit Rows** check box and enter a numeric value to limit the number of returned results.

Note: The message will now show the number of records displayed rather than the actual number of matching records. The query results preview is shown in the Query Result pane.

4. Click **OK** to save your query and update the DB Data block with the new query. If you click Cancel, all changes are discarded and no changes are made to the DB Data block.

Specifying Custom Queries

The DB Data block can use queries specified in a SQL (.sql) file in your Project instead of a query created using the Query Builder. To use a custom query:

- Create a .sql file in your db folder and specify the filename in the Query File property of the DB Data block. Make sure that the operation type is **SQLScriptFile**. Composer will read this file at runtime and use it to query the specified database.

The ability to use custom queries is useful in cases where the SQL query is already created using other tools, or if the query uses features not supported by the Visual Query Builder. The next topic describes limitations of the query builder.

Application Variables

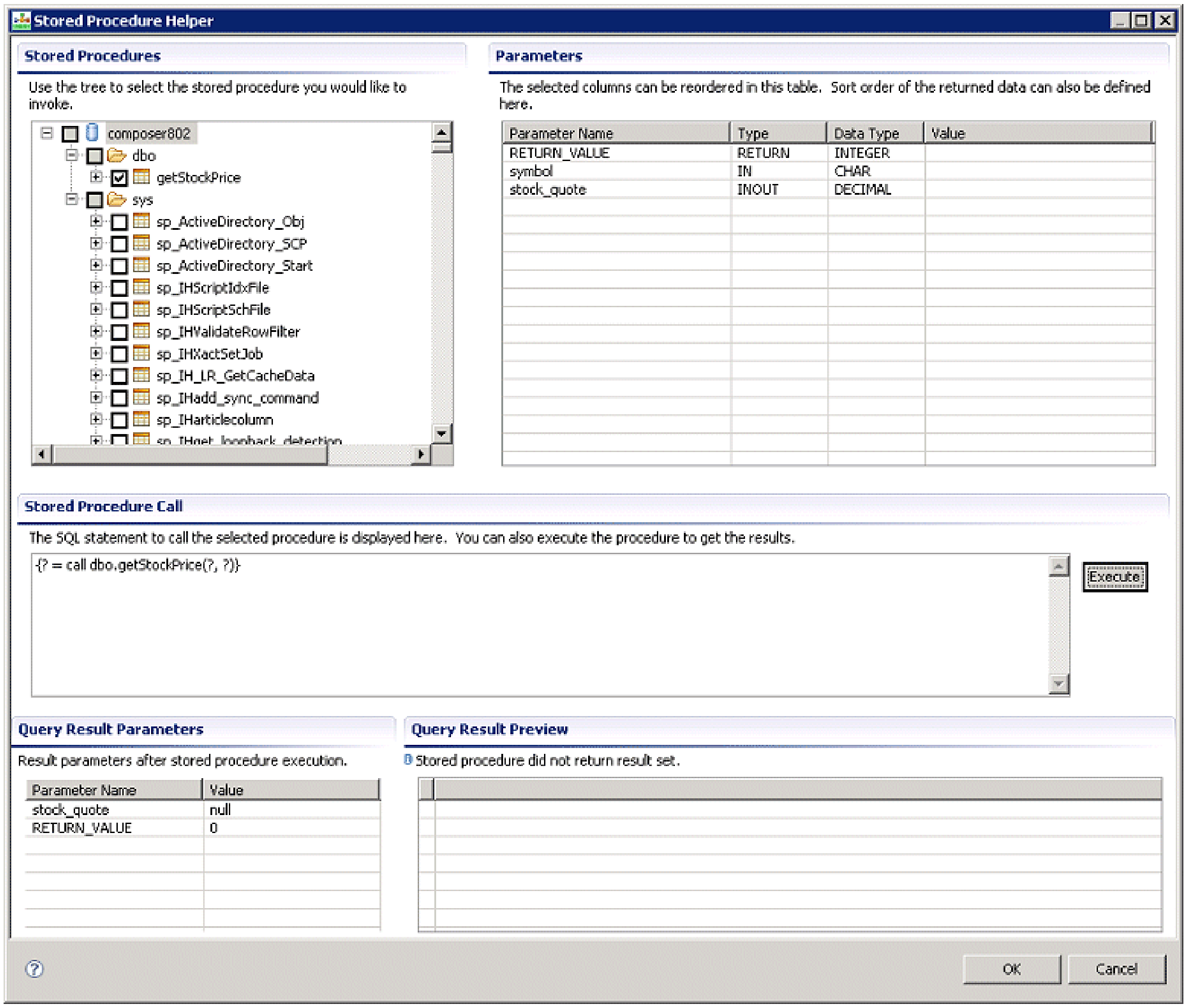
You can use Application variables in custom query files as part of the SQL statement. To use a variable, include its name within curly braces without the AppState. prefix. For example, the following statement uses varname1 and varname2. Their values will be substituted at the time the DB Data block queries the database. `SELECT name_of_function({varname1}, {varname2}) from dual` Results of the query are stored in a variable as a two-dimensional JSON array. This data can then be accessed via a Looping block or via scripting in the Assign or ECMAScript block. For example, if the database result set looks like this in tabular form:

Vegetables	Animals
lettuce	chicken
broccoli	lion

The JSON for the result will look like this: `{"db_result":[["lettuce", "chicken"], ["broccoli", "lion"]],"db_result_columns":["vegetables", "animals"]}` **Note:** An example of custom queries is in the Database Stocks Template application.

Stored Procedure Helper

If you select **StoredProcedure** for the Query Type property in the DB Data Block, you can click the ... button on the property row to open the Stored Procedure Helper dialog box. Here you can select a stored procedure, execute it, and get query results. A completed example is shown below.



Use the tree to select the stored procedure you would like to invoke.

- composer802
  - dbo
    - getStockPrice
  - sys
    - sp_ActiveDirectory_Obj
    - sp_ActiveDirectory_SCP
    - sp_ActiveDirectory_Start
    - sp_IHScriptIdxFile
    - sp_IHScriptSchFile
    - sp_IHValidateRowFilter
    - sp_IHXactSetJob
    - sp_IH_LR_GetCacheData
    - sp_IHadd_sync_command
    - sp_IHarticlecolumn
    - sp_IHget_loopback_detection

The selected columns can be reordered in this table. Sort order of the returned data can also be defined here.

Parameter Name	Type	Data Type	Value
RETURN_VALUE	RETURN	INTEGER	
symbol	IN	CHAR	
stock_quote	INOUT	DECIMAL	

The SQL statement to call the selected procedure is displayed here. You can also execute the procedure to get the results.

```
{? = call dbo.getStockPrice(?, ?)}
```

Result parameters after stored procedure execution.

Parameter Name	Value
stock_quote	null
RETURN_VALUE	0

Stored procedure did not return result set.

Setting up a Stored Procedure Call

The Stored Procedure Helper opens when Composer is successfully able to connect to the database specified in your connection profile. Any stored procedures in the database accessible from the specified user account are shown in hierarchical format in the Database Structure pane of the Stored Procedure Helper. To set up a stored procedure call:

1. Specify which stored procedure should be executed.
2. Select the stored procedure to execute by checking appropriate item in the Database Structure pane.
3. Parameters and Return Value appear in the Parameters pane. Specify the value (application variable) for each of the parameter into which the output value is stored after the stored procedure has executed.

4. To test the stored procedure, click the **Execute** button. This executes the stored procedure in the appropriate database. If the stored procedure returns any records, they are displayed in the Query Results Preview pane. Any output values are displayed in the Query Result Parameters pane. A message shows the number of records returned as a result of the query.
5. Click **OK** to save your query and update the DB Data block with the new query. If you click Cancel, all changes are discarded and no changes are made to the DB Data block.

Note: Composer does not support the REF CURSOR return type in a stored procedure.

Password Encryption

Composer can now encrypt the database connection profile passwords so that they are not written in the clear to the connection.properties file.

Encryption Key

In order to enable encryption, you must first create an encryption key. Composer requires a 128-bit (16 bytes) key, in hex-encoded format. This can be randomly generated by the OpenSSL tool, using the following command line:

```
$ openssl rand -hex 16 75b8ec9a3ce60a21c4f94236a1b55fb2
```

Any random source will do. Another example is `http://www.random.org/cgi-bin/randbyte?nbytes=16&format=h` (With this example, you will have to remove the spaces in the output.)

Save the encryption key to a text file. Note that this file should be securely stored, so that it can only be read by the Composer process and the backend Tomcat/IIS processes.

Configuring Composer Preferences

In the **Composer** > **Security** preference page, set the Encryption Key Location preference to point to the encryption key file created in the previous step.

Encrypting the Database Connection Profile Password

In the Connection Profile Editor, next to the Password field, enable the **Encrypt** checkbox. Now, when you save the Connection Profile, the password will be scrambled in the connection.properties file.

Enabling Decryption in the Backend

When the application runs, the application server will need to be able to decrypt the password so that it can connect to the database. For this, the application needs to be configured with the location of the encryption key file.

Java Composer Projects

If it doesn't already exist, create the file WEB-INF/composer.properties inside the project. Inside the file, enter the following line:

```
composerEncryptionKey=C:\\secrets\\encryption-key.txt
```

(Note that the backslashes here must be escaped.)

.NET Composer Projects

Edit the web.config file's appSettings entry:

```
<appSettings>

   <add key="composerEncryptionKey" value="C:\secrets\encryption-key.txt" />
 ...


</appSettings>
```

(Backslashes here are fine.)

Limitations and Workarounds

The Query Builder supports creating SELECT statements. The following is a list of limitations along with suggested workarounds:

- INSERT, UPDATE, and DELETE statements cannot be created using the Query Builder. Advanced SQL features, such as outer joins, subqueries, and unions are also not supported. A custom query can be used to overcome these limitations.
- if you rename a DB Data block, its corresponding SQL statement file in the db folder will not be updated and will not be valid until you generate code again.
- For details on SQL datatypes supported by Composer, see Supported SQL Datatypes.

Oracle Client Setup for IIS

To set up an Oracle client for Internet Information Services:

1. Install the Oracle client components on the application server.
2. Create a `tnsnames.ora` file in the `C:\oracle\ora81\network\ADMIN` folder where C:\oracle is the installation folder of Oracle client components.
3. Add the following lines to `tnsnames.ora` where COMPDB1 is any alias of choice, XYZ is the Oracle server, COMPOSER is the **Service Name** as configured on the Oracle listener (server). After doing this, you should be able to connect to Oracle using `sqlplus user/pwd@COMPDB1` as the command at the command prompt.

COMPDB1 =   (DESCRIPTION =   (ADDRESS_LIST =   (ADDRESS = (PROTOCOL = TCP)(HOST = XYZ.us.int.genesyslab.com)(PORT = 1521))   )   (CONNECT_DATA =   (SERVICE_NAME = COMPOSER)   )   )

4. Create a System DSN using the Data Sources (ODBC) under **Administrative Tools**.
5. Make sure that **Data Source Name** specified above is exactly same as the **Database Name** specified in the Composer database connection profile and **TNS Service Name** is the same as the alias in step 3.
6. Click on **Test Connection** in the database connection profile. The connection should be successful and the Composer VXML application should be able to connect to the database.

Steps 4, 5 and 6 can be avoided if the alias used in the `tnsnames.ora` file is same as the database name specified in Composer.

Supported SQL Datatypes

Composer's DB Data Block can access many common types of data stored in supported databases. The following tables summarize the level of support that Composer provides. The tables are organized by the Composer project type (Java or .NET), and by whether you're doing a standard SQL query or executing a stored procedure. The levels of support that Composer claims:

The levels of support that Composer claims:

?	Datatype is fully supported.
?*	Datatype is supported, but in the Composer UIs (Query Builder and Stored Procedure Helper), it may appear as "Unknown" or "Other." The queries themselves will work
?	Datatype is not currently supported.

Supported SQL Server Datatypes

Datatype	Java Project SQL query	Java Project Stored Procedure	.NET Project SQL query	.NET Project Stored Procedure
bigint	?	?	?	?
int	?	?	?	?
decimal	?	?	?	?
int	?	?	?	?
numeric	?	?	?	?
smallint	?	?	?	?
tinyint	?	?	?	?
float	?	?	?	?
real	?	?	?	?
date	?	?	?	?
datetime	?	?	?	?
datetimeoffset	?*	?*	?*	?*
char	?	?	?	?
text	?	?	?	?
varchar	?	?	?	?
nchar	?	?	?	?
ntext	?	?	?	?
nvarchar	?	?	?	?

Datatype	Java Project SQL query	Java Project Stored Procedure	.NET Project SQL query	.NET Project Stored Procedure
binary	?	?	?	?
sql_variant	?	?	?*	?
timestamp	?	?	?	?

Supported Oracle Datatypes

Datatype	Java Project SQL query	Java Project Stored Procedure	.NET Project SQL query	.NET Project Stored Procedure
number	?	?	?	?
binary_float	?*	?*	?*	?*
binary_double	?*	?	?*	?*
date	?	?	?	?
char	?	?	?	?
varchar	?	?	?	?
varchar2	?*	?*	?*	?*
nchar	?*	?	?*	?
nvarchar2	?*	?	?*	?*

Voice CTI Blocks

CTI (which stands for *Computer Telephony Integration*) blocks provide interfaces between Genesys Voice Platform (GVP) and Genesys Framework components and SIP Server. There are six CTI blocks:

- **Get Access Number Block** for using Get access number to retrieve the access code (number) of a remote site from an IVR Server.
- **Interaction Data Block** for sending attached data. Get and Put operations are supported.
- **Route Request Block** for sending route requests. It uses the Userdata extension attribute for sending back data attached to an interaction (attached data).
- **Statistics Block** to retrieve statistics from Stat Server via IServer.
- **ICM Interaction Data Block** to work with a Cisco product called Intelligent Contact Management (ICM), which provides intelligent routing and Computer Telephony Integration. You can use the GVP ICM Adapter in VoiceXML applications when invoking services, responding to requests, and sharing data.
- **ICM Route Request Block** to transfer a call to Intelligent Contact Management.

CTI Scenarios: SIPS versus CTIC

Composer will generate code for both SIP Server and CTI Connection scenarios simultaneously. The code to be executed at runtime depends on which scenario is active when the voice application runs. No decision is required at design time. For more information, see the topic CTI Scenarios. Also see the *VoiceXML Reference* on the Genesys Voice Platform Wiki.

CTI Scenarios

There are feature differences between the SIPS and CTIC scenarios. The following table gives a summary of the CTI blocks, and for each CTI block it lists the differences in behavior for the two CTI scenarios.

CTI Block Name	Supports CTIC Case?	Supports SIPS Case?	Comments
Interaction Data	Yes	Yes	Supported operations in each scenario: CTIC: • PUT • GET • DELETE • DELETEALL • REPLACE SIPS: • PUT • GET Types of interaction data supported: CTIC: • USERDATA SIPS: • USERDATA
Get access number	Yes	No	Get access number block can only be used in the CTIC scenario. Types of interaction data supported: CTIC: • USERDATA • EXTENSIONDATA
Statistics	Yes	No	Statistics block can only be used in the CTIC scenario.
Route Request	Yes	Yes	Types of interaction data supported: CTIC:

			• USERDATA • EXTENSIONDATA SIPS: • USERDATA Types of transfers supported: CTIC: • Blind • Bridge SIPS: • Consultation • Blind • bridge

In case a CTI block or feature is used in a CTI scenario in which it is not supported, appropriate exceptions will be thrown at runtime indicating that the feature is not supported. The table below gives a list of all exceptions that can be thrown by CTI blocks and other possible CTI-related exceptions that can be thrown if errors are encountered at runtime.

Block(s)	Exception	Error Message	Description
Interaction Data Get access number Statistics	error.com.genesyslab.composer.invalidkey	Missing <block name> key <key name>	This is the event error for handling an invalid key name.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.operationtimedout	Operation timed out.	This exception will be thrown when a <receive> operation, executed in the context of a CTIC specific operation, times out.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.receiveerror	<Error string returned by CTIC>	If the <receive> fails and an error is reported by CTIC, this exception will be thrown.
Interaction Data	error.com.genesyslab.composer.unsupportedop	Delete operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata DELETE in the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupportedop	DeleteAll operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata DELETEALL in the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupportedop	Replace operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata REPLACE in the CTI using SIPS scenario.

Get access number	error.com.genesyslab.composer.unsupported	AccessNumGet operation not supported in case of CTI using SIPServer.	If the user wants to do a AccessNumGet in the CTI using SIPS scenario.
Statistics	error.com.genesyslab.composer.unsupported	Statistics block not supported in case of CTI using SIPServer.	If the user wants to do a PeekStatReq or GetStatReq in the CTI using SIPS scenario.
Route Request	error.com.genesyslab.composer.unsupported	Consultation transfer is not supported in case of CTI using CTIConnector.	If user sets Transfer type to consultation in case of CTI using SIPS.

Get Access Number

The Get access number block uses Get access number to retrieve the access code (number) of a remote site from an IVR Server. It can be used to get the agent number when the application transfers a call to an agent at a remote site (remote switch transfers).

Notes:

- This block can be used in CTIC scenario only. It will not work when CTI functionality is accessed using SIP Server.
- This block is not supported when GVP is configured in Network mode.

Get Access Number Block Exception Events

The Get access number block has four exception events as described in Event Descriptions Exception Event Descriptions:

error.com.genesyslab.composer.invalidkey error.com.genesyslab.composer.receiveerror error.com.genesyslab.composer.operationtimeout error.com.genesyslab.composer.unsupported (preselected into the Supported column as a default exception)

The Get access number block has the following properties:

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Property Common Properties. The exception error.com.genesyslab.composer. unsupported is preselected into the Supported column of the Exceptions dialog box as a default exception.

Variables Property

To declare session variables for the application or subcallflow:

1. Select the Variables row in the block's property table.
2. Click the ... button to open the Variable Settings dialog box.

These variables apply only to the Property Entry block, unless otherwise indicated.

Note: Request URi parameters created in IVR Profiles during the VoiceXML application provisioning are passed to the Composer generated VoiceXML application as request-uri parameters in the `session.connection.protocol.sip.requesturi` session array. An Entry block variable can use these parameters by setting the following expressions to the variable values: `typeof session.connection.protocol.sip.requesturi['var1'] == 'undefined' ? "LocalDefaultValue" : session.connection.protocol.sip.requesturi['var1']`. If parameters are set as part of IVR Profiles provisioning in the Genesys VoiceXML provisioning system, and if these parameters have the same names as variables set in the Entry block's **Variables** property with the above mentioned `sip.requesturi` expression, then the `SIP-Request-URI` parameters will take precedence over the user variable values set in the Entry block.

Many blocks enable the use of variables rather than static data. For example, the Prompt block can play the value of a variable as Text-to-Speech. Variables whose values are to be used in other blocks must be declared here so that they appear in the list of available variables in other blocks.

The value collected by an Input block or a Menu block is saved as a session variable whose name is the same as the block name.

Destination Dn Property

To enter a Destination Dn:

1. Select the Destination Dn row in the block's property table.
2. In the Value field, type a Destination Dn.

Remote Switch Location Property

To enter a remote switch location:

1. Select the Remote Switch Location row in the block's property table.
2. In the Value field, type a value specifying the remote switch location.

Remote switch transfers use the AccessNumGet message, which is sent by the IVR to the IVR Server to request that the call be routed to a remote site. For information on AccessNumGet and the Location parameter, refer to the IVR SDK XML Developer’s Guide, which is available on the Genesys Technical Support website or on the Developer Documentation Library DVD. Refer to the Location parameter. The value of the Location parameter will be the name of the switch defined in the Configuration Database.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined Property variable for further processing.

Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Interaction Data Block

Use the Interaction Data block for sending attached data. Get and Put operations are supported. Background: Attached data can be attached to calls by different T-Server clients. For example, an IVR might attach data to a call by collecting the numbers that callers press on their telephone keypads in response to a prompt. An agent might also attach data to a call using a desktop application. Once T-Server attaches the data, it becomes interaction data, which can be used in expressions and for reporting. T-Server stores attached data in AttributeUserData of event messages.

- Get values are extracted from the User Data received at the start of the call as part of the INVITE to the GVP.
- For Put , the NGI extension <send> tag will be used to send data immediately to the SIP Server. The data will be sent in the SIP INFO Body.

This block supports working with both SIPServer and CTIConnector CTI scenarios. There are feature differences as listed in CTI scenarios. Also see the standard VoiceXML session variables documented in the GVP 8.1 VoiceXML 2.1 Reference Help (Help > Contents). The Interaction Data block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The Interaction Data block has the following Exception Events:

- error.com.genesyslab.composer.receiveerror
- error.com.genesyslab.composer.operationtimeout
- error.com.genesyslab.composer.unsupported (pre-selected as a default exception)
- error.com.genesyslab.composer.invalidkey

Operation Property

This property indicates the type of operation to perform:

- get--to fetch the user data (CTIC and SIPS)
- put--to send updated user data (CTIC and SIPS)
- delete--to delete selected user data (CTIC only)
- deleteall--to delete all user data (CTIC only)
- replace--to replace existing user data with alternate user data (CTIC only)

To select a value for the Operation property:

1. Select the Operation row in the block's property table.
2. In the Value field, select get, put, delete, deleteall, or replace from the drop-down list.

Note: delete, deleteall, and replace are not supported for CTI using SIP Server.

Values Property

The Values property holds the list of variables to be fetched or sent. The name of the variable must match the UserData key name. Note: All key names for attached data passed from an IRD Strategy must be in all lower case. To select values:

1. Click the Values row in the block's property table.
2. Click the ▭ button to open the Values dialog box.
3. Select individual global variables, or click Select all or Deselect all.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

1. Click OK.

Route Request Block

Use the Route Request block for sending route requests. It uses the Userdata extension attribute for sending back data attached to an interaction (attached data). Attached data can be attached to calls by different T-Server clients. For example, an IVR might attach data to a call by collecting the numbers that callers press on their telephone keypads in response to a prompt. An agent might also attach data to a call using a desktop application. Once T-Server attaches the data, it becomes interaction data, which can be used in expressions and for reporting. T-Server stores attached data in AttributeUserData of event messages. You can select any application variables to pass as interaction data. The name of the variable will be used as the Key of the interaction data. The Destination number represents the target to which the call will be routed. It can be one of following:

- Virtual Route Point Destination Number
- Direct extension of an Agent
- External Number to dial out

This block supports working with both SIPServer and CTIConnector CTI scenarios. There are feature differences as listed in CTI scenarios. The Route Request block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The Route Request block supports the following Exception Event Descriptions:

- connection.disconect.hangup
- connection.disconnect.transfer
- error
- error.com.genesyslab.composer.unsupported
- error.connection.baddestination (supported by default)
- error.connection.noauthorization

- error.connection.noresource
- error.connection.noroute
- error.connection
- error.unsupported.transfer.blind
- error.unsupported.transfer.consultation
- error.unsupported.uri

Language Property

The language set by this property overrides any language set by the Set Language block, the Project preferences, or the incoming call parameters. The property takes effect only for the duration of this block, and the language setting reverts back to its previous state after the block is done. In the case of the Route Request block, this property affects the language of grammars used for ASR input:

1. Click under Value to display a down arrow.
2. Click the down arrow and select English - United States (en-US) or the variable that contains the language.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Interaction Data Property

To select session variables:

1. Click the Interaction Data row in the block's property table.
2. Click the ⋯ button to open the Interaction Data dialog box.
3. Select individual global variables, or click Select all or Deselect all.
4. Click OK.

Transfer Audio Property

The optional Transfer Audio property plays a prompt to the end user while the number is being dialed out. You provide the URI of the audio source to play while the transfer attempt is in progress (before the other end answers). If the callee answers, the interpreter terminates playback of the recorded audio immediately. If the end of the audio file is reached and the callee has not yet answered, the interpreter plays the audio tones from the far end of the call (ringing, busy). If the resource cannot be fetched, the error is ignored and the transfer continues. To provide a Transfer Audio value:

1. Select the Transfer Audio row in the block's property table.
2. In the Value field, type a Transfer Audio value URI (HTTP or RTSP) specifying the location of the audio file to play.

Aai Property

Use the optional Application-to-Application Information (the Aai property) for the data that is to be transferred from the current application to another application. Use this option to transfer the call to a number that initiates another voice application. To assign a value to the Aai property:

1. Select the Aai row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Connect Timeout Property

Use the Connect Timeout property for the connection timeout value. The default is 30 seconds. To provide a timeout value:

1. Select the Connect Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Destination Property

The Destination property contains the destination phone number. The destination number can be one of the following:

- A Virtual Route point number on which the IRD Strategy is loaded
- Extension number of an Agent
- External number

The value must be specified in one of the formats below:

- sip:[user@]host[:port]
- tel:phonenumber e.g., tel:+358-555-1234567

For information on this property, select Help > Contents and see the VoiceXML Reference Help. Specifically see Standard VoiceXML > Variables > Transfer, attribute dest.   To assign a value to the Destination property:

1. Select the Destination row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described in the Entry block.

Max Call Duration Property

Use the Max Call Duration property for the maximum call duration. The default is 3600 seconds. (This is not supported for Consultation Transfer Type.) Note: If this is set to 0 (zero), an infinite value is supplied, and there is no upper limit to the call duration. To provide a value for the maximum call duration:

1. Select the Max Call Duration row in the block's property table.
2. In the Value field, type a value for the maximum call duration.

Transfer Type Property

The Transfer Type property specifies the type of transfer required. To assign a value to the Transfer Type property:

1. Select the Transfer Type row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

Note: The selected transfer type will work only if the platform is provisioned to support that type of transfer.

Blind

This is the default setting. The platform redirects the caller to the agent without remaining in the connection, and it does not monitor the outcome. The platform generates a connection.disconnect.transfer event immediately, regardless of the transfer outcome.

Bridge

The platform adds the agent to the connection, and it remains in the connection for the duration of the transferred call. Any included grammars control the listening during the transfer. Control of the call always returns to the application when the transfer ends, regardless of the transfer result. If the caller or network disconnects the call, the platform generates connection.disconnect.hangup event. If the agent disconnects the call, the transfer outcome is set to far_end_disconnect. Note: Use this option if the application needs to continue in self-service after the agent and caller communication is over; for example, to present a survey to the end user.

Method Property

The Method property specifies the type of route request required. To assign a value to the Method property:

1. Select the Method row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

Bridge

A Bridge method indicates that the Media Control Platform (MCP) bridges the media path.

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The route request fails if a non-2xx final response is received for the INVITE request.

This is a two-leg route request (in other words, it occupies two channels on the platform). The platform stays in the signaling path and is responsible for bridging the two call legs.

Hkf (Hookflash)

A Hookflash method indicates a route request using DTMF digits (RFC 2833).

1. The Media Control Platform (MCP) sends DTMF digits on the media channel. The platform leaves it to the media gateway or switch to perform the route request on the network.
2. Configurable options enable you to specify whether the call will be disconnected by the platform or by the remote end. Otherwise, the call is disconnected after a configured timeout.

This is a one-leg route request (in other words, it occupies only one channel on the platform).

Refer

A Refer method indicates that the route request is based on a SIP REFER message (RFC 3515).

1. The platform sends a REFER request to the caller, with the callee (as specified in the VoiceXML application) in the Refer-To: header.
2. The route request fails if a non-2xx final response is received for the REFER.

This is a one-leg route request (in other words, it occupies only one channel on the platform).

Referjoin

A Referjoin method indicates a consultative REFER route request (RFC 3891).

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The platform also sends a REFER request to the caller, with the callee’s information in the Replaces header.
3. The platform considers the route request to be successful if it receives a BYE from the caller after a 2xx response for the REFER.
4. The route request fails if a non-2xx final response is received for the INVITE request or for the REFER request.

This is a two-leg, or join-style, route request (in other words, it occupies two channels on the platform).

Mediaredirect

A Mediaredirect method indicates a media redirection route request. The Media Control Platform (MCP) uses SIP to handle call control between the caller and the callee, and the RTP media channel is connected directly between the caller and callee.

1. The platform sends an INVITE request to the callee without SDP.
2. If the route request is proceeding, the callee responds with a 200 OK that includes an SDP offer.
3. The platform forwards the SDP offer in a re-INVITE request to the caller.
4. The caller responds with a 200 OK that includes the SDP answer.
5. The platform forwards the SDP answer to the callee in an ACK response.
6. The route request fails if a non-2xx final response is received for the initial INVITE request.

This is a two-leg route request (in other words, it occupies two channels on the platform).

Get Shadow Variables Property

Shadow variables provide a way to retrieve further information regarding the value of an input item.

By setting this property to true, it will expose the block’s shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block’s Logging Details will show RouteRequest1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: RouteRequest1$.duration. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Transfer Results Property

To select transfer results:

1. Click the Transfer Results row in the block's property table.
2. Click the ⋯ button to open the Transfer Results dialog box.
3. Select items from the list of available CPA results, or click Select all or Deselect all as needed, then click OK.

For each item selected, an outport node is added to allow specific actions to be taken for that condition.

Input Grammar Dtmf Property

Use the Input Grammar Dtmf property to specify the DTMF Grammar for the Input Block. The DTMF Grammar is processed and handled by GVP. In the case of external grammars, this specifies the actual path of the grammar file / resource for DTMF Grammars. This is only valid when the Grammar Type is externalGrammar and Input Mode is dtmf or hybrid. To assign a value to the Input Grammar Dtmf property:

1. Select the Input Grammar Dtmf row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Input Grammar Voice Property

Use the Input Grammar Voice property to specify the Voice Grammar for the Input block. If you are writing hybrid applications that allow both DTMF and Speech input, specify both the DTMF and Voice grammars. The Voice Grammar is sent to the ASR Engine for processing, whereas the DTMF grammar is processed by GVP. As a result, you need two separate grammars for Voice and DTMF in the case of hybrid applications that allow both Voice and DTMF inputs. In the case of external grammars, this specifies the actual path of the grammar file / resource for ASR Grammars.. This is only valid when

Grammar Type is externalGrammar and Input Mode is voice or hybrid. To assign a value to the Input Grammar Voice property:

1. Select the Input Grammar Voice row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Input Mode Property

To assign a value to the Input Mode property:

1. Select the Input Mode row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

DTMF

The DTMF format indicates the menu option mode of input will be via the telephone keypad.

Voice

The Voice format indicates the menu option mode of input will be a voice phrase. The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase.

Statistic Block

Use the Statistics block to retrieve statistics from Stat Server via IServer. The Statistics block enables you to receive data on statistics such as CurrNumberWaitingCalls and ExpectedWaitTime. Additionally, you can get a full report on the requested statistics for a specified object in the Configuration Layer. The object may be a queue, route point, or group of queues.

This block supports the following actions (operations):

- GetStatReq
- PeakStatReq

The Statistics block also uses the <send> tag.

Note: This block can be used in CTIC scenario only. It will not work when CTI functionality is accessed using SIPServer.

The Statistics block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The Statistics block has four page exceptions:

- error.com.genesyslab.composer.invalidkey
- error.com.genesyslab.composer.receiveerror
- error.com.genesyslab.composer.operationtimeout
- error.com.genesyslab.composer.unsupported (pre-selected by default)

Operation Property

The Operation property indicates the type of operation to perform:

- get—to execute a GetStatReq to return the current value of the requested statistics for the specified object (queue, routepoint, or group of queues)
- peek—to execute a PeekStatReq to return the value of CurrNumberWaitingCalls or ExpectedWaitTime. It cannot return any other value.

To select a value for the Operation property:

1. Select the Operation row in the block's property table.
2. In the Value field, select get or peek from the drop-down list.

The following properties apply and must be set if you choose get:

- Object Id
- Object Type
- Server Name
- Statistic Type

The following property applies and must be set if you choose peek:

- Peek Return Value

Note: Statistics can be requested at any time during the call. They must be preconfigured in Genesys Administrator before they can be used. For more information on configuring statistics, see the Framework Stat Server User's Guide.

Object Id Property

The Object Id property is used for a GetStatReq (get) operation.

This property works with the Object Type property.

- For RoutePoint, the value is the Alias of the corresponding DN in the Configuration Database.
- For Queue and GroupQueues, the value is the name of the corresponding object in the Configuration Database.

To provide a value for the Object Id:

1. Select the Object Id row in the block's property table.
2. In the Value field, type a value for the Object Id.

Object Type Property

The Object Type property is used for a GetStatReq (get) operation. As described in the Stat Server Object Types chapter in the Framework Stat Server User's Guide, valid Object types are:

- Queue
- RoutePoint
- GroupQueues

To provide a value for the Object Type:

1. Select the Object Type row in the block's property table.
2. In the Value field, type a value for the Object Type.

Server Name Property

The Server Name property is used for a GetStatReq (get) operation. This can be the IP address/ hostname or the fully qualified domain name of the Stat Server.

To provide a value for the Server Name:

1. Select the Server Name row in the block's property table.
2. In the Value field, type a value for the Server Name.

Statistic Type Property

The Statistic Type property is used for a GetStatReq (get) operation. Refer to the Framework Stat Server User's Guide for details on what the values of these objects can be.

To provide a value for the Statistic Type:

1. Select the Statistic Type row in the block's property table.
2. In the Value field, type a value for the Statistic Type.

Peek Return Value Property

The Peek Return Value property is used for a PeekStatReq (peek) operation. This specifies the application variable to hold the result–the current number of calls in the queue.

To select a value for the Peek Return Value property:

1. Select the Peek Return Value row in the block's property table.
2. In the Value field, select CurrNumberWaitingCalls or ExpectedWaitTime from the drop-down list.

Configuring GetStatReq/PeakStatReq Requests

To get GetStatReq/PeakStatReq requests to work

Configure I-Server as follows:

1. In the I-Server Options tab, create the following section: Stat:ExpectedWaitTime
2. Under that section, create the following options/values:
    - obj_id = dn@switch (DN is the DNIS/Routing Point being called. The switch used is that to which SIP Server is associated in case of behind the switch and the Virtual switch in case of in front of the switch. Example: 9020@CTI_Switch
    - obj_type = SObjectQueue
    - server_name = stat_server_name (The name of the Stat Server object in the Configuration Database).
    - stat_type = ExpectedWaitTime
    - update_frequency = 5

Configure Stat Server as follows:

1. In the Stat Server options tab, create the following section: ExpectedWaitTime
2. Under that section, create the following options/values:
    - Category = ExpectedWaitTime
    - MainMask = CallWait
    - Objects = Queue
    - Subject = DNAction
3. Connect applications as follows:
    - T-Server IVR – Message Server
    - Ixn-Server – T-Server_IVR, Stat Server
    - URS – T-Server_IVR, Stat Server, Message Server
    - Stat Server – T-Server_IVR, Message Server

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing.

Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

For more information, see Upgrading Projects/Diagrams.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks .

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

ICM Interaction Data Block

ICM refers to a Cisco product called Intelligent Contact Management, which provides intelligent routing and Computer Telephony Integration. You can use the GVP ICM Adapter in VoiceXML applications when invoking services, responding to requests, and sharing data. Use this block to send interaction data to ICM. It functions the same way as the existing Interaction Data block. Composer uses the VXML <gvp:send> tag to implement the ICM Interaction Data functionality.

ICM Variables

Voice Projects have a Project-level flag (Enable ICM) which controls whether ICM variables are available for selection and assignment to variables within Composer's Entry block. The Exit block’s Return Values property dialog allows you to select the ICM variables to be returned. You can also set the Enable ICM flag by right-clicking the Project in the Project Explorer, selecting **Properties**, and **ICM Support**. The types of variables supported by ICM are:

- CED--This is a single variable with the name ICM_CED. It is automatically added to the variables list in the Entry block.
- Call variables--There are 10 CallVars, with names ICM_CallVar1 through ICM_CallVar10. They are automatically added to the variables list in the Entry block.
- ECC variables--These are user-named variables, which are identified by having a prefix of ICM_ECC_user; for example, ICM_ECC_userMyVariable. In the Application Variables dialog, you can enter the names of the variables with or without the prefix. Composer provides a mechanism to automatically add the prefix.

Note: In all cases, the Enable ICM flag must be set for ICM variables to be selectable in the Entry block. The ICM Interaction Data block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The ICM Interaction Data block has the following exception events:

- error.com.genesyslab.composer.receiveerror
- error.com.genesyslab.composer.operationtimeout
- error.com.genesyslab.composer.unsupported (pre-selected as a default exception)
- error.com.genesyslab.composer.invalidkey

Values Property

The Values property holds the list of variables to be fetched or sent. The name of the variable must match the UserData key name. To select values:

1. Click the Values row in the block's property table.
2. Click the button to open the Values dialog box.
3. Select individual global variables, or click Select all or Deselect all.
4. Click OK.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

ICM Route Request Block

ICM refers to a Cisco product called Intelligent Contact Management, which provides intelligent routing and Computer Telephony Integration (CTI). You can use the GVP ICM Adapter in VoiceXML applications when invoking services, responding to requests, and sharing data.   Use the ICM Route Request block to transfer a call to ICM. Note:This block functions in the same way as the existing Route Request block. Composer uses the VXML <transfer> tag to implement the ICM Route Request functionality. The ICM Route Request block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Exceptions Property

Find this property's details under Common Properties. The following events are supported:

- connection.disconnect.hangup
- connection.disconnect.transfer
- error
- error.connection.noauthorization
- error.connection.baddestination
- error.connection.noresource
- error.connection.noroute
- error.connection
- error.unsupported.transfer.blind
- error.unsupported.transfer.consultation
- error.unsupported.uri
- error.com.genesyslab.composer.unsupported

Custom events are also supported.

Interaction Data Property

To select session variables:

1. Click the Interaction Data row in the block's property table.
2. Click the ⋯ button to open the Interaction Data dialog box.
3. Select individual global variables, or click Select all or Deselect all.
4. Click OK.

Output Result Property

You must use the Output Result property to assign the collected data to a user-defined variable for further processing. Note! This property is mandatory. You must select a variable for the output result even if you do not plan on using the variable. If this is not done, a validation error will be generated in the Problems view.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

Aai Property

Use the optional Application-to-Application Information (the Aai property) for the data that is to be transferred from the current application to another application. Use this option to transfer the call to a number that initiates another voice application. To assign a value to the Aai property:

1. Select the Aai row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Transfer Audio Property

The optional Transfer Audio property plays a prompt to the end user while the number is being dialed out. You provide the URI of the audio source to play while the transfer attempt is in progress (before the other end answers). If the callee answers, the interpreter terminates playback of the recorded audio immediately. If the end of the audio file is reached and the callee has not yet answered, the interpreter plays the audio tones from the far end of the call (ringing, busy). If the resource cannot be fetched, the error is ignored and the transfer continues. To provide a Transfer Audio value:

1. Select the Transfer Audio row in the block's property table.

2. In the Value field, type a Transfer Audio value URI (HTTP or RTSP) specifying the location of the audio file to play.

Connect Timeout Property

Use the Connect Timeout property for the connection timeout value. The default is 30 seconds. To provide a timeout value:

1. Select the Connect Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

Connect When Property

This property controls whether the connection is made after the call is picked up, or immediately. Select one of the following:

- Immediate
- Answered

Destination Property

The Destination property contains the destination phone number. The destination number can be one of the following:

- A Virtual Route point number on which the IRD Strategy is loaded
- Extension number of an Agent
- External number

The value must be specified in one of the formats below:

- sip:[user@]host[:port]
- tel:phonenumber e.g., tel:+358-555-1234567

For information on this property, select Help > Contents and see the GVP 8.1Voice XML 2.1 Reference Help. Specifically see Standard VoiceXML > Variables > Transfer, attribute dest. To assign a value to the Destination property:

1. Select the Destination row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described in the Entry block.

Max Call Duration Property

Use the Max Call Duration property for the maximum call duration. Default value is 0. This property is not supported for Consultation and Blind transfer types. Note: If this is set to 0 (zero), an infinite value is supplied, and there is no upper limit to the call duration. To provide a value for the maximum call duration:

1. Select the Max Call Duration row in the block's property table.
2. In the Value field, type a value for the maximum call duration.

Transfer Type Property

The Transfer Type property specifies the type of transfer required. To assign a value to the Transfer Type property:

1. Select the Transfer Type row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

Blind

This is the default setting. The platform redirects the caller to the agent without remaining in the connection, and it does not monitor the outcome. Once the caller is handed off to the network, the caller's session with the VoiceXML application cannot be resumed. The VoiceXML interpreter throws a connection.disconnect.transfer immediately, regardless of whether the transfer was successful or not.

Bridge

The platform adds the agent to the connection. Document interpretation suspends until the transferred call terminates. The platform remains in the connection for the duration of the transferred call; listening during transfer is controlled by any included <grammar>s. If the caller disconnects by going onhook or if the network disconnects the caller, the platform throws a connection.disconnect.hangup event. If the agent disconnects, then transfer outcome is set to near_end_disconnect and the original caller resumes her session with the VoiceXML application.

Consultation

The consultation transfer is similar to a blind transfer except that the outcome of the transfer call setup is known and the caller is not dropped as a result of an unsuccessful transfer attempt. When performing a consultation transfer, the platform monitors the progress of the transfer until the connection is established between caller and agent. If the connection cannot be established (e.g. no answer, line busy, etc.), the session remains active and returns control to the application. As in the case of a blind transfer, if the connection is established, the interpreter disconnects from the session, connection.disconnect.transfer is thrown, and document interpretation continues normally. Any connection between the caller and the agent remains in place regardless of document execution. Note: The selected transfer type will work only if the platform is provisioned to support that type of transfer.

Method Property

The Method property specifies the type of route request required. To assign a value to the Method property:

1. Select the Method row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

Bridge

A Bridge method indicates that the Media Control Platform (MCP) bridges the media path.

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The transfer fails if a non-2xx final response is received for the INVITE request.

This is a two-leg transfer (in other words, it occupies two channels on the platform). The platform stays in the signaling path and is responsible for bridging the two call legs.

Hkf (Hookflash)

A Hookflash method indicates a transfer using DTMF digits (RFC 2833).

1. The Media Control Platform (MCP) sends DTMF digits on the media channel. The platform leaves it to the media gateway or switch to perform the transfer on the network.
2. Configurable options enable you to specify whether the call will be disconnected by the platform or by the remote end. Otherwise, the call is disconnected after a configured timeout.

This is a one-leg transfer (in other words, it occupies only one channel on the platform).

Refer

A Refer method indicates that the transfer is based on a SIP REFER message (RFC 3515).

1. The platform sends a REFER request to the caller, with the callee (as specified in the VoiceXML application) in the Refer-To: header.
2. The transfer fails if a non-2xx final response is received for the REFER.

This is a one-leg transfer (in other words, it occupies only one channel on the platform).

Referjoin

A Referjoin method indicates a consultative REFER transfer (RFC 3891).

1. The platform sends an INVITE request to the callee, and a dialog is established between the callee and the platform.
2. The platform also sends a REFER request to the caller, with the callee’s information in the Replaces

header.

3. The platform considers the transfer to be successful if it receives a BYE from the caller after a 2xx response for the REFER.
4. The transfer fails if a non-2xx final response is received for the INVITE request or for the REFER request.

This is a two-leg, or join-style, transfer (in other words, it occupies two channels on the platform).

Mediaredirect

A Mediaredirect method indicates a media redirection transfer. The Media Control Platform (MCP) uses SIP to handle call control between the caller and the callee, and the RTP media channel is connected directly between the caller and callee.

1. The platform sends an INVITE request to the callee without SDP.
2. If the transfer is proceeding, the callee responds with a 200 OK that includes an SDP offer.
3. The platform forwards the SDP offer in a re-INVITE request to the caller.
4. The caller responds with a 200 OK that includes the SDP answer.
5. The platform forwards the SDP answer to the callee in an ACK response.
6. The transfer fails if a non-2xx final response is received for the initial INVITE request.

This is a two-leg transfer (in other words, it occupies two channels on the platform). attcourtesy attconsult attconference attoobcourtesy attoobconsult attoobconference For information on these methods, consult the section on how the Media Control Platform works in the *Genesys Voice Platform 8.1 Deployment Guide.*

Do CPA Analysis Property

Triggers whether the platform will detect who or what answered the call. Select one of the following:

- True
- False (default, no detection)

Get Shadow Variables Property

Shadow variables provide a way to retrieve further information regarding the value of an input item. By setting this property to true, it will expose the block's shadow variable within the callflow. When enabled, the shadow variable will be included in the list of available variables. (For example, the Log block's Logging Details will show RouteRequest1$.) A shadow variable is referenced as blockname$.shadowVariable, where blockname is the value of the input item's name attribute, and shadowVariable is the name of a specific shadow variable, for example: RouteRequest1$.duration. To assign a value to the Get Shadow Variables property:

1. Select the Get Shadow Variables row in the block's property table.

2. In the Value field, select true or false from the drop-down list.

Transfer Result Property

To select transfer results:

1. Click the Transfer Results row in the block's property table.
2. Click the button to open the Transfer Results dialog box.
3. Select items from the list of available CPA results, or click Select all or Deselect all as needed, then click OK.

For each item selected, an outport node is added to allow specific actions to be taken for that condition.

Input Grammar Dtmf Property

Use the Input Grammar Dtmf property to specify the DTMF Grammar for the Input Block. The DTMF Grammar is processed and handled by GVP. In the case of external grammars, this specifies the actual path of the grammar file / resource for DTMF Grammars. This is only valid when the Grammar Type is externalGrammar and Input Mode is dtmf or hybrid. To assign a value to the Input Grammar Dtmf property:

1. Select the Input Grammar Dtmf row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Input Grammar Voice Property

Use the Input Grammar Voice property to specify the Voice Grammar for the Input block. If you are writing hybrid applications that allow both DTMF and Speech input, specify both the DTMF and Voice grammars. The Voice Grammar is sent to the ASR Engine for processing, whereas the DTMF grammar is processed by GVP. As a result, you need two separate grammars for Voice and DTMF in the case of hybrid applications that allow both Voice and DTMF inputs. In the case of external grammars, this specifies the actual path of the grammar file / resource for ASR Grammars.. This is only valid when Grammar Type is externalGrammar and Input Mode is voice or hybrid. To assign a value to the Input Grammar Voice property:

1. Select the Input Grammar Voice row in the block's property table.
2. In the Value field, select a value from the drop-down list.

Values are the Voice Application Variables described under the Variables Property.

Input Mode Property

To assign a value to the Input Mode property:

1. Select the Input Mode row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

DTMF

The DTMF format indicates the menu option mode of input will be via the telephone keypad.

Voice

The Voice format indicates the menu option mode of input will be a voice phrase.

Hybrid

The Hybrid menu mode will handle both DTMF and Voice inputs, that is via telephone keypad and voice phrase.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Working with CTI Applications

Composer provides CTI blocks for two CTI scenarios supported by GVP:

- SIP Server (SIPS) scenario, which uses the Genesys SIP Server component to gain access to CTI functionality.
- CTI Connector (CTIC) scenario, which uses GVP's CTI Connector component to access CTI functionality provided by Genesys Framework.

These two scenarios do not provide identical capabilities and key differences are highlighted later in these topics. Composer provides four CTI blocks for accessing CTI functions. It generates VXML for each of these blocks that can work in either CTI scenario (SIPS or CTIC), and does not ask the user to choose between the SIPS or CTIC scenarios at design time. The decision to use CTIC or SIPS is made at runtime based on the X-Genesys headers received from GVP's Resource Manager. Therefore, the Composer user interface does not need to expose a Project-level preference for specifying the CTI scenario. **Note:** The CTI Connector provides different capabilities depending on the configuration in which other Genesys components like the IServer are deployed. For more details, please refer to the GVP documentation. Also see GVP Debugging Limitations.

Design Paradigms for CTI Applications

There are two design paradigms for building CTI applications with GVP in which Composer can be used:

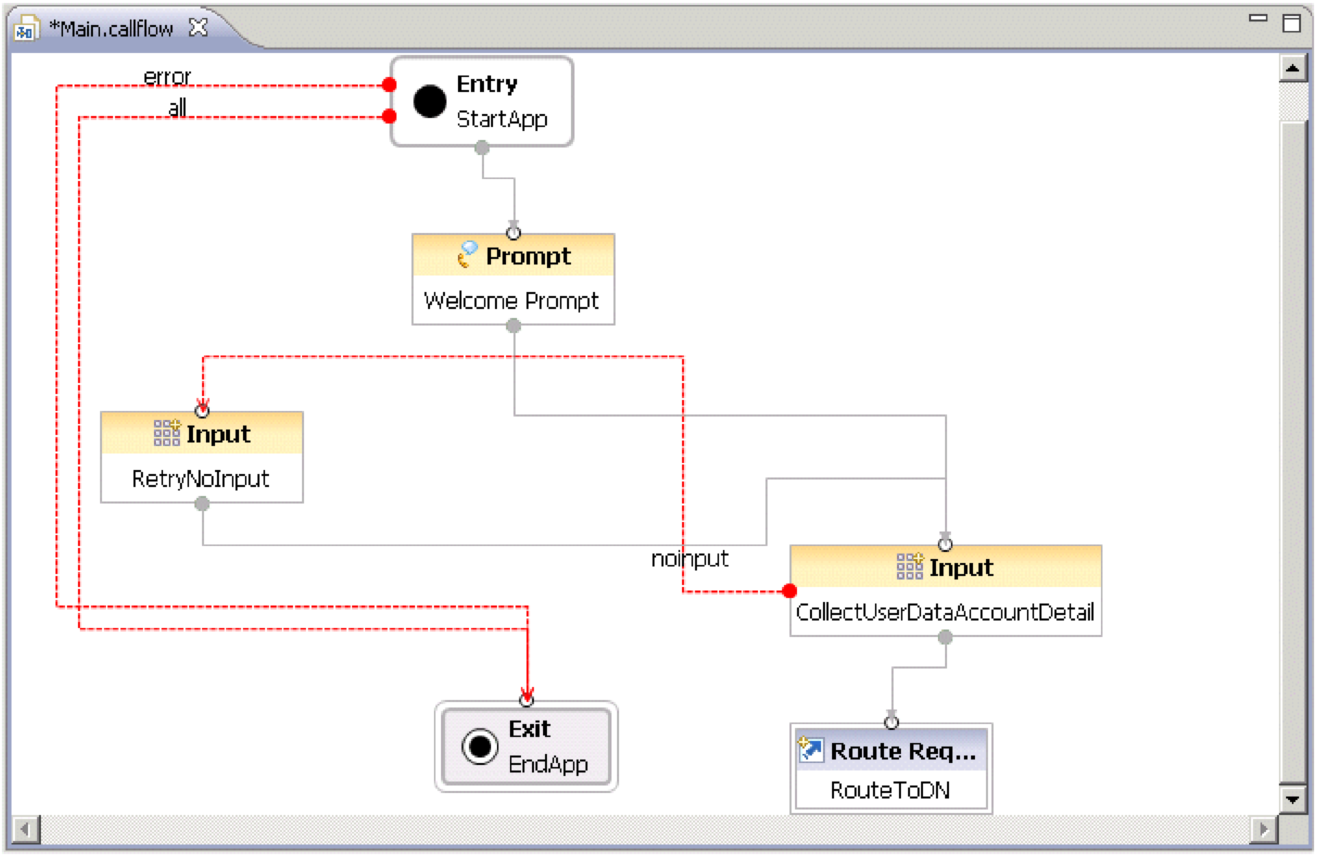
- Standard VXML Applications
- URS-Centric Applications

These paradigms differ in the extent to which the VXML application is involved in performing call control. **Standard VXML Applications** In this paradigm, the VXML application gets invoked first and can go through VXML interactions with the caller before using the <transfer> tag to transfer the call to another party such as queuing for an agent. At this point, the control of the call is passed to the SIP Server or CTI Connector while waiting for an agent. During this time, SIP Server or CTI Connector may invoke additional call treatments on GVP like playing music or invoking other applications. **URS-Centric Applications** In this paradigm, the VXML application is always invoked as a treatment by Genesys URS. The incoming call is controlled by Genesys URS and a strategy retains full control of the call. The strategy invokes specific treatments on GVP IVR as a media server to play prompts, play music, collect user input or execute a VXML application. In this paradigm, the VXML application does not use tags like <transfer> nor does any other kind of call control. Those decisions are left to the strategy. The VXML application returns user input collected during the call back to the strategy and lets the strategy make all call control decisions. Composer can be used to write VXML applications following either of the above paradigms.

Typical CTI Callflow

Before you start building a typical CTI application, the following information is required:

- The Genesys Virtual Route Point destination address. This is the address/location where the Genesys strategy is present (an integer number--for example, 5001).
- Strategy application on the Framework side (IRD) to find and transfers the call to an agent.



The following describes the interaction flow of this callflow:

1. GVP starts executing the generated VoiceXML application script.
2. The caller hears the Welcome prompt.
3. The caller is requested to enter the account details.
4. If the caller does not enter the required details within the maximum time frame provided, the caller is asked to retry.
5. The application issues a route request to the route DN configured in the Route Request block. (This occurs via the <transfer> tag, supported in both CTIC and SIP Server scenarios.)
6. The caller-entered data is sent as UserData to the routed DN, and the called strategy does the knowledge based transfer to the available agent based on the User Data .

7. This application ends after the Route Request has been issued.
8. The called strategy can play Voice treatments to the caller until the next available agent is available.
9. Finally, the caller will be transferred to the Agent.

Note: The Route Request block can be configured in various Transfer modes (Bridge / Consultation) to gain back the control of the callflow after the called strategy returns back the execution. Please check the Route Request topic block for more details.

CTI Scenarios

There are feature differences between the SIPS and CTIC scenarios. The following table gives a summary of the CTI blocks, and for each CTI block it lists the differences in behavior for the two CTI scenarios.

CTI Block Name	Supports CTIC Case?	Supports SIPS Case?	Comments
Interaction Data	Yes	Yes	Supported operations in each scenario: CTIC: • PUT • GET • DELETE • DELETEALL • REPLACE SIPS: • PUT • GET Types of interaction data supported: CTIC: • USERDATA SIPS: • USERDATA
Get access number	Yes	No	Get access number block can only be used in the CTIC scenario. Types of interaction data supported: CTIC: • USERDATA • EXTENSIONDATA

Statistics	Yes	No	Statistics block can only be used in the CTIC scenario.
Route Request	Yes	Yes	Types of interaction data supported: CTIC: • USERDATA • EXTENSIONDATA SIPS: • USERDATA Types of transfers supported: CTIC: • Blind • Bridge SIPS: • Consultation • Blind • bridge

In case a CTI block or feature is used in a CTI scenario in which it is not supported, appropriate exceptions will be thrown at runtime indicating that the feature is not supported. The table below gives a list of all exceptions that can be thrown by CTI blocks and other possible CTI-related exceptions that can be thrown if errors are encountered at runtime.

Block(s)	Exception	Error Message	Description
Interaction Data Get access number Statistics	error.com.genesyslab.composer.invalidkey	Missing <block name> key <key name>	This is the event error for handling an invalid key name.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.operationtimedout	Operation timed out.	This exception will be thrown when a <receive> operation, executed in the context of a CTIC specific operation, times out.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.receiveerror	<Error string returned by CTIC>	If the <receive> fails and an error is reported by CTIC, this exception will be thrown.
Interaction Data	error.com.genesyslab.composer.unsupported	Delete operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata DELETE in the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupported	DeleteAll operation not supported in case of CTI	If the user wants to do a userdata DELETEALL in

		using SIPServer.	the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupported	Replace operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata REPLACE in the CTI using SIPS scenario.
Get access number	error.com.genesyslab.composer.unsupported	AccessNumGet operation not supported in case of CTI using SIPServer.	If the user wants to do a AccessNumGet in the CTI using SIPS scenario.
Statistics	error.com.genesyslab.composer.unsupported	Statistics block not supported in case of CTI using SIPServer.	If the user wants to do a PeekStatReq or GetStatReq in the CTI using SIPS scenario.
Route Request	error.com.genesyslab.composer.unsupported	Consultation transfer is not supported in case of CTI using CTIConnector.	If user sets Transfer type to consultation in case of CTI using SIPS.

Script ID Usage in the GVP 8 Environment

In Genesys VoiceXML 2.1, ScriptId refers to the script identifier, as generated by the CTI Connector, to handle call treatments. The use of ScriptId is specific to GVP 7.x and was mandatory for treatments. Since the GVP 7.x design is "IVR-centric," the treatment would be invoked on the same VXML session. Things are a bit different with GVP 8.x and the Next Generation Interpreter (NGI) where APP_URI is used instead of ScriptId and the treatments are executed on different VXML sessions. **GVP 8 and NGI** In GVP 8.x, request for treatment execution comes in as a NETANN request with the APP_URI being passed in as a VoiceXML parameter. GVP executes the requested page to kick off the treatment. Unlike the GVP 7.x environment, treatments get invoked as separate VXML sessions and terminated at the end of the treatment execution. Hence, ScriptId switching is no longer needed here, unless an application wants to do branching based on ScriptId.

- Note: Composer provides support for both SIPS and CTIC scenarios for achieving the CTI functionality. However, SIPS may not support passing additional request-uri parameters like ScriptId, therefore, this option is limited only to CTIC scenarios.

Please refer to *GVP 8.x VXML Help* under Sample Voice XML Applications > CTI Interactions > Treatments for more details on this topic.

Accessing ScriptId in Composer

Use if you want your application to do ScriptId-based switching like GVP 7.x. **CTIC Scenario (IRD strategy + Composer Callflow)**

1. Use the APP_ID property in IRD's Play Application block.
2. Define a new Input type variable named ScriptId in the Entry block of your callflow to collect the ScriptId.

Composer Workflow + Composer Callflow)

1. On the VXML callflow side, define a new Input type variable named ScriptId in the Entry block to collect the APP_ID (i.e., ScriptId) passed from the workflow.
2. On the SCXML workflow side, use the Play Application block to invoke the callflow created using step#1. Then do an auto-synchronize for the parameters, and specify the ScriptId value.
3. The ScriptId (i.e., APP_ID) passed from the workflow will be automatically collected on the VXML side from the session.connection.protocol.sip.requesturi array.

SIPS Scenario

1. SIPS may not support passing additional request-uri parameters. Pass ScriptId as attached data on the strategy side (If using IRD) or on the SCXML side (If using Composer workflows).
2. Define a new Input type variable named ScriptId in the Entry block to collect the ScriptId.
3. The ScriptId (i.e., APP_ID) passed from the strategy will be automatically collected on the VXML side from the session.com.genesyslab.userdata array.

Voice External Message Blocks

The External Messaging palette provides blocks for NGI extensions to send and receive external messages to/from external entities such as CCXML applications. There are four External Message blocks:

- **Receive Block** for receiving synchronous and asynchronous SIP INFO messages. This is can be used to receive messages from CCXML applications.
- **Send Data Block**, which is a wrapper around the <send namelist> tag) for sending a list of variables as SIP INFO to the other end point. The data is sent in the form-url-encoded format, in the BODY of the SIP INFO.
- **Send Info Block** generates the NGI VXML <send body> tag for sending any content in the Body of the SIP INFO. By default, content-type is set to text/plain. Typically, this can be used in conjunction will CCXML applications.
- **Send Event Block** generates the NGI VXML <send event> tag to send SIP INFO events or custom events between the VXML dialog and the CCXML application.

For all the Send [xxx] blocks, you have the option to specify the Wait for response property as true in those blocks to send the message synchronously.

Receive Block

Use the Receive block for receiving synchronous and asynchronous SIP INFO messages. This is can be used to receive messages from CCXML applications.

A typical use case is for a CCXML application to interrupt the VXML dialog in order to take some action.

Depending upon how the data is sent, the content, content-type or event properties will be filled.

The Receive block has the following properties:

The Receive block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Content Property

The Content property is the variable used to collect the content of the received event.

To select a variable:

1. Select the Content row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

Content Type Property

The Content Type property is the variable used to collect the content type of the received event.

To select a variable:

1. Select the Content Type row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

By default, Content Type is set to text/plain.

Event Name Property

The Event Name property is the variable used to collect the name of the received event.

To select a variable:

1. Select the Event Name row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Send Data Block

Use the Send Data block (a wrapper around the <send namelist> tag) for sending a list of variables as SIP INFO to the other end point. The data is sent in the form-url-encoded format, in the BODY of the SIP INFO.

Typically, Send Data can be used by VXML applications to send data to a CCXML application or to CTI applications.

For example, CCXML use cases that use Composer External Messaging Blocks (such as Send Data, Send Info, and Send Event), see the Genesys Voice Platform 8.1 CCXML Reference Manual. See the Features chapter, Dialogs section.

When using either the Send Data or Send Info block, the result on the CCXML side is to create a dialog.user. * event. The name of the event is set to dialog.<event name>.

Dialog User Event Example

The VoiceXML dialog may send a user event to the CCXML application by using the <send namelist="name type uri"/> tag. Here is an example of the VoiceXML <send> block:

```
<var name="name" expr="'transfer'"/>

<var name="type" expr="'bridge'"/>

<var name="uri" expr="'1111@205.150.90.19'"/>

<gvp:send namelist="name type uri"/>
```

The CCXML session receives the following:

```
15:02:04.416 Int 51030 F9187A00-E558-44C6-61AE-FFA9A066180C-FF326086-ECB5 dlg_event

7|dialog.user.transfer|DD92E8B2-51AD-4F3F-8C8D-

40AFA169EA9B|values.name="transfer";values.type="bridge";values.uri="1111@205.150.90

.19
```

This raises a dialog.user.transfer event to the CCXML application that owns the dialog. The event itself contains the following properties:

```
event$.values.name=transfer

event$.values.type=bridge

event$.values.uri=1111@205.150.90.19
```

Note: The event$ is a generic name for CCXML events, and in the preceding example, it is dialog.user.transfer. The contenttype attribute is not supported by the <send> tag if the namelist is used.

The Send Data block has the following properties:

The Send Data block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Values Property

The Values property holds the list of variables to be sent.

To select values:

1. Click the Values row in the block's property table.
2. Click the button to open the Values dialog box.
3. Select individual variables, or click Select all or Deselect all.
4. Click OK.

Wait For Response Property

The Wait For Response property allows a message to be sent synchronously (when set to true). By default, data is sent asynchronously for all the Send [xxx] blocks (when this property is set to false).

To assign a value to the Wait For Response property:

1. Select the Wait For Response row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Condition Property

Find this property's details under Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Send Event Block

Use the Send Event block, which generates the NGI VXML <send event> tag, to send SIP INFO events or custom events between the VXML dialog and the CCXML application. Examples: logging events or any event specific to the dialog and the CCXML application. For more information, see the Genesys Voice Platform 8.1 CCXML Reference Manual, Event/IO Processor, Sending Events.

The Send Event block has the following properties:

The Send Event block has no page exceptions.

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Event Name Property

The Event Name property is the variable used to collect the name of the sent event.

To select a variable:

1. Select the Event Name row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

Wait For Response Property

The Wait For Response property allows a message to be sent synchronously (when set to true). By default, data is sent asynchronously for all the Send [xxx] blocks (when this property is set to false).

To assign a value to the Wait For Response property:

1. Select the Wait For Response row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

Send Info Block

Use the Send Info block, which generates the NGI VXML <send body> tag, for sending any content in the Body of the SIP INFO. By default, content-type is set to text/plain.

Typically, this can be used in conjunction will CCXML applications.

For example, CCXML use cases that use Composer External Messaging Blocks (such as Send Data, Send Info, and Send Event), see the *Genesys Voice Platform 8.1 CCXML Reference Manual*. See the Features chapter, Dialogs section.

When using either the Send Data or Send Info block, the result on the CCXML side is to create a dialog.user. * event. The name of the event is set to dialog.<event name>.

For an example, see the Dialog User Event Example in the Send Data block description.

The Send Info block has the following properties:

The Send Info block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Content Property

The Content property is the variable used to collect the content of the sent event.

To select a variable:

1. Select the Content row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

Content Type Property

The Content Type property is the variable used to collect the content type of the sent event.

To select a variable:

1. Select the Content Type row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

By default, Content Type is set to text/plain.

Wait For Response Property

The Wait For Response property allows a message to be sent synchronously (when set to true). By default, data is sent asynchronously for all the Send [xxx] blocks (when this property is set to false).

To assign a value to the Wait For Response property:

1. Select the Wait For Response row in the block's property table.
2. In the Value field, select true or false from the drop-down list.

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks.

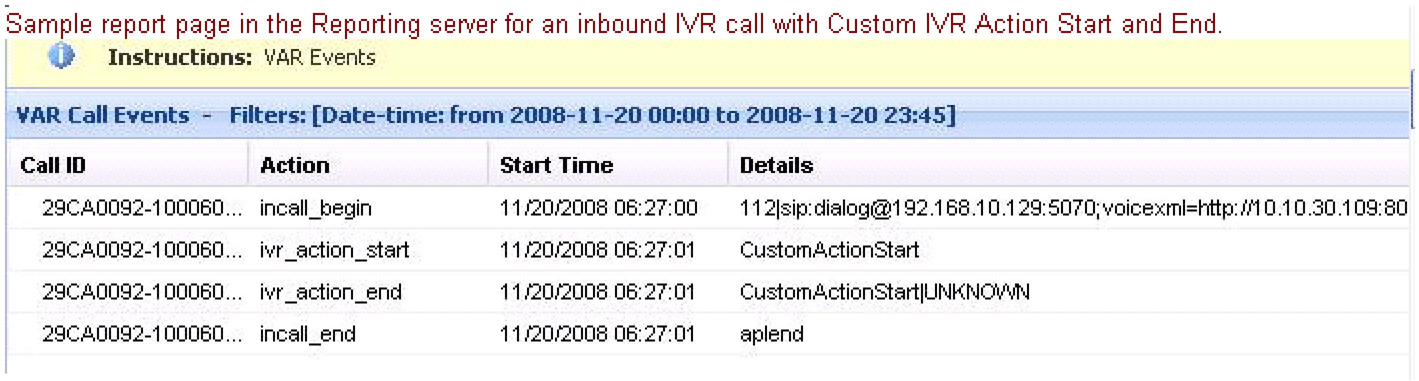
Reporting Blocks

Reporting Blocks provide interfaces for GVP and Reporting Server whenever the application needs to perform Voice Application Reporting for IVR actions. There are four Reporting blocks:

- **Action Start Block** indicates the start of a Voice Application Report (VAR) transaction.
- **Action End Block** allows the application to indicate the end of a Voice Application Report (VAR) transaction.
- **Set Call Data Block** allows the application to report custom data for the call.
- **Set Call Result Block** allows the application to indicate the end of a call.

Action Start Block

The Action Start block indicates the start of a Voice Application Report (VAR) transaction. You can specify the Action Id and Parent Action for the action. Composer generates Subcallflow start and End events whenever a <SubDialog> (Subcallflow) got executed in the call. Composer-generated VXML code automatically generates the events. With this feature all the events (Main and Sub callflow events) generated for a call can be found with in a single umbrella in the Reporting server.



Sample report page in the Reporting server for an inbound IVR call with Custom IVR Action Start and End.

Instructions: VAR Events

VAR Call Events - Filters: [Date-time: from 2008-11-20 00:00 to 2008-11-20 23:45]

Call ID	Action	Start Time	Details
29CA0092-100060...	incall_begin	11/20/2008 06:27:00	112|sip:dialog@192.168.10.129:5070;voicexml=http://10.10.30.109:80
29CA0092-100060...	ivr_action_start	11/20/2008 06:27:01	CustomActionStart
29CA0092-100060...	ivr_action_end	11/20/2008 06:27:01	CustomActionStart|UNKNOWN
29CA0092-100060...	incall_end	11/20/2008 06:27:01	aplend

The Action Start block has no page exceptions.

The Action Start block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Action Id Property

Note: The GVP 8 platform provides an extension to the <log> tag that allows application developers to indicate the start of an IVR Action The Action Id and Parent Action Id properties are used for this purpose. The syntax is as follows:

<log label="com.genesyslab.var.ActionStart">actionID[|parentID=<PID>]</log>

The Action Id property specifies a variable containing the name of the IVR action to report as being started. The actionID is any valid UTF8 string that does not contain spaces or pipes, and is restricted to a maximum of 64 characters.

1. Select the Action Id row in the block's property table.
2. In the Value field, select one of the available variables from the drop-down list.

Parent Action Property

See Note in Action Id property description.

If an IVR action is to be nested inside some other active action, then the parent action’s ID must also be included (PID). The Parent Action property specifies the variable used for the name of the parent action in which the new Action has to be contained.

1. Select the Parent Action row in the block's property table.
2. In the Value field, select one of the available variables from the drop-down list that contains the identifier for the Parent Action.

Important! If the Parent Action ID specified does not refer to an action that was already started, the Genesys Voice Platform Reporting Server will ignore the entire Action Start request.

Note: If the Parent Action ID specified does not refer to an action that was already started, the GVP Reporting Server will ignore the entire Action Start request.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Action End Block

The Action End block allows the application to indicate the end of a Voice Application Report (VAR) transaction. You can specify the reason, results and notes corresponding to the action. Composer generates Subcallflow start and End events whenever a <SubDialog> (Subcallflow) got executed in the call. Composer-generated VXML code automatically generates the events. With this feature all the events (Main and Sub callflow events) generated for a call can be found with in a single umbrella in the Reporting server.

You are responsible for making sure to provide a valid Action Id name, for an action that was previously started in the application using the Action Start block.

By default an action end event will be sent by each terminating block of a callflow. This includes the Exit and Disconnect blocks.

The Action End block has no page exceptions.

The Action End block has the following properties:

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Action Id Property

The Action Id property is the variable used in the Action Start block for the action to report as ended. It must be the same Action Id variable used in the Action Start block.

To select a variable:

1. Select the Action Id row in the block's property table.
2. In the Value field, select one of the available Property variables from the drop-down list.

Notes Property

The Notes property allows you to enter text (up to 4 KB of data) associated with the Action End event. Since Composer generates <log> labels for the Reporting blocks, text entered here can appear on voice application reports as described in the Genesys Voice Platform 8.1 User's Guide. See Provisioning GVP.

To enter notes:

1. Click the Notes row in the block's property table.
2. Click the button to open the Notes dialog box.
3. Type text notes as needed and click OK.

Reason Property

The Reason property allows you to enter text for a reason for ending the action. The Reason field allows up to 4 KB of data. Note text appears on voice application reports.

To enter reason text:

1. Click the Reason row in the block's property table.
2. Click the dropdown arrow and select the variable that contains the reason text.

Result Property

The Result property contains the result of the action that was just ended.

To assign a value to the Result property:

1. Select the Result row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

UNKNOWN

The action had an unknown result.

SUCCESS

The action completed successfully.

FAILED

The action did not complete successfully (failed).

Condition Property

Find this property's details under Property Common Properties for Callflow Blocks or Property Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Details Property Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Level Property Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Set Call Data Block

The Set Call Data block allows the application to report custom data for the call. You can select the list of variables to be reported. The name of the variable is used as the CustomData key. If eight keys are provided, the Reporting server will reject the data for any new keys received after that.

The Set Call Data block has no page exceptions.

The Set Call Data block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Variables Property

Use the Variables property to create custom variables. Variable content appears on GVP Voice Application reports (the VAR CDR Details Report). For more information, refer to the **Per-Call IVR Actions Report** section on page 367 in the *GVP 8.1 User Guide*. To create custom variables:

1. Click the **Variables** row in the block's property table.
2. Click under **Value** to add an entry to define application variables.
3. In the **Application Variables** dialog box, click **Add**.
4. In the **Variable Name** field, accept the default name or change it.
5. In the **Value** field, select a variable from the drop-down list.
6. In the **Description** field, type a description for this variable.
7. Click **Add** again to enter another parameter, or click **OK** to finish.

Delete Button

To delete a custom variable:

1. Select an entry from the list.

2. Click **Delete**.

Note: In version 8.1.300.xx, ignore the **Restore System Variables Default Values** button.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Set Call Result Block

The Set Call Result block allows the application to indicate the end of a call. You can specify the reason, results and notes corresponding to the call result. In addition to tagging calls for Voice Application Reporting (VAR), you can also use this block for Service Quality Analysis (SQA) call status (success, failure) reporting. For information on SQA, see Genesys Voice Platform 8.1 Deployment Guide and Genesys Voice Platform 8.1 User's Guide.

The Set Call Result block has the following properties:

The Set Call Result block has no page exceptions.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Can be used for both callflow and workflow blocks to add comments.

Notes Property

The Notes property allows you to enter text (up to 4 KB of data) associated with the end of a call. Since Composer generates <log> labels for the Reporting blocks, text entered here can appear on voice application reports as described in the Genesys Voice Platform 8.1 User's Guide. See Provisioning GVP.

To enter notes:

1. Click the Notes row in the block's property table.
2. Click the ⋯ button to open the Notes dialog box.
3. Type text notes as needed and click OK.

Reason Property

The Reason property allows you to enter text for a reason for ending the call (maximum length of 256 characters).

To enter reason text:

1. Click the Reason row in the block's property table.
2. Click the dropdown arrow and select the variable that contains the reason text.

Result Property

The Result property contains the result of the call that was just ended.

To assign a value to the Result property:

1. Select the Result row in the block's property table.
2. In the Value field, select one of the following from the drop-down list:

UNKNOWN

SUCCESS

FAILED

UNKNOWN

The call had an unknown result.

SUCCESS

The call completed successfully.

FAILED

The call did not complete successfully (failed).

This property can be used for reporting both VAR metrics and SQA services as described above. Refer to Genesys Voice Portal documentation for information usage of this field for VAR (<log> label com.genesyslab.var.CallResult) and SQA (<log>label com.genesyslab.quality.failure).

Notes:

- Composer will not log SUCCESS and UNKNOWN call results, already available for VAR, to SQA.
- MCP will still log a call as a failure if it fails to meet one of the thresholds, even if the application never explicitly calls the <log> tag to indicate SQA failure.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Server-Side Common Blocks

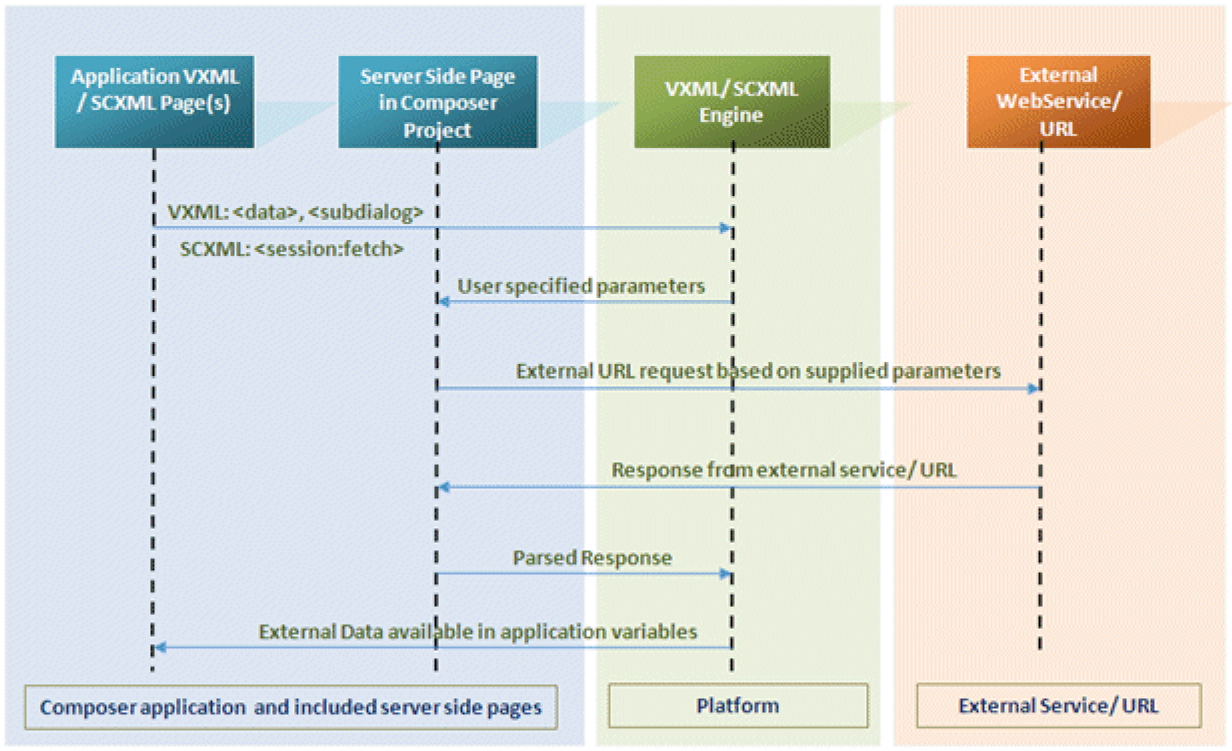
Both routing and voice applications use the Server-Side blocks.

- **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.
- **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)
- **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.
- **TLib Block** (route only). Use this block in workflows and sub-workflows that will use <session:fetch> method="tlib". The block exposes properties to form a TLib request to set agent status not ready equivalent to TAgentSetNotReady. It also sets srcexpr and <content> element to make it possible to form generic TLib requests.

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.

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	
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 the Web Request blocks. What can I use?	Backend block	Try starting with the Backend block since the implementation is 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.

Backend Common Block

The Backend block is used for both routing and voice applications. Use to invoke custom backend Java Server Pages (JSP). You have the option to pass back all the application session state data to the backend logic page on the server. Data being returned will be sent back as a JSON string. Other features:

- Provides a mechanism for creating new backend logic JSP. The added JSP file will have a basic template code already filled out. As the application developer, you will only need to implement a performLogic function. The VXML/SCXML to return back control will be auto-generated in the template.
- User-written custom backend logic pages are stored in the Java Composer Project's `src` folder. Composer provides standard include files for Backend logic blocks in the Java Composer Project's `include` folder.

Note: If any custom backend logic pages use libraries, place the libraries in the Java Composer Project's `WEB-INF/lib` directory. This directory typically contains JAR files that contain Java class files (and associated resources) required for the application. Note: The Tomcat application server should be restarted after changing any JAR files in this folder.

- Composer includes a CHEAT SHEET for creating a Backend logic application as well.

The Backend block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for voice blocks or Common Properties for Workflow Blocks.

Uri Property

The Uri property specifies the http:// page to invoke. To set a URL destination for the Uri property:

1. Select the Uri row in the block's property table.
2. In the Value field, click the button to open the Uri dialog box.
3. Select a file from the available projects.

Encoding Type Property

The Encoding Type property (used for callflows only) indicates the media encoding type of the submitted document. GVP 8.1 supports two encoding types:

- application/x-www-form-urlencoded
- multipart/form-data

To select a value for the Encoding Type property:

1. Select the **Encoding Type** row in the block's property table.
2. In the **Value** field, select **application/x-www-form-urlencoded** or **multipart/form-data** from the drop-down list.

Parameters Property

Note: Parameters cannot be entered until the Uri property is specified. Use the Parameters property to specify parameters to pass to the invoked backend JSP. To specify parameters:

1. Click the Parameters row in the block's property table.
2. Click the button to open the Parameter Settings dialog box.

Add Button Use the Add button to enter parameter details.

1. Click **Add** to add an entry to Backend Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the Backend application.
Out	Output parameters are variables that the Backend application returns and will be reassigned back to the current callflow.

InOut	InOut parameters are parameters that act as both input and output.

4. In the **Expression** drop-down list, select from among the listed variables, type your own expression, or click the ▭ button to use Skill Expression Builder.
5. In the **Description** field, type a description for this parameter.
6. Click **Add** again to enter another parameter, or click OK to finish.

Delete Button To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

Pass State Property

Note: This property is used for callflows only. The Pass State property Indicates whether or not to pass the application state to the backend. The application state includes all the variables shown in the Entry block as well as all variables containing returned values from user Input blocks. You can find Instructions on how to access these backend variables in Creating a Backend JSP File and Creating a Backend ASP.NET File. The Parameters property can also be used to pass specific parameters into the backend and, for efficiency reasons, should be considered first. There is also a Cheat Sheet, Creating a Backend Logic Block (**Help** > **Cheat Sheets** > **Composer** > **Building Voice Applications**). To select a value for the Pass State property:

1. Select the **Pass State** row in the block's property table.
2. In the **Value** field, select **true** or **false** from the drop-down list.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Business Rule Common Block

Business Rules

Composer interfaces with the Genesys Rules Engine, which is part of the Genesys Rules System. A business rule is an external piece of business logic, which can be customized, and then invoked by Genesys applications. Here is an example business rule for a bank: `IF product = 'mortgage' and loanAmount >=200000 THEN TTSMsg = 'You must have a credit score of 300 or great to qualify for this loan.'` To simplify rule creation, the Genesys Rules System uses Rule Templates. These are initially created by developers and IT professionals. A Composer-compatible plug-in is available for developing business Rule Templates. This plug-in is provided as part of the Genesys Rules System. For information on installing the plugin, refer to the *Genesys Rule System 8.1 Deployment Guide*. See Chapter 2, Installation. Once validated and deployed, Rule Templates are available for customization in the Genesys Rules Authoring Tool GUI. Business analysts then use the templates to create related sets of business rules called Rule Packages. Packaging rules together allows the business analyst to define which rules will support a particular application. You can use Composer's Business Rule block to request the Genesys Rules Engine to execute a Rule Package in a routing workflow or voice callflow and write the results back to a variable. A business rule preference specifies the Genesys Rules Authoring Tool server to work with. Find information on using the business rules GUI in the following documents:

- *Genesys Rules System 8.1 Deployment Guide*
- *Genesys Rules System 8.1 Rules Authoring Tool Help*
- *Genesys Rules System 8.1 Rules Development Tool Help*

Note: In the Genesys 8.1 release, the Genesys Rules System is packaged only with the intelligent Workload Distribution product and the Conversation Manager product.

Business_Rules_Preferences

The preferences entered here are used in the Business Rule block, Business Rule Package property. To set Business Rules Preferences:

1. Select **Window** > **Preferences** > **Composer** > **Business Rules**.
2. Configure the connection to the Genesys Rules Authoring Tool (GRAT) server by entering the following fields:
    - **GRAT Server**. Enter the address of the Application server hosting the GRAT Server. When using the Business Rule Package property in the Business Rule block, Composer will connect to this server to query information about packages and rules. Example only: http://ca-to-lennon:8080.
    - **Server Path**. Enter the name of the web application deployed as the GRAT. For example, if you have the GRAT running at http://ca-to-lennon:8080/genesys-rules-authoring, then the

GRAT server is http://ca-to-lennon:8080 and the Server Path is /genesys-rules-authoring.

- **Tenant**. To obtain a list of Rule Packages, Composer will query the GRAT server using an HTTP request to http://{server-address:port}/tenant/packages. Enter the name of the tenant as defined in the Configuration Database.
- **Username**. Enter the username defined in the Configuration Database for logging into the GRAT server.
- **Password**. Enter the password defined in the Configuration Database for logging into the GRAT server.
- **Genesys Rules Engine (Optional). GRE URL**. Enter the URL for the GVP Debugger to use when starting a call. The GRE URL will be passed to the VXML application in the SIP URL. If set, this value will be passed to the voice or routing application and will override the value set in the Rules Engine URL property of the Business Rule Block (see that section below).

Business Rule Templates

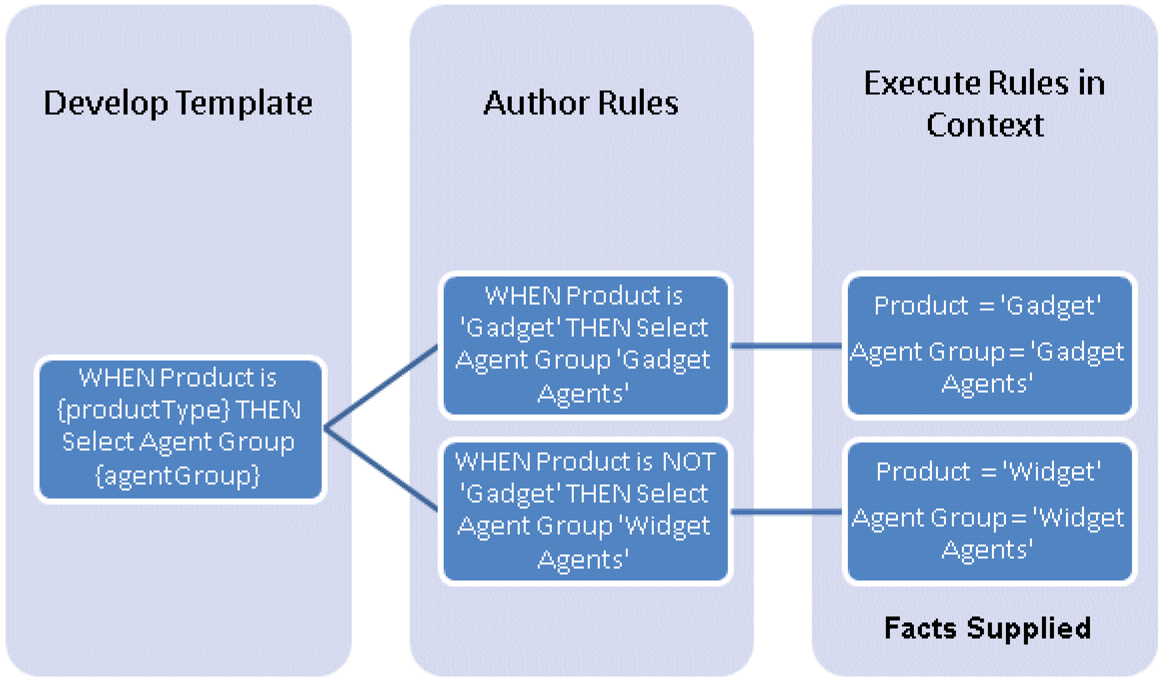
This functionality is enabled by an Eclipse plug-in that can be installed within Composer or in a standalone Eclipse environment.

- To install the plugin, refer to the *Genesys Rule System 8.1 Deployment Guide*. See Chapter 2, Installation.

The plug-in enables developers to create Rule Templates. Rule Templates consist of rule parameters, conditions, actions, and functions. When a Rule Template is published to the Rules System repository, it is made available to be added to Rule Packages. Rule Packages are the deployable objects, which are used to expose rule conditions and actions to business users for creating rules through the Genesys Rules Authoring tool. A brief summary of Rule Templates is presented below. For detailed information, see the *Genesys Rules System 8.1 Rules Development Tool Help*. Once you install the plugin, this help system is available within Composer by selecting **Help** > **Contents**.

Genesys Rules System Architecture

A logical view of the Genesys Rules System architecture is shown below.


Develop Template
Author Rules
Execute Rules in Context
WHEN Product is {productType} THEN Select Agent Group {agentGroup}
WHEN Product is 'Gadget' THEN Select Agent Group 'Gadget Agents'
WHEN Product is NOT 'Gadget' THEN Select Agent Group 'Widget Agents'
Product = 'Gadget'
Agent Group = 'Gadget Agents'
Product = 'Widget'
Agent Group = 'Widget Agents'
Facts Supplied

- The first category reflects Rule Template creation, which can be done in Composer if the set of plugins is installed.
- The second category reflects rule creation by business analysts in the Genesys Rules Authoring Tool.
- The third category reflects rule evaluation by the Genesys Rules Engine using the Business Rule block once the Facts are known.

Type of Rules

The Genesys Rules System supports both basic and decision table rules.

Basic Rule

A basic or linear business rule is of this form: WHEN {condition} THEN {action} In other words, when the condition is true, the action will occur. This is a rule template. When a business analyst uses the Genesys Rules Authoring Tool to customized a template with valid values, this creates a business rule. The following rules are all valid instances:

- `WHEN Product = 'Gadget' THEN Select Agent Group 'Gadget Agents'`
- `WHEN Product = 'Widget' THEN Select Agent Group 'Widget Agents'`
- `WHEN Customer Segment = 'Gold' THEN Assign Credit Limit '200000'`

This form of rule is preferred for simple actions, such as assigning a value to return back to the application.

Decision Table Rule

A business rule can also take form of a decision table. For example, assume in a particular scenario that there are 3 customer levels: Gold, Silver, and Bronze. For each of these levels, we wish to make an offer to customers based on a qualifying purchase they may have made. Gold customers automatically qualify for a Premium Offer. Silver customers need to have spent $1000 or more to qualify for that offer, otherwise they get the Special Offer. Finally, Bronze customers need to have spent $5000 or more for the Premium Offer; or $2000 or more for the Special Offer; otherwise they are informed of the offers available if they make the qualifying purchase level. **Note:** The Genesys Rules Engine cannot execute Rule Templates.

Business Rule Block

Once the Rule Package (created from Rule Templates) that you want to work with are deployed to the Genesys Rules Engine, you can use the Business Rule block on the Server Side palette to create voice and routing applications that use business rules. Use this block to have Composer query the Genesys Rules Authoring Tool (GRAT) for deployed packages. 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.   **Note:** This last step (evaluation) happens as part of a VXML or SCXML application that Composer developer creates, not as part of Composer. A business rule preference specifies the Genesys Rules Engine to work with. **Runtime Parameters** The following parameters (defined in Preferences) are used at runtime, when the VXML and SCXML application queries the GRAT to execute the rule.

- `grat_username` -- a user login for accessing the GRAT server
- `grat_password` -- the password for the above login
- `grat_server` -- a URL for the GRAT server, for example: http://hostname:8080/genesys-rules-authoring
- `grat_tenant` -- the tenant associated with the login, e.g. Environment

The Business Rule block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Business Rule Package Property

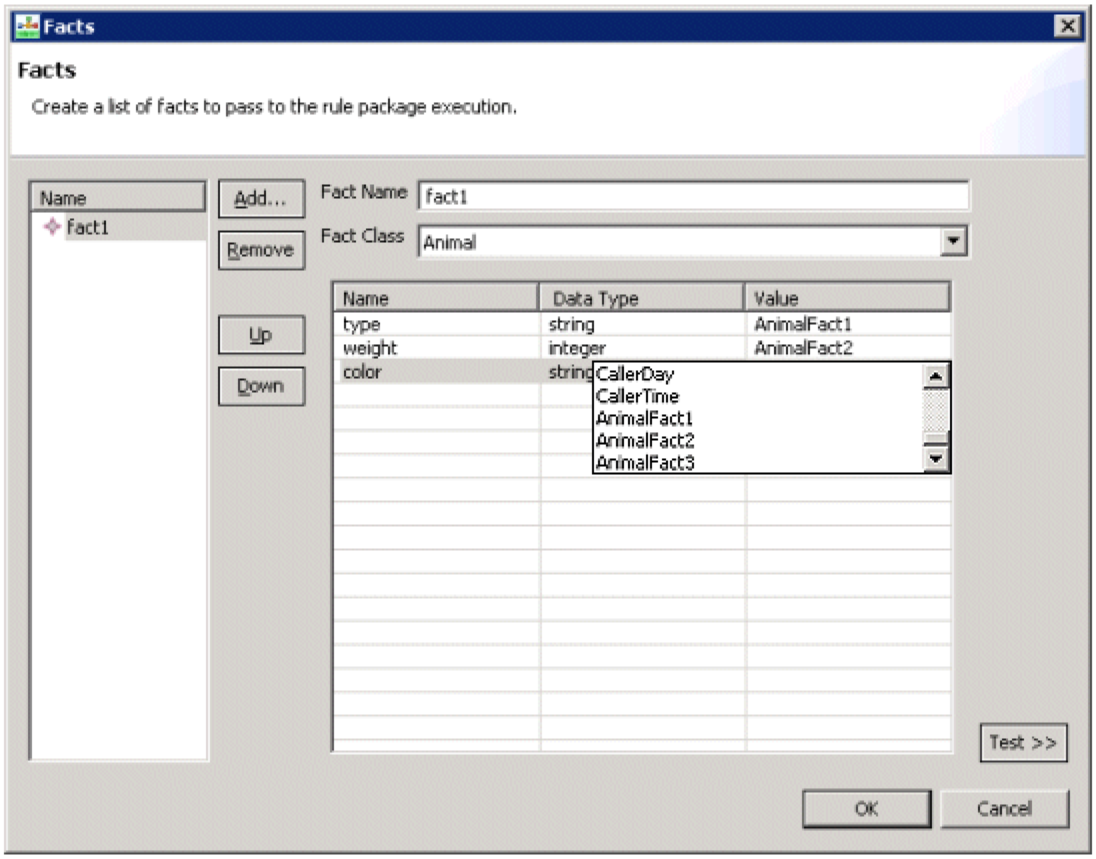
Use to select the Rule Package (collection of related rules) you would like to execute. Packaging rules together allows the business analyst to define which rules will support a particular application. Before using this property, you must set Business Rules Preferences.

1. Click the button to request Composer to connect to the Genesys Rules Authoring Tool Server using the information specified in Business Rule Preferences. After a successful connection, the Business Rule Package dialog appears.
2. Select a Rule Package and click **OK**. The dialog closes and the name of the Rule Package appears under Value.

Facts Property

Use this property to execute the logic contained in the selected Rule Package by supplying input parameters called *Facts*. To specify Facts:

1. Click the **Facts** row in the block's property table.
2. Click the button to open the Facts dialog box.
3. Click **Add**. The dialog box adds additional fields consisting of the Facts to use when executing the Rule Package. You then click the down arrow and select a value or a variable that contains the value for each Fact. An example dialog box is shown below.



Facts

Create a list of facts to pass to the rule package execution.

Fact Name: fact1

Fact Class: Animal

Name	Data Type	Value
type	string	AnimalFact1
weight	integer	AnimalFact2
color	string	

Facts.gif

4. Enter the **Fact Name** field.
5. Click the down arrow and select an entry for the **Fact Class** field.
6. Click the down arrow and select a value or a variable that contains the Fact value.
7. Click **Add** again to enter another Fact, or click **OK** to finish.

Delete Button To delete a Fact:

1. Select an entry from the list.
2. Click **Delete**.

Rules Engine URL

Select the variable containing the Genesys Rules Engine URL. Background: Starting with 8.1.2, Composer-generated applications no longer interact with the GRAT server at runtime. Previous requests to the GRAT Server were done to retrieve the URL of the GRE server to which a rules package is deployed. Instead, the runtime applications now use the Rules Engine URL property, which is passed into the application via the IVR Profile or an Enhanced Routing Script object. You can use this Rules Engine URL property to override any GRE URL configured in the IVR Profile or

EnhancedRouting Script object.

Exceptions Property

The Business Rule block supports the following exceptions. They correspond to the HTTP status codes returned by the Business Rule Server (BRS).

Exception Event Name	**HTTP Return Code**	**BRS Error Code**	**Description**
error.com.genesyslab.composer.badrequest	400	610	The received URI does not match the Engines REST specification.
error.com.genesyslab.composer.notfound	404	620	The package for the evaluation request received was not found.
error.badfetch.http			Any other HTTP error.
error.com.genesyslab.composer.notacceptable	406	602	The evaluation request received could not be converted to a valid knowledgebase-request message, or the evaluation request received could not be evaluated due to an exception.

Details of the exception can be obtained from the body of the response. The Composer application will log the description. The JSON body of the response will look like the following: { `error:{ code:6xx, description:error message` } } Also see Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Applicable to workflows only. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction. Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Output Result Property

Use this property to save the results of the business rule execution to a variable. To select a variable:

1. Select the **Output Result** row in the block's property table.
2. In the Value field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

The format of returned data is JSON. Any post-processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It supports writing simple or complex expressions to extract values out of JSON strings and arrays. In a workflow, the Output Result can be attached to User Data. In the Specify Output Result Location dialog box, select

User Data or Variable. If User Data is selected, the specified name is used as a prefix of the keys that will be added to user data. For example, if you specify abc, then the User Data will look like:

```
  'abc_fact1'(list) '@class'      'com.genesyslab.animals.Animal'
                                  'color'       'red'
                                  'type'        1903
                                  'weight'      123                   'abc_fact2'(list)
'@class'     com.genesyslab.animals.Car'                                      'make'
       'mazda'
```

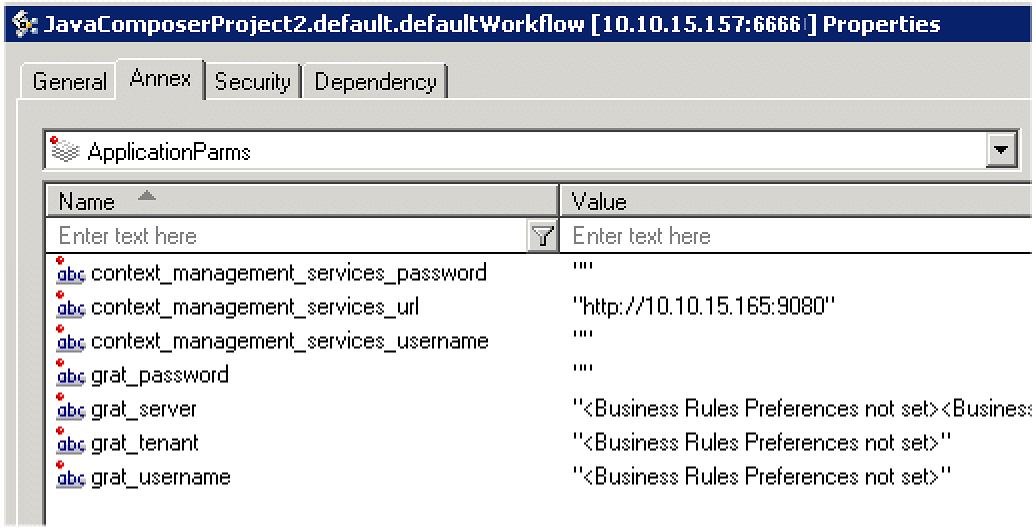
Note: The Output Result property takes effect only during application runtime. Its purpose is to take the output of the rule execution (at runtime) and store returned results back in the specified application variable so other parts of the application can access the data.

Business Rules Block Runtime Configuration

The table below shows the parameters that must be set up in Genesys Administrator in order for the Business Rules block to work.

	ERS Object Key Names	IVRprofile Object Key Names
GRS	grat_server	grat_server
	grat_tenant	grat_tenant
	grat_username	grat_username
	grat_password	grat_password

The figure below shows an example Enhanced Routing Script object created by Composer. It creates these parameters in the ApplicationParms section in the Annex, so you do not have to key in parameter names. Note: If you accidentally changes parameter names, these functions will not work.



JavaComposerProject2.default.defaultWorkflow [10.10.15.157:6666] Properties

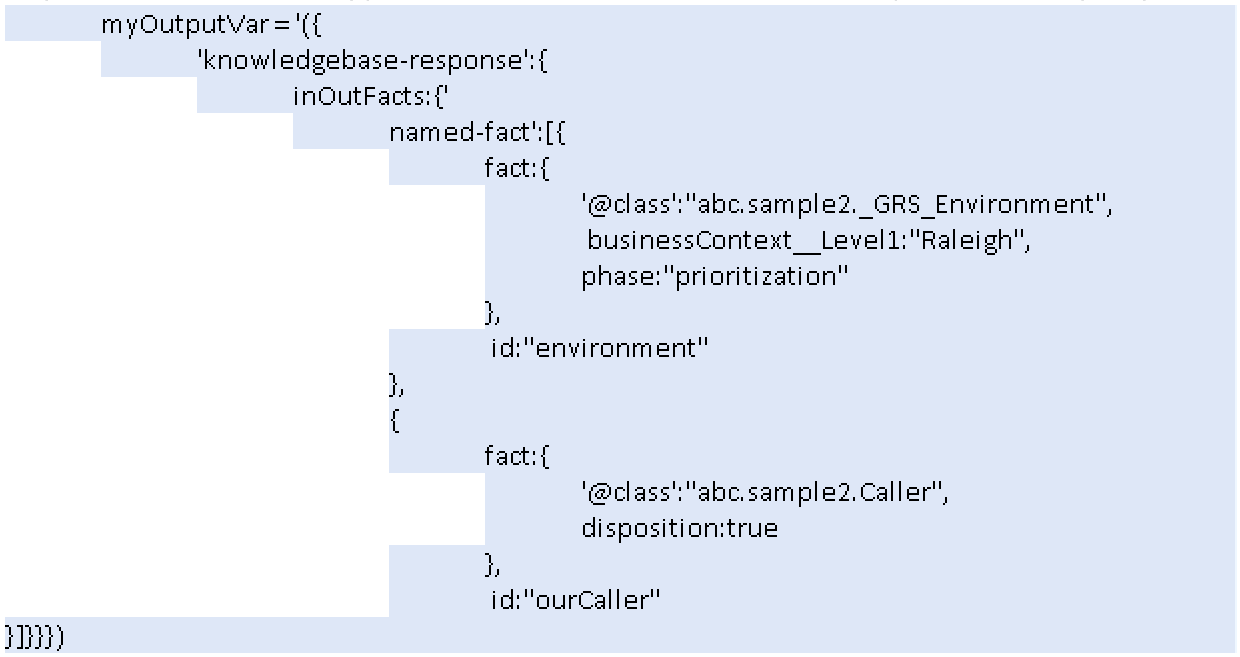
General | Annex | Security | Dependency

ApplicationParms

Name	Value
Enter text here	Enter text here
context_management_services_password	""
context_management_services_url	"http://10.10.15.165:9080"
context_management_services_username	""
grat_password	""
grat_server	"<Business Rules Preferences not set><Business
grat_tenant	"<Business Rules Preferences not set>"
grat_username	"<Business Rules Preferences not set>"

Working With Returned Data

Below is an example on how to work with data returned by the Business Rules block. A sample of the output can look like the snippet below, which will be stored in the output variable myOutputVar.



```
        myOutputVar='({
                'knowledgebase-response':{
                        inOutFacts:{'
                                named-fact':[{
                                        fact:{
                                                '@class':"abc.sample2._GRS_Environment",
                                                 businessContext__Level1:"Raleigh",
                                                phase:"prioritization"
                                        },
                                         id:"environment"
                                },
                                {
                                        fact:{
                                                '@class':"abc.sample2.Caller",
                                                disposition:true
                                        },
                                         id:"ourCaller"
}]}}})
```

To extract the value of the disposition field, an expression like this can be used: `myDisposition = myOutputVar["knowledgebase-response"].inOutFacts["named-fact"][1].fact.disposition` This will return true.

DB Data Common Block

Important Note: When using the DB Data block, database errors (such as failure to connect to a database) could result in an invalid JSON message being returned to the workflow or callflow, which could cause an application to fail. In order to avoid this potential issue, Genesys recommends upgrading to Release 8.1.301.02 as it contains an update to the database access library.

The DB Data block is available for both routing and voice applications. 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 Using the Query Builder or Stored Procedure Helper. It can also use a SQL script file. **Note:** When using the DB Data block to connect to and query information from an Oracle database, some connections may remain in the TIME_WAIT state. If you encounter this situation, use connection pooling in order to avoid exhausting the number of allowed Oracle connections. This block acts as a data source for the DB Prompt and DB Input blocks (available only in callflows). An Entry block user variable can also be used to access the results of a Stored Procedure call specified in a DB Data block for both voice and routing applications. **Note:** The Looping block can work with the DB Data block. For example, you can use the Looping block to Iterate over a data set returned by the DB Data block to map values returned from a database query to application variables. Also see: Working with Database Blocks. The DB Data block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. **Note:** if you rename a DB Data block, its corresponding SQL statement file in the db folder will not be updated and will not be valid until you generate code again.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Connection Profile Property

The Connection Profile property allows you to select a previously-created database Database Connection Profiles that specifies database details for this DB Data block. If you have not created a connection profile, open the Connections Profile editor as follows:

1. Under **Value**, click the down arrow.
2. Select **Create New Profile Using Editor...**

Refer to the topic Database Connection Profiles for instructions. To select a connection profile for your database query:

1. Select the **Connection Profile** row in the block's property table.
2. Select the connection profile to use for this query.

Connection Properties Property

The Connection Properties property allows you to override the parameters in connection profile during runtime. The properties that can be overriden are Hostname, Password, Port, Database, Username and other Custom Parameters. Variable mapping can be configured in the dialog box provided for the property. To define the variable mapping for Connection Parameters:

1. Click the button to open the Connection Properties variable mapping dialog.
2. Dialog displays the parameter name and value in connection profile. Select the system variable in the drop down combo against each property.
3. Click OK

Connection String Property

The Connection String Property allows you to define the value of Connection String that need to be used at Runtime. If this property is specified, the parameters from Connection Profile is ignored. To define this property enter either literal value or select system variable from the combo provided for the property.

Timeout Property

The Timeout property defines the length of time in seconds that the voice application will wait for query execution to complete. To provide a timeout value**:**

1. **Select the** Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

The default value (20 seconds) of this property is used if not specified explicitly. Disable the timeout by setting to -1. If the query takes longer than this specified time to complete the `error.com.genesyslab.composer.dbtimeout` exception is thrown. In order to select a query type, the Connection Profile property must be set.

Query Type Property

To define a query type:

1. Select the **Operation Type** row in the block's property table.
2. Select one of the following:
   - **SQLQuery**
   - **SQLScriptFile**
   - **StoredProcedure**

Based on the value selected for **Operation Type**, the specified value is used and some properties are not used. Query Property The Query property opens the Query Builder in which you can visually build the database query. **Note:** The Query property and Query File property are mutually exclusive. If both are entered, then the Query File property takes precedence over the query defined in the Query property. To define a query:

1. Select the **Query** row in the block's property table.
2. Click the ▭ button to open the Query Builder.

Query File Property

The Query File property accepts a filename that points to a SQL file that the user has written. To provide a filename for a user-written SQL file:

1. Select the **Query File** row in the block's property table.
2. In the Value field, type the filename of the SQL file (the file is usually in the db folder of your project. If it is present in a different location, specify a relative path, such as `../myfolder/myquery.sql`.

Stored Procedure Property

The Stored Procedure property opens the Stored Procedure Helper in which you can visually build the database query. To define a stored procedure call:

1. Select the **Stored Procedure** row in the block's property table.
2. Click the ▭ button to open the Stored Procedure Helper.

Column Names Variable Property

The Column Names Variable property maps the list of column names in the result to the specified variable. The default is Use system default, in which case the system uses an internal variable which is named in the format below. Genesys recommends that you define a user variable for this purpose in the Entry block and specify it in the DBData block. For Callflow diagrams: AppState.<blockname>DBResultColumnsNames For Workflow diagrams: App_<blockname>['DBResultColumnsNames'] To select a variable:

1. Select the **Column Names Variable** row in the block's property table.
2. In the **Value** field, select the variable from the dropdown list.

Records Variable Property

The Records Variable property maps the records (data) in the result set to the specified variable. The default value is Use system default, in which case the system creates an internal variable which is named in the format below. However, Genesys recommends that you specify a user variable in the Entry block. For Callflow diagrams: AppState.<blockname>DBResult For Workflow diagrams: App_<blockname>['DBResult'] To select a variable:

1. Select the **Records Variable** row in the block's property table.
2. In the **Value** field, select the variable from the dropdown list.

Note: The following applies to all methods of getting database results (query builder, stored procedure helper, custom queries): Results are stored in a variable as a two-dimensional JSON array. This data can then be accessed via a Looping block or via scripting in the Assign or ECMAScript block. For example, if the database result set looks like this in tabular form:

Vegetables	Animals
lettuce	chicken
broccoli	lion

The JSON for the result will look like this: `{"db_result":[["lettuce", "chicken"], ["broccoli", "lion"]],"db_result_columns":["vegetables", "animals"]}`

Suppress Empty Result Set Exception Property

The Suppress Empty Result Set Exception property determines if the dbemptyresultset exception should be thrown if a query or a stored procedure execution results in an empty result set (number of records returned is zero). To provide a value:

1. Select the **Suppress Empty Result Set Exception** row in the block's property table.
2. Select **true** or false.

Exceptions Property

Find this property's details under Common Properties for voice blocks or Common Properties for Workflow Blocks. The Exceptions dialog box for the DB Data block has the following exception events:

- `error.com.genesyslab.composer.dbconnectionerror`
- `error.com.genesyslab.composer.dberror` (pre-selected in the Supported column)

- `error.com.genesyslab.composer.dbemptyresultset` (pre-selected in the Supported column)
- `error.com.genesyslab.composer.dbtimeout`

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

External Service Block

This block 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. Can be used for both voice and non-voice interactions. **Notes:**

- In order to use this object, the third party server/application must already be defined in the Configuration Database as a server of type Third Party Server or Third Party Application. Before completing the External Service block properties, you must already know the names of Services, Methods, and Signatures (requested input/output parameters) provided by the external service.
- The Composer External Service block does not automatically pass user data in the ESP call unlike the legacy IRD External Service object. Therefore, ESP methods that expect user data cannot be called using this block. Please refer to the ESP method/API documentation to determine if user data is required. To call an ESP API that requires user data, a hand coded SCXML page can be used and invoked using the SubRoutine block. Please refer to the <session:fetch> documentation in the Orchestration Server Developers Guide. See Action Elements under Session Interface for details on how to pass user data in ESP requests.

Use Case

A customer has a custom integration to a third party application (a workflow system), through the Open Media API. The workflow system uses Genesys to distribute work items at various times during the workflow. At some point in the IPD handling a work item, there is a need to update the workflow system and assign a new value to one of the attributes of the work item. The Genesys developer has the IPD call a routing strategy, which uses the External Service block to call a specific method exposed by the third party application. This allows the developer to update the value of the specific attribute of the work item. The External Service block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Application Property

Use this property to select the name of the third party application to be contacted or the general application type to be contacted, which must be defined in the Configuration Database.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. Select the third party application to be contacted.
4. Click **OK**.

Method Name Property

Use this property to specify the Method defined by the third party server or application.

1. Click under **Value** to display the button.
2. Click the button to open the Method Name dialog box.
3. Opposite **Type**, select one of the following as the source for the name:
    - **Literal** to enter the method name manually in the Value field.
    - V**ariable** to select a variable for the method name in the Value field.
4. Click **OK** to close the dialog box.

Method Parameters Property

Use this property to specify the list of input parameters to be passed to the specified external service. Click the button to add a new entry:

1. Click under **Value** to display the button.
2. Click the button to open the Method Parameters dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite **Key**, leave **Literal** in the first field and enter the input parameter name in the second field.

5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
6. Click **OK** to close the Select Items dialog box. The Method Parameters dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the Method Parameters dialog box. .

Service Name Property

Use this property to specify the name of the Service defined by the third party server or application for the functionality requested.

1. Click under **Value** to display the button.
2. Click the button to open the Service Name dialog box.
3. Opposite **Type**, select one of the following as the source for the name:
    - **Literal** to enter the service name manually in the Value field.
    - **Variable** to select a variable for the service name in the Value field.
4. Click **OK** to close the dialog box.

Service Timeout Property

Use this property to specify the timeout in seconds (s) to be used for invoking this method. If not checked, URS uses the Reconnect Timeout entered for third party server or application in Configuration Server. In the case of a connection or service request failure, error codes are returned. The default is 30 seconds.

User Data Property

Use this property to specify the list of User Data parameters to be passed to the specified external service. Click the button to add a new entry:

1. Click under **Value** to display the button.
2. Click the button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.

5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Result Property

Use this property to specify an application variable to store the results. These results will then be available in other blocks in the application for further processing. The format of returned data is JSON. Any post processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It already supports writing simple or complex expressions to extract values out of JSON strings and arrays.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OPM Common Block

The OPM block 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). The OPM block is available for both Callflows (VXML) and Workflows (SCXML).

The OPM block allows browsing through a JSON structure based on metadata retrieved from the GAX server in order to form a JSON expression. The OPM block generates code to evaluate the specified expression and assign results to the **App_OPM** (voice application) or **system.OPM** (routing application) application variable accessible via the Entry block.

GAX Server

GAX refers to a Genesys Administrator Extension (GAX) plug-in application used by Genesys web application EZPulse. EZPulse enables at-a-glance views of contact center real-time statistics in the GAX user interface. A button on the Composer main toolbar, Launch GAX Server Command, lets you launch the Genesys Administrator Extension used by the GAX Server. Composer uses the server host, port, username, and password entered on the GAX Server Preferences page to fetch audio resource management parameters or an audio resource IDs list. Before using this block set GAX Server Preferences.

Note: The OPM block in Composer 8.1.2 supports GAX 8.1.2.

Note: GVP 8.1.6 supports OPM parameters only with lowercase key names - Composer includes a warning to that effect. Please consult your GVP version's documentation for any changes to this behavior.

Note: OPM Complex parameters like 'Schedule', 'Custom List' types are not supported by GVP 8.1.6 and also not supoprted in Composer 8.1.3.

The OPM block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for

Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Assign OPM Data Property

Use this property to assign OPM parameters from GAX to variables.

1. Click opposite **Assign OPM Data** under **Value.** This brings up the … button.
2. Click the … button to bring up the Assign OPM Data dialog box.
3. Click **Add**.
4. Select the variable or click the **Variables** button to add a new variable.

5. Enter a value for the variable or click the button where you can create an expression with Expression Builder.

TLib Block

Use this block in workflows and sub-workflows that will use <session:fetch> method="tlib". The block exposes properties to form a TLib request to set agent status not ready equivalent to TAgentSetNotReady. It also sets srcexpr and <content> element to make it possible to form generic TLib requests. The TLib block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks.

Condition Property

Find this property's details under Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Workflow Blocks.

Result Property

Response returned from the TLib function. You may use this property to assign the collected data to a user-defined variable for further processing.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

Application Property

The T-Server application object in Configuration Server that represents the third party server to be contacted. You can select Configuration Server, enter a literal or enter a value.

Content Property

Use this property to specify input parameters to be passed to the specified TLib function. You can enter as literals and enter property names and values or select a variable.

URL Property

Select the variable that contains the URI for the TLib function that is to be used.

Enable Status Property

Find this property's details under Common Properties for Workflow Blocks.

Web Request Common Block

The Web Request block is used for both routing and voice applications. Use to invoke any supported HTTP web request or REST-style web Service.

- It supports PUT, DELETE, GET and POST methods over HTTPS.
- It is based on common Web Services standards such as XML, SOAP and WSDL instead of proprietary standards that are currently being replaced.

REpresentational State Transfer (REST) is an XML-based protocol for invoking Web Services over HTTP. REST is a lighter version of SOAP, which has evolved into a more complex protocol. REST-style web services offer a less coupled paradigm whereby simpler requests and responses are used. As an example, a simple HTTP request follows the REST methodology. The Web Request block allows the user to query "RESTful" Web services. The supported return formats for the Web Request block are:

- plain text

Note: For workflows, the result will be returned in a JSON string with key name result, e.g., {"result":"This is a plain text result"}

- plain XML
- JSON string (See an issue pertaining to JSON objects in Troubleshooting.)

Important

Composer does not support fetching URLs using HTTPS in Web Request and Web Service blocks.

The Web Request block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks You can also define custom events.

Request Method Property

This property Indicates the method for invoking the web request:

- **get**--Invoked using HTTP Get
- **post**--Invoked using HTTP Post. This option is valid only when the parameters are passed as a namelist (Use Namelist property is set to true). This is generally used when a large amount of data needs to be sent as an input value for a subdialog.
- **put**--Invoked using HTTP Put
- **delete**--Invoked using HTTP Delete

To select a value for the Request Method property:

1. Select the **Request Method** row in the block's property table.
2. In the **Value** field, select get, post, put, or delete from the drop-down list.

Uri Property

The Uri property specifies the http:// page to invoke. To set a URL destination for the Uri property:

1. Select the **Uri** row in the block's property table.
2. In the **Value** field, click the down arrow and select the variable that contains URL.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Authentication Type Property

The Authentication Type property specifies whether to use an anonymous or basic authentication for the web request. To assign a value to the Authentication Type property:

1. Select the Authentication Type row in the block's property table.
2. In the Value field, select anonymous (default) or basic from the drop-down list. With the anonymous type of access, no user name/password is passed to Web service for client authentication in order to get data. If you select the basic type of access, you must supply the Login Name and Password properties.

Encoding Type Property

The Encoding Type property (used for callflows only) indicates the media encoding type of the submitted document. GVP 8.1 supports two encoding types:

- application/x-www-form-urlencoded
- multipart/form-data

To select a value for the Encoding Type property:

1. Select the **Encoding Type** row in the block's property table.
2. In the **Value** field, select one of the following:
    - **application/x-www-form-urlencoded (default)**
    - **application/json**

Input Parameters Property

Use the Input Parameters property to specify a list of required Name/Value pairs to pass as parameters to the http:// page. To specify input parameters:

1. Click the Parameters row in the block's property table.
2. Click the button to open the Parameter Settings dialog box.

Add Button Use the Add button to enter parameter details.

1. Click **Add** to add an entry to Web Request Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the web request.
Out	Output parameters are variables that the web request returns and will be reassigned back to the current callflow/workflow.
InOut	InOut parameters are parameters that act as both input and output.

1. In the Expression drop-down list, select from among the variables shown, type your own expression, or click the button to use Skill Expression Builder.
2. In the Definition field, type a description for this parameter.
3. Click Add again to enter another parameter, or click OK to finish.

Delete Button To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

JSON Content Property

If the HTTP request to be invoked expects JSON content, this property can be used to specify that input. It expects a variable whose content will be sent to the API specified in the HTTP URI property of the block. Set the `Encoding Type` property of the block to `application/json`. In this case, the `Input Parameters` property will not be used.

The variable selected in this property should contain a JavaScript object. The object can be built from a JSON string, or using the ECMAScript block.

For example, if you would like to pass a JSON content to the HTTP URI, using a variable named "content", the variable can be initialized in the following ways:

- If you have a JSON string, you can use the Assign block to assign the following value to "content":

```
JSON.parse('{"abc": "def", "xyz": 3}')
```

- Alternately, you can build a JavaScript object using an ECMAScript block with code like the following:

```
var content = new Object(); content['abc'] = 'def'; content['xyz'] = 3;
```

In both cases, set the `JSON Content` property of the Web Request block to the variable named "content".

Timeout Property

Select the variable containing the number of seconds that the application will wait when fetching the result of the Web Service or the Web Request. If the requested resource does not respond in that time, then a timeout event will occur.

Custom HTTP Headers Property

Use this property to add Custom headers to be sent along with the HTTP request during the runtime execution of the Server Side block.

1. Click the row in the block's property table.
2. Click the ... button to open the Custom HTTP Headers dialog box.
3. Click **Add** to open Configuration Custom HTTP Headers dialog box.
4. Select a Header type.
5. Select **Literal** or **Variable**.
6. Type the literal value or select the variable that contains the value.

Login Name Property

Used when Authentication type = basic. The Login Name property specifies the login name for the invoked web page. To provide a login name for the web request:

1. Select the **Login Name** row in the block's property table.
2. In the **Value** field, type a valid login name.

Password Property

Used when Authentication type = basic. The Password property specifies the password for the invoked web page. To provide a password for the web request:

1. Select the **Password** row in the block's property table.
2. In the **Value** field, type a valid password that corresponds to the login name above.

Result Property

The Result property is the variable used to get back a result from the web request. To select a variable:

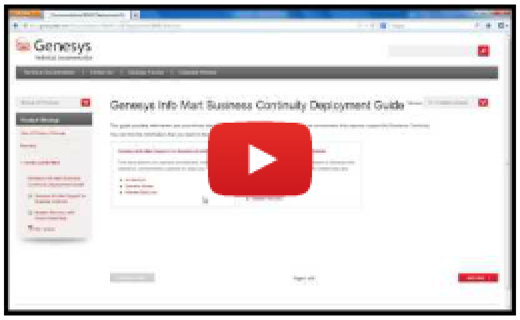
1. Select the **Result** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

Web Service Common Block

Video Tutorial

Below is a video tutorial on using the Web Service block.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```



SOAP-Compliant Web Services

This block can be used to invoke SOAP 1.1 compliant Web Services. It accepts and parses WSDL content for the WebService and collects input parameters based on this WSDL content.

- Uses common Web Services standards such as XML, SOAP and WSDL.
- You can pass parameters (as in subdialogs) and store the return values in variables.
- GET, POST and SOAP over HTTPS are supported.
- Supports SOAP 1.1 and therefore requires a WSDL file to describe endpoints and services. The Web Service block will not work without this WSDL file.
- WSDL-based Web Services are supported with certain limitations. The WSDL is parsed and you are provided the option to select the service name, bindings type, operations, service end point, and mode (GET / POST). The Input and Output parameter list is pulled by default from the WSDL.

Data returned by the Web Service is converted to JSON format and made available in the application. (See an issue pertaining to JSON objects in Troubleshooting.)

Important

SOAP 1.2 is not supported.

Additional Information

For additional information, see:

- Web Service Block and Signed SOAP Requests and Web Service SOAP Message Examples.
- WSDL_SOAP_XSD_WSSE_Support

Web Service Block Security

For Java and .NET Composer projects, the Web Service Block supports secured SOAP communication using XML Digital Signature with a Client Certificate for Java Composer Projects. XML Digital Signature authentication is in compliance with the Second Edition of the XML Signature Syntax and Processing Specification and the OASIS Web Services Security SOAP Messages Security Specification. The Authentication Type property below allows you to select various types of authentication.

Important
Composer does not support fetching URLs using HTTPS in Web Request and Web Service blocks.

Testing the Web Service Block

When working with either a callflow or workflow, the Web Service block provides menu option to test the configured SOAP Web Service using the Web Services Explorer. Right-click the Web Services block and select **Test with Web Services Explorer.** The Web Service block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events. The Web Service block Exceptions dialog box has the following pre-set exceptions:

- Callflows: `error.badfetch and error.com.genesyslab.composer.webservice.badFetch`
- Workflows: `error.session.fetch and error.com.genesyslab.composer.webservice.badFetch`

Service URL Property

The Service URL property specifies the WSDL URL of the Web Service to invoke. To set the Service URL:

1. Select the **Service URL** row in the block's property table.
2. In the **Value** field, type a valid URL.

When you provide the WSDL URL in the Service URL property, Composer will try to access the URL and parse it to populate the drop-down lists for the remaining properties:

- **Available Services**
- **Bindings**
- **Operations**
- **Service End Point**
- **Use Protocol**

Note: When ***upgrading older diagrams to 8.1.1 and higher, it is necessary to clear out the service URL***and specify it again. This is needed in newer versions to re-parse the WSDL obtained from the specified URL and not use the cached information stored in the diagram.

Available Services Property

When Composer accesses the Service URL, the available Web Services will populate the drop-down list of the Available Services property. To select an available service:

1. Click the **Available Services** row in the block's property table.
2. In the **Value** field, select an available Web Service from the drop-down list.

Bindings Property

When Composer accesses the Service URL, the available bindings will populate the drop-down list of the Bindings property. To select a binding:

1. Click the **Bindings** row in the block's property table.
2. In the **Value** field, select an available bindings setting from the drop-down list.

Operations Property

When Composer accesses the Service URL, the available operations will populate the drop-down list of the Operations property. To select an operation:

1. Click the **Operations** row in the block's property table.
2. In the**Value** field, select the desired operation from the drop-down list.

Service End Point Property

When Composer accesses the Service URL, the service end point options will populate the drop-down list of the Service End Point property. To select a service end point:

1. Click the **Service End Point row in the block's property table.**
2. In the **Value** field, select the service end point from the drop-down list.

Service End Point Variable Property

Property to parameterize the Service End Point in the Web Service Block. This property will overwrite the 'Service End Point' properties literal value.

Use Protocol Property

When Composer accesses the Service URL, the protocol options (SOAP and HTTP) will populate the drop-down list of the Use Protocol property. To select a protocol:

1. Click the **Use Protocol** row in the block's property table.
2. In the **Value** field, select SOAP or HTTP from the drop-down list.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Input Parameters Property

Note: The Web Service block won't work with IRD if the Web Service parameters are named double since URS considers it a reserved keyword. The same Web Service block will work fine in the voice application. After you have chosen the available service and operations which you want to invoke, along with bindings, service end point, and protocol, use the Input Parameters property to specify a list of required Name/Value pairs to pass as parameters to the Web Service URL. To specify input parameters:

1. Click the **Parameters** row in the block's property table.
2. Click the ... button to open the Parameter Settings dialog box.

Add Button

Use the **Add** button to enter parameter details.

1. Click **Add** to add an entry to Web Service Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the web service.
Out	Output parameters are variables that the web service returns and will be reassigned back to the current callflow/workflow.
InOut	InOut parameters are parameters that act as both input and output.

4. In the **Expression** drop-down list, select from among the variables shown, type your own expression, or click the button to use Skill Expression Builder.
5. In the **Definition** field, type a description for this parameter.
6. Click **Add** again to enter another parameter, or click OK to finish.

Delete Button

To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

Timeout Property

Select the variable containing the number of seconds that the application will wait when fetching the result of the Web Service or the Web Request. If the requested resource does not respond in that time, then a timeout event will occur.

Custom HTTP Headers Property

Use this property to add Custom headers to be sent along with the HTTP request during the runtime execution of the Server Side block.

1. Click the row in the block's property table.
2. Click the button to open the Custom HTTP Headers dialog box.
3. Click **Add** to open Configure Custom HTTP Headers dialog box.

Note: The list of headers is a standard list defined by the HTTP protocol. You can optionally specify a list of headers. For each header, the name can be selected from the drop down list or keyed in. The value can be specified as literal values or as variable. There is no special format.

1. Select a **Header type**.
2. Select **Literal** or **Variable**.
3. Type the literal value or select the variable that contains the value.

Authentication Type Property

To assign a value to the Authentication Type property:

1. Select the **Authentication Type** row in the block's property table.
2. In the **Value** field, select from the following:

    - **Anonymous**--With the anonymous type of access, no user name/password is passed to Web service for client authentication in order to get data.
    - **HTTP Basic Authentication**--HTTP Protocol level Basic Authentication using Authorization header. If you select the basic type of access, you must supply the Login Name and Password properties.
    - **SOAP Message Level Basic Authentication**--SOAP Message level Basic Authentication for legacy Web Services using <BasicAuth> header.-- Rarely used but for compatibility.
    - **SOAP XML Signature Authentication**--SOAP Message level XML Digital Signature Authentication using Client Certificate.
    - **SOAP Signature with HTTP Basic Authentication**--SOAP Message Level XML Digital Signature Authentication using Client Certificate + HTTP Basic Authentication (for the Web Server level).

Important
Basic HTTP Authentication properties in the Web Service block are validated only during runtime in the server-side pages (ASPX/JSP). For design time, WSDL parsing authentication is not supported. You can copy the WSDL file to the **Include** folder within the required **Composer Project** folder and specify `include/<filename.wsdl>` in the **Service URL** property to parse the WSDL file and configure the block.

Login Name Property

The Login Name property specifies the login name for the invoked web page. To provide a login name for the web request:

1. Select the **Login Name** row in the block's property table.
2. In the **Value** field, type a valid login name.

Password Property

The Password property specifies the password for the invoked web page. To provide a password for the web request:

1. Select the **Password** row in the block's property table.
2. In the **Value** field, type a valid password that corresponds to the login name above.

Certificate Store Name Property

Use this property to specify the name of the Windows Certificate Store. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate Store Name** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Certificate Alias Property

Use this property to specify the Client Certificate Name. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate Alias** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Certificate or Key Store Location Property

Use this property to specify the location of the Certificate Store or Key Store. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate or Key Store Location** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Key Algorithm Property

Select DSA (default) or RSA to specify the Key Algorithm to sign the SOAP Digital Signature. See Web Service Block and Signed SOAP Requests. Use this property to specify the Key Store Password. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Key Algorithm** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Key Store Password Property

To select a variable:

1. Select the **Key Store Password** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

Private Key Property

Use this property to specify private key of the Client Certificate. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Private Key** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Private Key Password Property

Use this property to specify the private key password. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Private Key Password** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Custom Prefix Property

Use this property to set custom namespace to the generated SOAP message tags. If this property is set it will overwrite the default / WSDL namespace prefix.

Note: To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.

Add Namespace Prefix Property

Use this property to add Namespace prefix to the generated SOAP message. By default Composer Web Service client doesn't generate namespace prefixes.

1. None - Do not add any namespace prefix to the SOAP:Body elements.
2. Method Name Tag Only - Add namespace prefix only to the Method Name tag (Operational name tag).

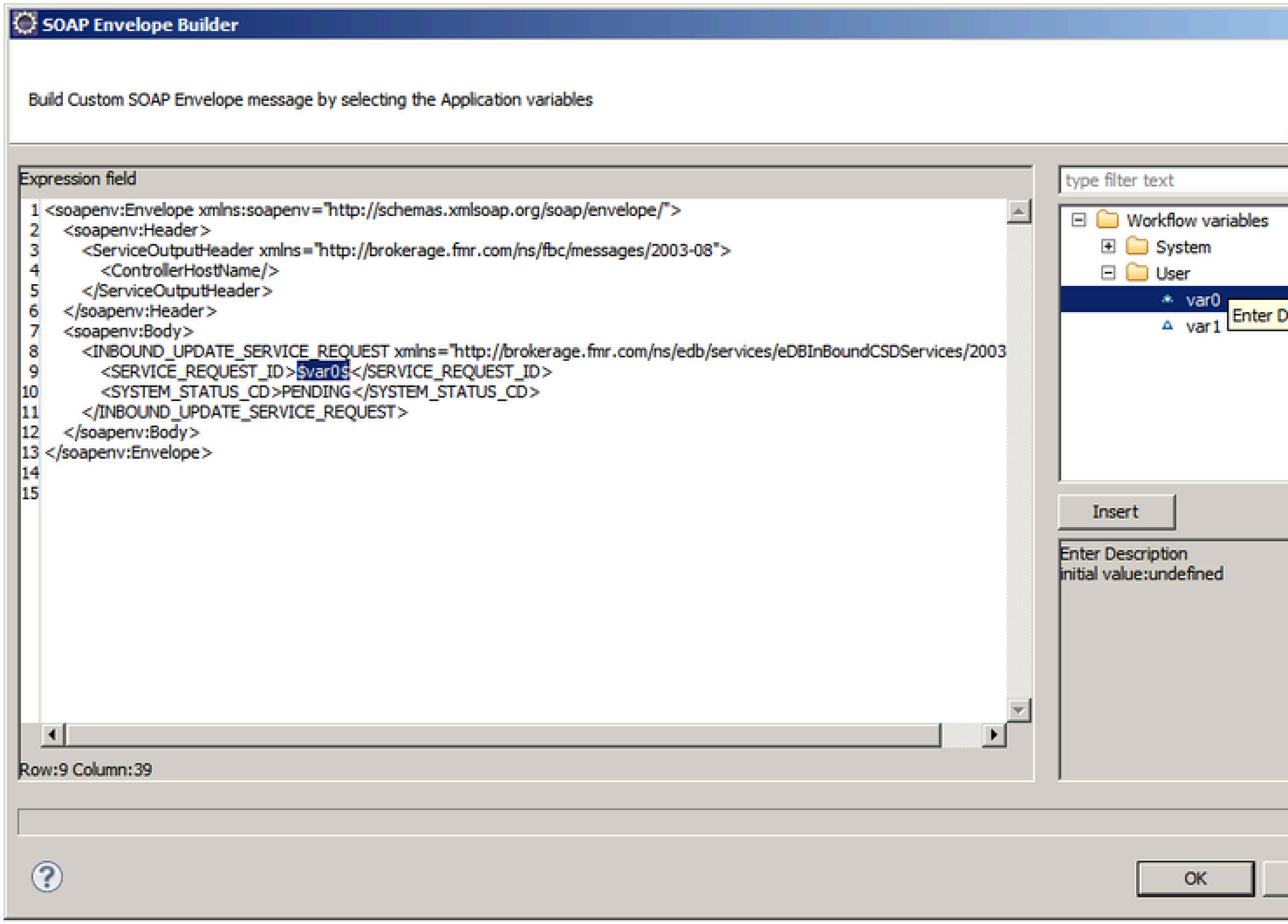
3. Method Name Tag and Child Tags - Add namespace prefix to all the tags in the SOAP message.

Note: To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.

Custom SOAP Envelope Property

Use this property to set Custom SOAP Envelope messages. If this property is set, the Composer Web Service run-time client will use this message to get a Web Service response.

1. Click the **Custom SOAP Envelope** property under the **SOAP Message Generation** category. The Custom SOAP Envelope dialog is displayed.
2. Add the custom SOAP Envelope message (the message can be generated using any Web Service client tool).



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <ServiceOutputHeader xmlns="http://brokerage.fmr.com/ns/fbc/messages/2003-08">
      <ControllerHostName/>
    </ServiceOutputHeader>
  </soapenv:Header>
  <soapenv:Body>
    <INBOUND_UPDATE_SERVICE_REQUEST xmlns="http://brokerage.fmr.com/ns/edb/services/eDBInBoundCSDServices/2003
      <SERVICE_REQUEST_ID>$var0$</SERVICE_REQUEST_ID>
      <SYSTEM_STATUS_CD>PENDING</SYSTEM_STATUS_CD>
    </INBOUND_UPDATE_SERVICE_REQUEST>
  </soapenv:Body>
</soapenv:Envelope>
```

Row:9 Column:39

Enter Description
initial value:undefined

Custom SOAP Envelope Dialog

The custom message is sent to the Web Service URL at run-time. Diagram application variables can be used to form dynamic contents. Variables can be used in the custom message with a **$<Variable_Name>$** notation.

Note:

1. To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.
2. This property is supported for both Java Composer Projects and .NET Composer Projects.
3. Callflow-Root document variables and Workflow-Project variables are not supported in this property.

Output Result Property

When the Map Output Values to Variables property above is set to true, the Output Result property maps the Web Service response keys to AppState variables. If Map Output Values to Variables is set to false, the entire Web Service response will be assigned to a variable. The Output Result property is the variable used to get back the invoked Web Service result. To select a variable:

1. Select the **Output Result** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

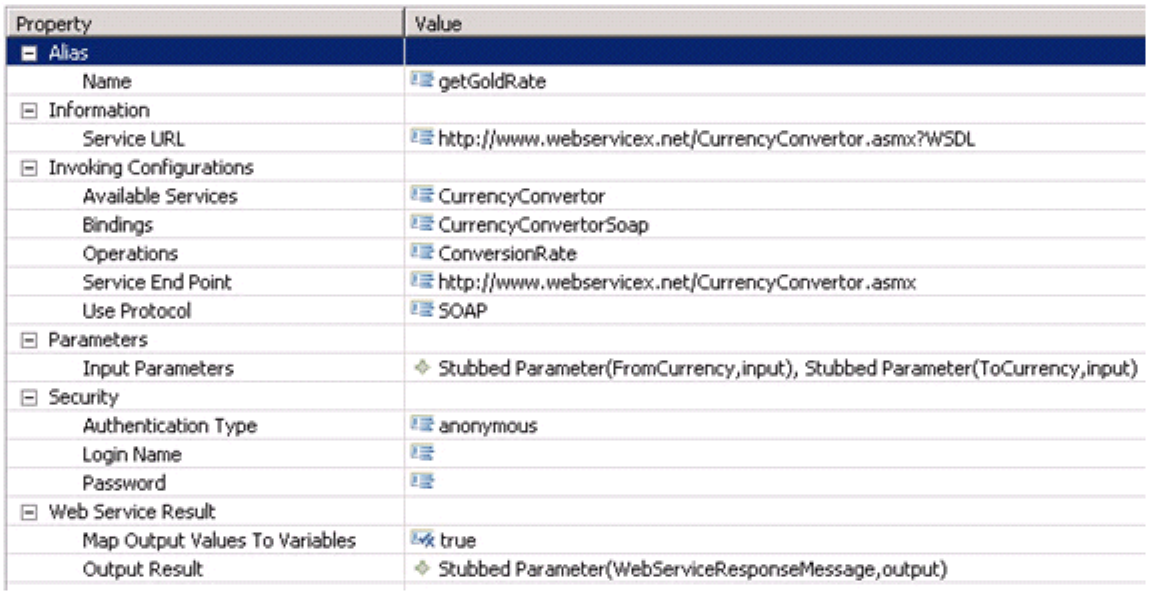
Map Output Values to Variables Property

The Map Output Values to Variables property indicates whether or not to map the Web Service response keys to `AppState` variables. To select a value for the Map Output Values to Variables property:

1. Select the **Map Output Values to Variables** row in the block's property table.
2. In the **Value** field, select true or false from the drop-down list.

Example Block Properties

Example properties for a Web Service block are shown below.



Property	Value
⊟ Alias	
Name	getGoldRate
⊟ Information	
Service URL	http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
⊟ Invoking Configurations	
Available Services	CurrencyConvertor
Bindings	CurrencyConvertorSoap
Operations	ConversionRate
Service End Point	http://www.webservicex.net/CurrencyConvertor.asmx
Use Protocol	SOAP
⊟ Parameters	
Input Parameters	Stubbed Parameter(FromCurrency,input), Stubbed Parameter(ToCurrency,input)
⊟ Security	
Authentication Type	anonymous
Login Name	
Password	
⊟ Web Service Result	
Map Output Values To Variables	true
Output Result	Stubbed Parameter(WebServiceResponseMessage,output)

ExampleWebService.gif

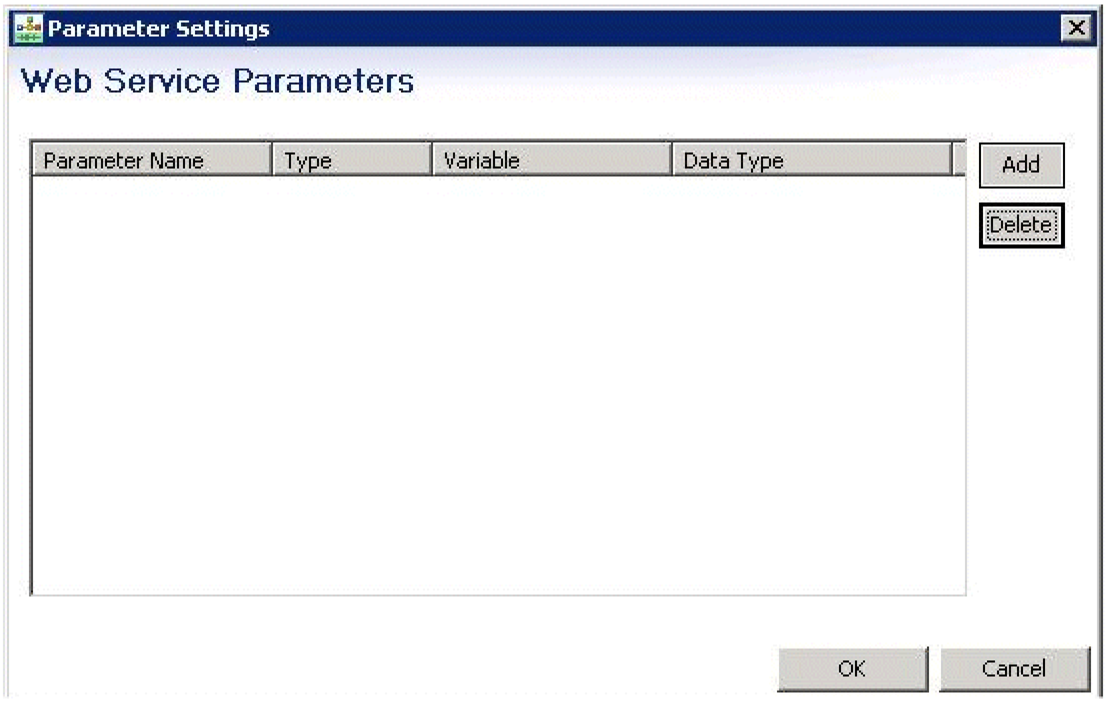
Web Services Description Language (WSDL) Support

Composer supports WSDL definitions conforming to the version WSDL 1.1 schema.

Errors in WSDL Parsing

The following Composer symptom may indicate errors in WSDL parsing:

- If the WSDL definition contains any of the unsupported types and elements, Composer may not be able to parse the WSDL correctly to identify the input and output parameters of the Web Service.
- If the Composer WSDL parser is unable to properly parse the WSDL definition for the Web Service, the input and output parameters fields in the Web Service Parameters dialog box might be empty, with no pre-configured parameters as shown below.



Workarounds

Currently, the following workarounds are available to change the schemas to work with Composer:

- Use qualified elementFormDefault (elementFormDefault="qualified") and define types with fully qualified namespace definitions.
- Define all wsdl types in one schema.
- Replace reference attributes with the actual types being referenced.
- Use the Add/Delete buttons to add or remove any parameters that may not have been automatically displayed. The SOAP request that will be generated at application runtime will take these changes into account.

Note

Composer Web Service Block generated SOAP messages does not have prefix in the SOAP elements. Web Services created using Metro / JAX-WS framework may return Null Pointer Exception or Unexpected Result due to the prefix limitation. Updating the JAX-WS API's / GlassFish server / Metro WS Framework to latest versions may help.

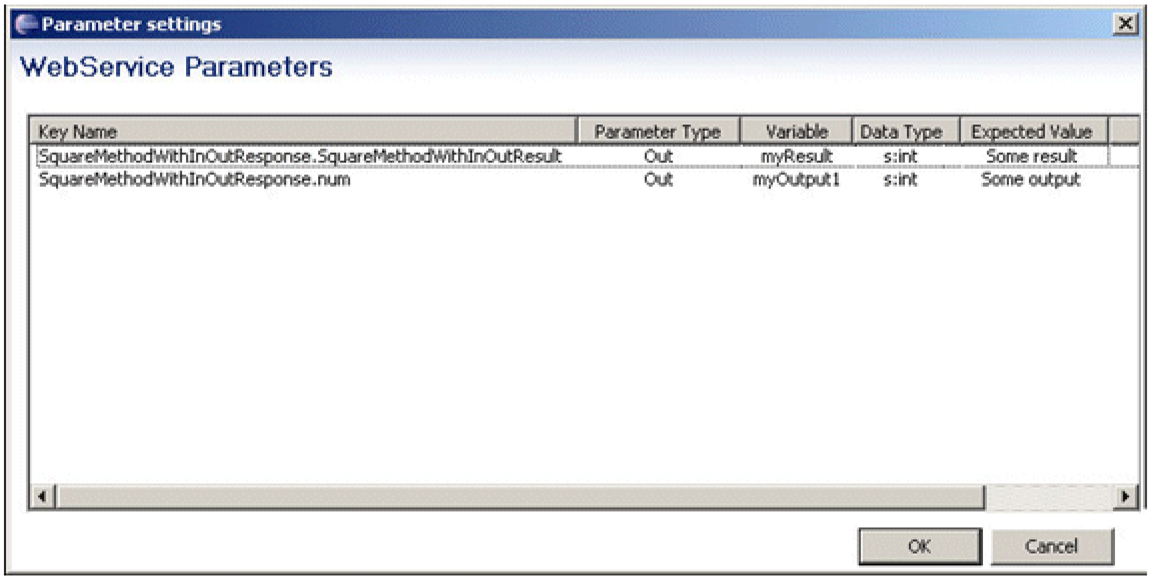
Web Service Stubbing

Composer's Web Services stubbing feature allows you to work with Web Services in off-line mode when you do not have access to the Web Service itself or if the Web Service is under development. This feature is intended to be used in a test environment. It is not intended for a production environment unless there is need to remove an active Web Service from a callflow for debugging purposes.

Using Web Services Stubbing

To use Web Services stubbing:

1. To enable stubbing, add the variable COMPOSER_WSSTUBBING to your Entry block and set its value to 1 indicating stubbing is turned on (0 = stubbing is turned off). In Composer 8.0.2 and later, this variable is present by default in the Variables Setting dialog box, which opens from the Entry object.
2. Create the Web Service block.
3. Place the Web Service Description Language (WSDL) file in your Project. The assumption is that the WSDL file for the Web Service is available at all times.
4. For the Service URL property, use a local URL to the WSDL file. When the Web Service is ready to be used, change this local URL to the correct URL.)
5. To specify the expected output value for the Web Service result as well as any output parameters, use the Output Result property of the Web Service block. An example is shown below.



Key Name	Parameter Type	Variable	Data Type	Expected Value
SquareMethodWithInOutResponse.SquareMethodWithInOutResult	Out	myResult	s:int	Some result
SquareMethodWithInOutResponse.num	Out	myOutput1	s:int	Some output

Using the above examples:

- If stubbing is on, the myResult variable will be assigned the value Some result and myOutput1 will be assigned the value of Some output.

- If stubbing is off, the value returned by the Web Service will be stored in these two variables.

Limitation

Web Service stubbing currently does not support auto-synchronization of output parameters in case of Web Services with complex return types.

Web Service SOAP Messages

Use the examples below when configuring the Web Service block.

SOAP Message Level Basic Authentication

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
              <SOAP-ENV:Header>
                           <h:BasicAuth xmlns:h="http://soap-authentication.org/basic/2001/
10/" mustUnderstand="1">
                                       <Name>UserName</Name>
                                       <Password>Pass</Password>
                          </h:BasicAuth>
              </SOAP-ENV:Header>
              <SOAP-ENV:Body>
                            <p:getNumber xmlns:p="http://webservice.com"/>
              </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Signed Using Client Digital Certificate (DSA Key Algorithm)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" SOAP-ENV:mustUnderstand="1">
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-2">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="SOAP-
ENV"/>
</ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<ds:Reference URI="#id-1">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"><ec:InclusiveNamespaces
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"  PrefixList=""/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#sha1"/><ds:DigestValue>yMmgdHRevOnFPGtnSZx4JV9hiuI=</ds:DigestValue></ds:Reference></ds:SignedInfo><ds:
Id="KI-8D7856F18A7AB8CF5613098009924472">
<wsse:SecurityTokenReference wsu:Id="STR-8D7856F18A7AB8CF5613098009924473">
<wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-
profile-1.0#X509v3">MIICwDCCAn2gAwIBAgIETfBxZzALBgcqhkjOOAQDBQAwQzELMAkGA1UEBhMCVVMxDDAKBgNVBAoTA1N1bjERMA8GA1U
yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/
IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/
```

```
mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTD
5Ykhco6lMBRRncJwGuWB/mFPhaX8Odfj8NMEih1+ICIjhVwGk1p6P3Gu2Dm+45TYJCxBktdOlU0uy/
Uj8E61NZSaeQL9WA4gGz5Hb5uMAsGByqGSM44BAMFAAMwADAtAhQqfbMb9hd1vpBAAJntCDSOY5uP2AIVAJGy1E7Zx4268n3fD34gLcpkZoKc</
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="id-1">
<p:methodName xmlns:p="http://example.com"><key>value</key></p:methodName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Signed Using Client Digital Certificate (RSA Key Algorithm)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" SOAP-ENV:mustUnderstand="1">
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-2">
<ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="SOAP-
ENV"/>
</ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#id-1">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"><ec:InclusiveNamespaces
xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList=""/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>yMmgdHRevOnFPGtnSZx4JV9hiuI=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>MX9c9C5Tpfvp32e2pPjkCv4ycZhcuZVMFHo8DlGKWi331fnG3oqXLg==</ds:SignatureValue>
<ds:KeyInfo Id="KI-8D7856F18A7AB8CF5613098009924472">
<wsse:SecurityTokenReference wsu:Id="STR-8D7856F18A7AB8CF5613098009924473">
<wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-
profile-1.0#X509v3">MIICwDCCAn2gAwIBAgIETfBxZzALBgcqhkjOOAQDBQAwQzELMAkGA1UEBhMCVVMxDDAKBgNVBAoTA1N1bjERMA8GA1U
yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/
IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/
mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTD
5Ykhco6lMBRRncJwGuWB/mFPhaX8Odfj8NMEih1+ICIjhVwGk1p6P3Gu2Dm+45TYJCxBktdOlU0uy/
Uj8E61NZSaeQL9WA4gGz5Hb5uMAsGByqGSM44BAMFAAMwADAtAhQqfbMb9hd1vpBAAJntCDSOY5uP2AIVAJGy1E7Zx4268n3fD34gLcpkZoKc</
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
```

```
utility-1.0.xsd" wsu:Id="id-1">
<p:methodName xmlns:p="http://example.com"><key>value</key></p:methodName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Signed SOAP Requests

The Web Service block enables Composer applications to invoke Web Services, which require message-level authentication. The message level security support provided by the Web Service block is limited to one-way signed SOAP requests from the Composer application to the Web Service. Web Services can then verify that the request received from a Composer application includes a valid certificate.

Prerequisites

The prerequisites are:

- Web Service is able to verify only the signature (Timestamp, UsernameToken and Encryption are not supported).
- Web Service sends an unsigned response, i.e., Web Service is not configured to process outgoing response (only InflowSecurity is configured).
- X.509 certificate for the client is available and is trusted by the Web Service. Certificates can be purchased from a certificate authority or can be generated (for testing) using tools, such as OpenSSL.
- Certificates should be based on one of the supported encryption algorithms, RSA or DSA.
- Certificate Stores:
- For Java projects, certificates and keys should be available in a Java Keystore (*.jks file). OpenSSL and Keytool (available in JDK 1.6) can be used to create and import certificates.
- For .NET projects, certificates and keys should be available in the Windows Certificate Store. OpenSSL can be used to create certificates and Certificates (snap-in in Microsoft Management Console) can be used to import certificates.
- For .NET projects, WSE 3.0 (runtime) should be installed on the machine running Composer.

Enabling Signing of SOAP Messages

To enable signing of SOAP messages, set the Authentication Type property in the Security section to one of the following values

- SOAPDigitalSignatureAuthentication -- for signing messages when not using HTTP Basic authentication.
- SOAPSignatureWithHTTPBasicAuthentication -- for signing messages when used along with HTTP Basic Authentication (Security Basic Authentication Credentials section is specified)

Once enabled to sign the request, the application will need information regarding the public key (certificate) and private key (key) as below:

- Certificate Store Name (.NET only) -- Windows Store Name containing the client certificate and private key. Value should be one of the following predefined Windows Certificate Stores or the name of a custom Store in which the certificate and key are stored. Note that this Store should contain the client certificate (should include the private key as well).

- AddressBook -- The X.509 Certificate Store for other users.
- My -- The X.509 Certificate Store for personal certificates.
- TrustedPeople -- The X.509 Certificate Store for directly trusted people and resources.
- Certificate Alias -- Alias that identifies the certificate and key in the Store. For .NET projects, this refers to the subject of the certificate, e.g., CN=ComposerCertificate.
- Certificate or Key Store Location -- Path to the Certificate Store location containing the certificate and private key. In .NET, the value should be set to one of the following:
- StoreLocation.LocalMachine (default when value is not one of these)
- StoreLocation.CurrentUser
- Key Algorithm -- Algorithm to be used for encryption. This is the same as the algorithm that was specified when the key was generated; usually received from the certificate authority issuing the certificate.
- Key Store Password (Java only) -- Java Key Store password for the key store specified as the key store location.
- Private Key Alias (Java only) -- Alias by which the private key is identified in the key store.
- Private Key Password (Java only) - Password to access the private key to be used when signing a message. For .NET projects, it is expected that the password be stored as part of the settings for the certificate.

At run time, the Composer application will create a SOAP message and then sign it using its private key. The signed message will include an encrypted signature in the SOAP header and the SOAP request as the body. This signed message is sent to the Web Service for processing. Web Service will decrypt the signature using the client certificate (public key previously imported into the Web Service certificate store) and hence authenticating that the source of the request is valid.

Signature Validation Failure Causes

Signature Validation by the Web Service may fail for the following reasons:

- Syntax of request (signature) doesn't conform to the policy enforced by the Web Service. Example: Timestamp is required by the Web Service but was not included in the request because Composer doesn't support Timestamp policy.
- Validation of signature failed. Example: Web Service uses RSA key, but the request was signed using DSA key.
- Application validation policy rejects the request. Example: Signature created by an untrusted key.

Once signature validation is successful, the Web Service will process the request and then send the unsigned response back to the Composer application. Composer processes the response without signature validation. The above will ensure that Web Services will process requests only from legitimate clients, the Composer application being one of them.

Connection and Read Timeout

By default, the Web Service and Web Request blocks use 20 seconds each for the connection timeout and read timeout. The following steps describe how to configure the timeouts:

1. Create a new folder called `WEB-INF` inside the Composer Project.
2. Create a property file named `composer.properties` inside the `WEB-INF` folder.
3. Add the following properties (case-sensitive) to composer.properties with the timeout values:
    - `web.connectionTimeout=40000`
    - `web.readTimeout=40000`

The value specified should be in milliseconds. The connectionTimeout property is a specified timeout value, in milliseconds, to be used when opening a communications link to the resource referenced by the URL. The readTimeout property is a nonzero integer value, in milliseconds, to be used when reading from an input stream when a connection is established to a URL resource.

Server-Side Troubleshooting

The table below lists Server Side block troubleshooting situations and steps to remedy.

Situation	Block	Steps to Troubleshoot
I entered the Service URL but getting a pop-up with Check the Web Service URL	Web Service Block	Verify that the WSDL definition is accessible in a web browser. Check the Composer logs for possible errors in fetching the WSDL. Location: <workspace>\.metadata Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
I entered the Service URL and I can choose the SOAP operations, but the parameters do not show up in the dialogs	Web Service Block	Verify that the WSDL definition is accessible in a web browser. Check the Composer logs for possible errors in fetching the WSDL. More details can be found in the logs in the following location: <workspace>\.metadata\.plugins\ com.genesyslab.studio.model folder Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
Using the Web Services Explorer utility	Web Service Block	The Web Services Explorer is a JSP Web application hosted on the Apache Tomcat servlet engine contained within Eclipse. The Web Services Explorer is provided with Composer and allows you to explore, import, and test WSDL documents. Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
Errors during runtime	Web Service Block Web Request Block Backend Block	Check the Composer logs for possible errors in fetching the WSDL. Check the backend logs. For ASP.NET projects, check the IIS logs For Java Projects, check the Tomcat standard logs. Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
I copied my callflow/workflow from one project to another but my Backend block does not work	Backend Block	Check that any custom backend libraries or applications have also been copied to the new project.

Backend block		

Logs:

- Java Composer Projects Server Side Backend logging can be controlled using the `log4j.xml` file present in the `$ComposerInstalledLocation\tomcat\lib` folder.
- For DotNetComposer Projects web.config file can be used to control the Server Side logging.
- Java Composer Projects exported as WAR files will have the `log4j.xml` bundled inside the `WEB-INF\lib` folder. If the `log4j.xml` configuration format is not working, you can add a `log4j.properties` in the `tomcat/webapps/<application name>/WEB-INF/classes` folder.

Notes:

- Service URL has to end with wsdl or WSDL
- Cannot use - or other reserved characters in the Entry block for a variable value. Enter the value directly in the input parameters dialog by typing the value in the Expression column as a string; example: 'atm near 37.771008, -122.41175'

Outbound Common Blocks

The Outbound blocks support Genesys Outbound Contact, an automated product for creating, modifying, running, and reporting on outbound campaigns for proactive customer contact. Outbound Contact Solution (OCS) provides automated dialing and call-progress detection, so that an Agent is required only when a customer is connected. Composer supplies the following Outbound blocks:

Block Name	Block Description
Add Record	Automates building of Calling Lists by adding a new record to a specified Calling List.
Cancel Record	Cancels a customer record in a calling list.
Do Not Call	Adds a contact record, such as a phone number or an e-mail address, to a specified Do Not Call List and marks the corresponding record as Do Not Call.
Record Processed	Marks a record as requiring no further handling.
Reschedule Record	Reschedules a customer interaction from the specified Calling List.
Update Record	Updates a Calling List record that you specify via a RecordHandle parameter.

OCS Variables

The Outbound blocks use OCS variables for SCXML applications and OCS variables for VXML applications, present in the Entry blocks of their respective diagrams. These variables are prefixed by "OCS_" and are added to the Entry by default.

Using the Outbound Blocks

Outbound blocks are specifically designed to be used in callflows/workflows that are configured to work with Outbound records, the essential element of which is communication between Universal Routing Server (URS) and Interaction Server, and between Interaction Server and OCS. For additional information, see:

- *Outbound Contact 8.1 Deployment Guide*
- *Outbound Contact 8.1 Reference Manual*

Add Record Block

Use this block to automate building of Calling Lists by adding a new record to a specified Calling List. For example, you can use the Add Record block to automatically develop a Calling List, such as one to follow up on inbound calls that were abandoned during traffic peaks. You can then configure a routing workflow to detect abandoned calls and add records to the Calling List with the parameters of the incoming interactions. The Calling List can then be used by an outbound campaign that dials out to these customers during off-peak hours and has the Agent apologize and follow up. Also see:

- OCS Application Variables
- Using the Outbound Blocks

This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Contact Info Property

Click the down arrow and select the variable that contains the contact telephone number (home, work, cell), FAX number, or e-mail address.

Contact Info Type Property

Click the down arrow and select a Contact Information Type: **No Contact Type**, **Home Phone**, **Direct Business Phone**, **Business with Extension**, **Mobile**, **Vacation Phone**, **Pager**, **Modem**, **Voice Mail**, **Pin Pager**, **Email Address**, **Instant Messaging**.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Record Status Property

Click the down arrow and select a record status, such as **Ready**, **Retrieved**, **Updated**, **Stale**, **Cancelled**, **Agent Error**, **Missed Callback**.

Record Type Property

Click down arrow and select the Type of record, such as **No Record Type**, **Unknown**, **General**, **Campaign Rescheduled**, **Personal Rescheduled**, **Personal Callback**, **Campaign CallBack**, **No Call**.

Call Time Property

This property specifies the time when record was called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the call date and time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the call timetime.
4. Click **OK** to close the dialog box.

Call Time From Property

This property specifies the time frame when a record can be called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time From dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the time.
4. Click **OK** to close the dialog box.

Call Time Until Property

This property specifies the time frame when a record can be called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time Until dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the time.

4. Click **OK** to close the dialog box.

Scheduled Date and Time Property

This property specifies the date/time at which scheduled call should be dialed.

1. Click under **Value** to display the button.
2. Click the button to open the Scheduled Data and Time dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the date and time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the date and time.
4. Click **OK** to close the dialog box.

Time Zone Property

This property specifies the name of a Time Zone, associated with the customer record and configured in Configuration Server.

1. Click under **Value** to display the button.
2. Click the button to open the Time Zone dialog box.
3. From the **Type** dropdown, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Time Zone from the **Value** field.
    - Select **Literal** from the dropdown menu and then enter the Time Zone in the **Value** field.
    - Select **Variable** from the dropdown menu and then select the variable that contains the Time Zone from the **Value** field.
4. Click **OK** to close the dialog box.

Attempts Property

Click the down arrow and select the variable that specifies the maximum number of attempts to dial the record in the Calling List during one Campaign.

Calling List Property

This property specifies the name of a Calling List, which is configured in Configuration Server.

1. Click under **Value** to display the button.
2. Click the button to open the Calling List dialog box.
3. From the **Type** dropdown, do one of the following:
   - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Calling List from the **Value** field.
   - Select **Literal** from the dropdown menu and then enter the Calling List in the **Value** field.
   - Select **Variable** from the dropdown menu and then select the variable that contains the Calling List from the **Value** field.
4. Click **OK** to close the dialog box.

Call Result Property

Click the down arrow and result code as defined in a Configuration Manager Enumeration table, such as Abandoned, Agent Callback Error, All Agents Busy, Answer, Answering Machine Detected, Bridged, Busy, Call Drop Error, and so on.

Campaign Property

This property specifies the name of an Outbound Campaign associated with the Calling List, which is configured in Configuration Server.

1. Click under **Value** to display the button.
2. Click the button to open the Campaign dialog box.
3. From the **Type** dropdown, do one of the following:
   - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Campaign from the **Value** field.
   - Select **Literal** from the dropdown menu and then enter the Campaign in the **Value** field.
   - Select **Variable** from the dropdown menu and then select the variable that contains the Campaign from the **Value** field.
4. Click **OK** to close the dialog box.

Chain ID Property

Click the down arrow and select the variable that contains a unique chain identifier (optional). If missing, it is assumed that a record forms a new chain.

Chain N Property

Click the down arrow and select the variable that contains a unique number in a chain (optional). If missing, the next available number is assigned.

OC Server Property

This property identifies the Outbound Contact Server that will interact with the block. You can specify a different OCS application for a specific block. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key-value pairs for user data attached to the interaction.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
    - Select the **Value is numeric box if applicable.**
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Cancel Record Block

Use this block to cancel a customer record in a calling list. You can identify the customer record to cancel by using the Record Handle, Contact Info, or Customer ID property (one of these must be specified). If you specify more than one of these properties, the identifiers are prioritized as follow: Record Handle (highest), Contact Info, Customer ID (lowest). This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Contact Info Property

Select the variable that contains contact information, such as telephone number (home, work, cell), FAX number, or e-mail address. This parameter can be used for Inbound calls to reference the customer record when Record Handle is not available.

Customer ID Property

Select the variable that identifies the customer when a user-defined field is present in the calling list as described in the *Outbound Contact 8.1 Deployment Guide*. You can use for Inbound calls to reference the customer record when Record Handle is not available.

OC Server Property

This property identifies the Outbound Contact Server processing this Calling List. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

Record Handle Property

Select the variable that identifies the customer using the Record ID assigned by Outbound Contact Server if available. Either Record Handle, Contact Info or Customer ID must be specified.

Tenant Property

Select the variable that identifies the tenant associated with the Calling List.

Update Record Chain Property

Select False to indicate if only the customer record should be cancelled. Select True if all records chained to the customer record should be canceled.

Do Not Call Block

Use this block to add a contact record, such as a phone number or an e-mail address, to a specified Do Not Call List and marks the corresponding record as Do Not Call. **Note:** Do not use the Do Not Call and Record Processed blocks to finalize Outbound record processing. You cannot use other Outbound blocks to process records with the same Record Handle after using Processed or Do Not Call in a workflow. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Contact Info Property

Select the variable that contains contact information, such as telephone number (home, work, cell), FAX number, or e-mail address. This parameter can be used for Inbound calls to reference the customer record when Record Handle is not available.

Customer ID Property

Select the variable that identifies the customer when a user-defined field is present in the Calling List as described in the *Outbound Contact 8.1 Deployment Guide*. You can use for Inbound calls to reference the customer record when Record Handle is not available.

OC Server Property

This property identifies the Outbound Contact Server processing this Calling List. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

Record Handle Property

Select the variable that identifies the customer using the Record Handle if available. Either Record Handle, Contact Info or Customer ID must be specified.

Tenant Property

Select the variable that identifies the tenant associated with the Calling List.

Update Record Chain Property

Select False to indicate if only the customer record should be cancelled. Select True if all records chained to the customer record should be canceled.

Record Processed Block

Use the Record Processed block to mark a record as requiring no further handling. When an Agent finishes processing a Calling List record, Genesys Desktop sends a RecordProcessed event to indicate that the record is processed and Outbound Contact Server updates the record accordingly. Use the Record Processed block in a workflow to have URS request (through Interaction Server) that Outbound Contact Server finish processing a record created as a result of a customer inquiry. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. **Note:** Do not use the Do Not Call and Record Processed blocks to finalize Outbound record processing. You cannot use other Outbound blocks to process records with the same Record Handle after using Processed or Do Not Call in workflow. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server that will interact with the block. You can specify a different OCS application for a specific block. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key-value pairs for user data attached to the interaction.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
    - Select the **Value is numeric box if applicable.**
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Reschedule Record Block

Use this block to Reschedule a customer interaction from the specified Calling List. A record is typically rescheduled during a call when a customer requests a callback at a certain time. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server (OCS) application that the block will interact with. It allows you to specify a different OCS application for a specific block. By default, the OCS_Record_URI application variable is used.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.
   - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Outbound Contact Server object from the **Value** field.
   - You can also select **Literal** and enter the name of the server in the **Value** field.
   - You can also select **Variable** and select the variable containing the name from the **Value** field.

If the datasource is Configuration Server, Composer reads the OCS host, listening port, and connection protocol from Configuration Server. If the datasource is Literal/Variable, use the format `[http|https]://<host>:<port>`.

Scheduled Date and Time Property

Specify the date/time at which scheduled call should be dialed.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the Scheduled Date and Time dialog box.
3. The next step depends on whether you are connected to Configuration Server.
4. Do one of the following.
   - Select **Literal** and select the date and time from the **Value** field.

- Select **Variable** and select the name of the variable containing the date and time.
- Select **Delay** and select an amount of time to delay from the **Value** field.

Update Record Block

Use this block to update a Calling List record that you specify via a RecordHandle parameter. For example, in Predictive dialing mode, this request can be used to overwrite the call result detected by call progress detection when needed. Or you can overwrite an answer call result with the wrong party call result. **Note:** When this block is executed in a workflow, it results in an External Service Request (through Interaction Server) to Outbound Contact Server. Since the request goes through Interaction Server, you must have the Genesys Multimedia product installed and an Open Media component to handle External Service processing. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server (OCS) application that the block will interact with. It allows you to specify a different OCS application for a specific block. By default, the OCS_Record_URI application variable is used.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.

    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Outbound Contact Server object from the **Value** field.
    - You can also select **Literal** and enter the name of the server in the **Value** field.
    - You can also select **Variable** and select the variable containing the name from the **Value** field.

If the datasource is Configuration Server, Composer reads the OCS host, listening port, and connection protocol from Configuration Server. If the datasource is Literal/Variable, use the format `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key value pairs for user data attached to the interaction.

1. Click under **Value** to display the button.
2. Click the button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.

- If you select **Literal**, enter the name of the key in the second field.
- If you select **Variable**, select the name of the variable from the second field.
- Select the **Value is numeric box if applicable.**

6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Using Voice Blocks

This section describes:

- Common Properties for Callflow Blocks
- Working with Grammar Builder
- Working with CTI Applications
- Working with Prompts
- Working with Database Blocks
- User Data Voice
- Connection Pooling

Common Properties for Callflow Blocks

The following properties are common to multiple blocks. Their descriptions are placed here to minimize duplication of content:

Name Property

The Name property is present in all blocks in Composer. The Name property is the first property for all blocks. Use the Value field beside in the Name property row of the block's property table to name the block.

- Block names should conform to ECMAScript and VoiceXML identifier naming conventions.
- There is no maximum limit to the number of characters allowed.
- Names must consist only of numbers, letters, and underscore characters.
- Names must begin with a letter or underscore.
- Except for the Entry and Exit blocks, you should give all blocks a descriptive name. For example, if an Input block asks the caller to input an account number, then the name of the block could be Input_Account_Number.
- The name of the block is used as the Name of the VXML <form> tag that gets generated for that block.

To provide a name for a block:

1. Select the Name row in the block's property table.
2. In the Value field, type a block name that conforms to the restrictions above.

Block Notes Property

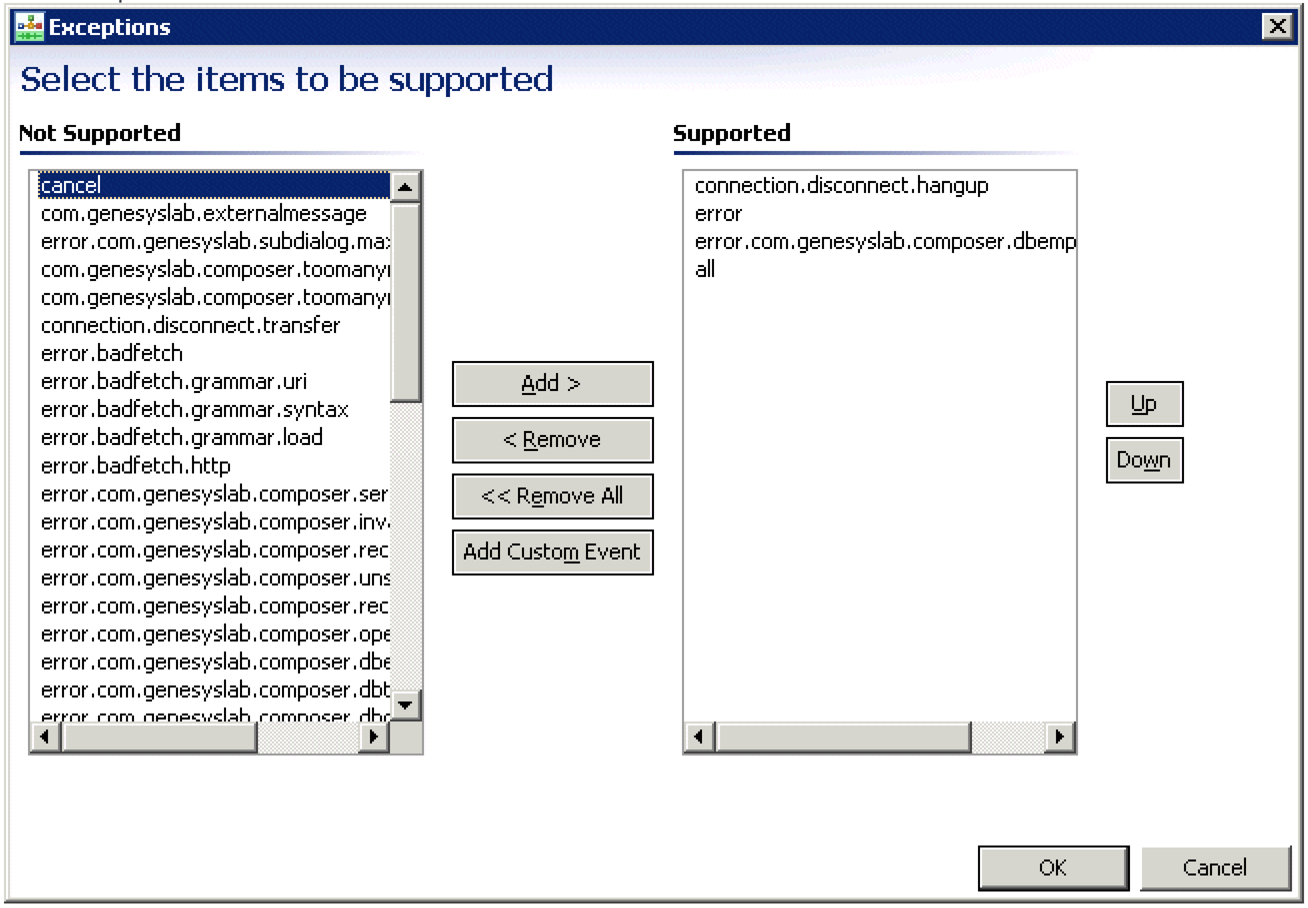
Can be used for both callflow and workflow blocks to add comments. When migrating strategies created with Interaction Routing Designer into Composer as workflow diagrams, this property is the equivalent of the IRD Object Comments feature. For the IRD objects migrated into Composer blocks, Composer supplies the note text by combining the IRD equivalent object name plus the original comments from the IRD object.

Exceptions Property

Use this property to define which exception events the block is designed to handle. These are VoiceXML events that are either thrown by the interpreter, or generated in response to a caller action. Note: A catch handler called all has been added to catch all exception events. To handle (support) a specific event:

1. Click the Exceptions row in the block's property table.
2. Click the button to open the Exceptions dialog box.
3. From the list of events on the Not Supported pane, select the event that you want to handle.
4. Click the Add > button to move the event to the Supported pane.

An example is shown below.



Not Supported:

```
cancel
com.genesyslab.externalmessage
error.com.genesyslab.subdialog.ma:
com.genesyslab.composer.toomanyi
com.genesyslab.composer.toomanyi
connection.disconnect.transfer
error.badfetch
error.badfetch.grammar.uri
error.badfetch.grammar.syntax
error.badfetch.grammar.load
error.badfetch.http
error.com.genesyslab.composer.ser
error.com.genesyslab.composer.inv
error.com.genesyslab.composer.rec
error.com.genesyslab.composer.uns
error.com.genesyslab.composer.rec
error.com.genesyslab.composer.ope
error.com.genesyslab.composer.dbe
error.com.genesyslab.composer.dbt
```

Supported:

```
connection.disconnect.hangup
error
error.com.genesyslab.composer.dbemp
all
```

To explicitly not handle (not support) a specific event marked as supported:

1. Click the Exceptions row in the block's property table.
2. Click the button to open the Exceptions dialog box.
3. From the list of events on the Supported pane, select the event that you do not want to handle.
4. Click the < Remove or < Remove All button to move the event (or all events) to the Not Supported pane.

To rearrange (reorder) the sequence of events on the Supported pane:

1. Click the Exceptions row in the block's property table.
2. Click the button to open the Exceptions dialog box.

3. From the list of events on the Supported pane, select an event that you want to rearrange.
4. Do one of the following:

- To move the event higher in the sequence, click the Up button.
- To move the event lower in the sequence, click the Down button.

Notes:

- Each block has its own predefined set of events on the Exceptions property dialog box. Genesys recommends that you not remove any of the predefined events from the Supported list.
- Before generating code, each supported event must be handled by connecting its red node on the side of the block to the inport (input node) of another block.
- The events in the Entry block are global in scope.
- Events defined in other blocks are local to that block only. When an event is thrown, if a handler for that event is declared in the current block, that local event handler is called.
- If there is no local event handler for the event, but there is a global event handler declared in the Entry block, then the global event handler from the Entry block is called.

Condition Property

The Condition property indicates that the log will be active only if the given condition is true at runtime. To provide a condition setting for a log:

1. Select the **Condition** row in the block's property table.
2. Type the condition to evaluate against.

For example, assume in Entry block, there is a variable "MyVar==3". Assume also that you would like to log the session ID (GVPSessionID variable in Entry block) for all sessions where MyVar=3. In this case you must set the condition to "AppState.MyVar=3". If this condition is true, then GVPSessionID will be written to the log, otherwise it will be ignored.

Enable Status Property

This property controls whether or not a block contributes code to the application. Diagrams visually indicate when a block is disabled. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later. You can also right-click a block and select **Toggle Enable Status**. The GVP Debugger skips over deactivated blocks.

Logging Details Property

Logging details contains the expression that will be logged at runtime by GVP. If logging details are specified, then logging is generated for the block; if no logging details are specified, no logging is generated. To create logging details:

1. Click the **Logging Details** row in the block's property table.
2. Click the button to open the Logging Details dialog box.
3. In the Logging Details dialog box, click **Add** to open Expression Builder.
4. Create an expression to be used for logging details, such as an expression based on the variables whose content you wish to log.

Log Level Property

To assign a value to the Log Level property:

1. Select the **Log Level** row in the block's property table.
2. In the **Value** field, select one of the following from the drop-down list:

   - **Project Default**. The block uses the project's default log level, which can be configured through the Project properties.
   - **Info**. This is an Informational level to log application-specific data.
   - **Debug**. This is a Debug level used for application debugging.
   - **Error**. This is an Error level to log error details.
   - **Warn**. This is a Warning level to flag any application warnings.
   - **Alarm**. This is an Alarm level to send the message as an alarm to the Genesys management framework.

Prompts Property

Use the Prompts property to specify the audio prompts that are played to the caller. You can specify pre-recorded prompts, text, video, and several standard data types. SSML tags can be used inline in TTS prompts. To add, delete, or arrange prompts:

1. Click the Prompts row in the block's property table.
2. Click the button to open the Set Prompts Properties dialog box.

Set Prompt Properties Dialog Box

Prompt Messages Area

- **Name**--Displays the name of the prompt based on what you enter under the Prompt Details area.
- **Value***--Displays the prompt's value based on what you enter in the Prompts Details area.
- **Interpret-As**--Displays the data type of the prompt. The table below details available selections.

Prompt Details Area

Type--Displays whether the prompt is **ARM**, **Resource**, **Value**, or **Variable** based on what you enter under the Prompt Details area. Note that when **Type** is **Variable**, the runtime values of the specified variable should be of type string. Numerical values should be quoted, e.g. when assigning a value using the Assign block, or during a debugging session. **Type/Interpret-As Combinations** When **Type** is set to **Value** and

- Interpret-As is set to **Time**, you can select the Time Format in the drop-down list. The time format is displayed in 12-hour mode (1:00 PM, 2:00 PM, and so on), or 24-hour mode (13:00, 14:00, and so on).
- Interpret-As is set to **Audio**, you can specify an HTTP or RTSP URL.

When Type is set to **Variable** and **Interpret-as** is set to **Custom**, a Custom-Interpret As field is enabled, which can be used for custom prompt types as detailed in the table below. When Type is set to **Resource** and Interpret-As is set to **Audio**, an Alternative Text field displays. This alternative text is played back to you in the event that the audio file is not available. When Type is set to **ARM** and Interpret-As is set to **Audio**, you can specify a base URL, audio resource ID, and personality ID. These can be used for managing audio resources in the arm (Audio Resource Management (ARM)) section of the Genesys Administrator Extension Server Application object. When Type is set to **Variable** and Interpret-As is set to **Audio**, you can specify a variable that contains an HTTP or RTSP URL. This applies to the Prompt, DB Prompt, Input, Menu, and Record blocks. **Add Button** Use the Add button to enter prompt details.

1. Click **Add** to enable the fields.
2. In the **Name** box, accept the default name or change it.
3. From the **Type** drop-down list, select **ARM**, **Resource**, **Value**, or **Variable**.
4. In the **Interpret-As** drop-down list, select from among the data types shown in the following table:

AUDIO	Plays an audio sound file. Notes: If you select Audio, an audio file is optional, and you select the audio file if needed using the Browse button. Use the Clear button to remove an audio resource file selection. You can then specify an audio resource URL through Expression Builder, an audio resource identifier, personality identifier, and audio format. When you select ARM from the Type dropdown list, Interpret-As defaults to Audio. The VOXFILEDIR variable in the Entry block defines the audio file directory. For more information, see the Entry block help. You can also specify an alternative text for the audio file. This alternative text is played

	back to you in the event that the audio file is not available or is not provided. Typically, you can use this option during development, when the production audio files are not recorded yet.
BOOKMARK	An indicator that sets the place in a sequence of prompts. It can be used to detect the barge-in position during playback of a prompt. It uses the TTS engine.
CURRENCY	An optional currency specifier followed by a number with at most two decimal places. The currency specifier can be: • $, British pound sign, yen sign, or Euro sign, OR • 3-character ISO4217 currency code In the U.S. English locale, 11234 would be spoken as "eleven thousand, two hundred and thirty-four dollars."
DTMF	Plays DTMF tones. Any string of numerical digits, the characters a to d, #, or *
DATE	Speaks the specified date. yyyyMMdd, e.g. 20080604 Note: If you select the DATE type, click the drop-down arrow to display a calendar from which you can select the date.
NUMBER	Speaks a number. For example, 1234 would be spoken as "one thousand, two hundred, thirty-four." Any integer (no decimals)
ORDINAL	Speaks the number as an ordinal. For example, 1 would be spoken as "first." Any integer (no decimals)
STRING	Speaks a string of letters or numbers one character at a time. For example, 1234 would be spoken as "one, two, three, four." Note: The STRING type for U.S. English local accepts 0-9, A-Z, and +<=%->&.#*@. All other locales accept only 0-9 and A-Z.
TEXT	Plays the specified text with text-to-speech software
TIME	Speaks the specified time. hhmm[ss][?hap] (seconds is optional, and format specifier is optional) The format specifiers mean the following: ? -- neither am or pm, e.g. two o'clock or two fifteen h -- 24-hour clock, e.g. fourteen hundred hours or fourteen fifteen a -- AM, e.g. two AM or two fifteen AM p -- PM, e.g. two PM or two fifteen PM If no format specifier is given, it defaults to ?, i.e. am/pm is unknown. Note: 12 hour time selection will show the Time value in 24 Hr format in the Prompt Message Table. (e.g. 1:45:39 PM will be

	shown as 134539) whereas it will work as expected in the generated code to read the value in 12 hour format during runtime.
VIDEO	Use to allow VoiceXML to insert text into an existing video image/stream. If Video is selected, you can check the Enable Text Overlay box. • Click the Fx button to open the Video Text Overlay dialog box. • Click Add to specify: text (required), font name, font style, font color, background color, font size, font width, X axis offset, and Y axis offset.
CUSTOM	This Interpret-As option can be used to define Custom Prompts to customize the Prompt reading functions. To define a Custom Prompt: • Open the predefined `customprompts.js` file inside each language locale folder applicable for the Project. (`Resource\Prompts\$Language$`). • Use the `customprompts.js` file present inside to define custom prompt methods. • Refer to the syntax and rules mentioned in the default `customprompts.js` file inside`./Resources/Prompts/en-US` folder. • Start each Custom Prompts methods with the language locale name to achieve Multilangual support during runtime execution (mandatory). The Prompts property dialog will only parse methods defined in the `customprompts.js` file. During design time, the default language locale `customprompts.js` file is parsed and listed for method selection. During the runtime call, the APP_LANGUAGE variable value is used to dynamically select the language local folder. • Use 'audio' option to play audio files in the Custom Prompts methods using <audio> tag and 'value' option to play expressions using <value> tag.

5. In the Value box, enter data for the selected data type.

Place the audio files in the Resources\Prompts\{APP_LANGUAGE} folder under the Java Composer Project. Audio files can be added to the project by copying and pasting from the Windows file system into the Java Composer Project in the Project Explorer. Note: By default, Genesys supplies .vox files only for mulaw 8Khz. If you are using any other audio format for playback of audio files, replace the files with the corresponding audio files in the required audio format. **Up/Down Buttons** Use the

Up and Down buttons to reorder your prompt elements. Select the element you want to reposition, and then click Up or Down, as necessary. **Delete Button** To delete a prompt:

1. Select an entry from the list.
2. Click **Delete**.

This property is used in the following blocks: Prompt Block Menu Block Input Block Record Block

Retry Prompts Property

The Retry Prompts property in a Menu block, Input block, or Record block enables you to set different retry prompts that are played to the caller when the voice application encounters a nomatch or noinput condition. You are allowed up to three retries for either a noinput or a nomatch error condition. You must select the listed items in sequence and add the necessary vox file or text input. To set retry prompt properties:

1. Click the Retry Prompts row in the block's property table.
2. Click the button to open the Retry Prompts dialog box.

Note: You must set the Number Of Retries Allowed property to a value greater than 0 in order to have access to the Retry Prompts dialog box. **Prompts Fields**

- Name*-- Displays the name of the retry prompt.
- Type--Displays whether the retry prompt is a Resource, Value, or Variable.
- Interpret-As-- Displays the data type of the retry prompt.
- Alternate Text--(Enabled only when Interpret-As is set to Audio.) This alternative text is played back to you in the event that the audio file is not available.
- Value*--Displays the retry prompt's value (Retry Prompt).

Note: When Interpret-As is set to Time, you can select the Time Format in the drop-down list. The time format is displayed in 12-hour mode (1:00 PM, 2:00 PM, and so on), or 24-hour mode (13:00, 14:00, and so on).

Retry Prompt Messages Property

For Input and Menu Blocks:

After setting a value for the Number Of Retries Allowed property, Retry Prompt Messages will contain one noinput and one nomatch entry per retry. For example, if Number Of Retries Allowed is set to 2, the Retry Prompt Messages table contains the following entries: `noinput1 nomatch1 noinput2 nomatch2`

For Record Blocks:

Retry Prompt Messages will contain one noinput entry by default. To set or change retry prompt properties:

1. Select a retry prompt in the Retry Prompt Messages table to enable Prompt Details fields.
2. In the Name* box, accept the default name or change it.
3. From the Type drop-down list, select Resource, Value, or Variable.
4. In the Interpret-As drop-down list, select from among the data types shown in the following table:

AUDIO	Plays an audio sound file. This is available only when Resource or Variable is selected as the Type. Note: If you select Audio, an audio file is optional, and you select the audio file if needed using the Browse button. Use the Clear button to remove an audio resource file selection. The VOXFILEDIR variable in the Entry block defines the audio file directory. For more information, see the Entry block help. You can also specify an alternative text for the audio file. This alternative text is played back to you in the event that the audio file is not available or is not provided. Typically, you can use this option during development, when the production audio files are not recorded yet.
BOOKMARK	An indicator that sets the place in a sequence of prompts. It can be used to detect the barge-in position during playback of a prompt. It uses the TTS engine.
CURRENCY	An optional currency specifier followed by a number with at most two decimal places. The currency specifier can be: • $, British pound sign, yen sign, or Euro sign, OR • 3-character ISO4217 currency code In the U.S. English locale, 11234 would be spoken as "eleven thousand, two hundred and thirty-four dollars."
DATE	Speaks the specified date. yyyyMMdd, e.g. 20080604 Note: If you select the DATE type, click the drop-down arrow to display a calendar from which you can select the date.
DTMF	Plays DTMF tones. Any string of numerical digits, the characters a to d, #, or *
NUMBER	Speaks a number. For example, 1234 would be spoken as "one thousand, two hundred, thirty-four." Any integer (no decimals)
ORDINAL	Speaks the number as an ordinal. For example, 1 would be spoken as "first."

	Any integer (no decimals)
STRING	Speaks a string of letters or numbers one character at a time. For example, 1234 would be spoken as "one, two, three, four." Note: The STRING type for U.S. English local accepts 0-9, A-Z, and +<=%->&.#*@. All other locales accept only 0-9 and A-Z.
TEXT	Plays the specified text with text-to-speech software
TIME	Speaks the specified time. hhmm[ss][?hap] (seconds is optional, and format specifier is optional) The format specifiers mean the following: ? -- neither am or pm, e.g. two o'clock or two fifteen h -- 24-hour clock, e.g. fourteen hundred hours or fourteen fifteen a -- AM, e.g. two AM or two fifteen AM p -- PM, e.g. two PM or two fifteen PM If no format specifier is given, it defaults to ?, i.e. am/pm is unknown. Note: 12 hour time selection will show the Time value in 24 Hr format in the Prompt Message Table. (e.g. 1:45:39 PM will be shown as 134539) whereas it will work as expected in the generated code to read the value in 12 hour format during runtime.

5. In the Value box, enter data for the selected data type, or keep the default value of Retry Prompt.

See template samples that use the Menu or Input blocks.

Working with Grammar Builder

Grammar Builder provides a solution for supplying simple grammars, without requiring GRXML expertise. This editor provides a hierarchical view of certain grammar concepts in a simple, abstract way. Each level of this tree contains properties which affect all child members. Following is a brief description of the key concepts of the Grammar Builder model as well their relationship with GRXML.

Note: A Grammar built with the Grammar Builder is not a GRXML file.

Grammar

The grammar is the root object of the tree. It serves to provide an implicit description of the intended use of a grammar. For example, a grammar which would be used for a bank customer could be called bankmenuinput. The selection of a grammar name is determined by the file name or the related gbuilder file, and its setting influences the file name(s) of any exported GRXML data.

Within the grammar object are properties for the setting of languages. These languages, or locales, indicate support for a particular language. For each locale that is added to a grammar, a distinct GRXML file will be created specifically to support that language. DTMF, or touch-tone input, is considered a language even though it is not spoken.

Rule

Every grammar must contain at least one rule, but may contain many. Rules provide a grouping for a spoken (or DTMF) items. Continuing the bank customer scenario, we could have rules for yes/no responses, another for menu options and perhaps another for branch cities. Rules are the product that is referenced in a voice application.

At a point in an application where we wish to retrieve the branch city, we must refer to that grammar’s rule. If an application designer does not specify a rule and instead only specifies the grammar file, the default rule is used. Additionally, a rule may be hidden from outside applications by declaring it as private. Usually this is for more sophisticated grammar cross-referencing, which is not currently supported in the more elementary Grammar Builder.

Keywords

Within a rule are specific keywords that will be used to add intelligence to an end application. Keywords become the value which can be identified within the application for use in branching or other application constructs. However, this keyword is independent of what may actually be spoken and is instead an internal identifier.

To bridge the gap between what a caller says or presses on their keypad, locale-specific synonyms are defined. Remember that the languages supported are defined at the grammar level. It is at this point that those defined languages come into play. Each keyword will have a list of words (synonyms) which relate to the keyword for a given language.

For example, assuming as part of our yes/no rule, we have a keyword for yes. This keyword could contain the word yes for English, 1 for DTMF and oui for French. Regardless of which locale ends up being used in the running application, yes (the keyword identifier) will be returned.

Working with Grammars will guide you through the process of creating a simple grammar, using a user color selection problem as the example to model.

Using the Grammar Builder

Let's create a simple grammar for use with a project and the Grammar Menu block. Our example will be a user color selection problem. You will perform the following steps:

1. Create a new grammar builder file with initial settings.
2. Add keywords and synonyms for a rule.
3. Save the grammar builder file.
4. Export the grammar builder file to standard GRXML format.

Composer provides a cheat sheet for building a simple grammar file:

- Select **Help** > **Cheat Sheets** > **Composer** > **Building Voice Applications** > **Creating a simple grammar**.

Creating a New Grammar Builder File With Initial Settings

The first step is to create a new grammar builder file and provide its initial settings. Follow these steps:

1. Select **File** > **New** > **Other**.
2. From the New dialog box, expand the **Composer** folder, then expand the **Grammars** folder.
3. Select Grammar builder file and click **Next** to continue.
4. In the Container field of the wizard dialog box, click **Browse** to select a project-specific folder to contain the new .gbuilder file. Genesys recommends `<voiceprojectname>/Resources/Grammars` for the location.
5. Set the file name to use for this grammar. File names should give an indication of the context this grammar will be used in. For example, type `colors.gbuilder` in the File name field.

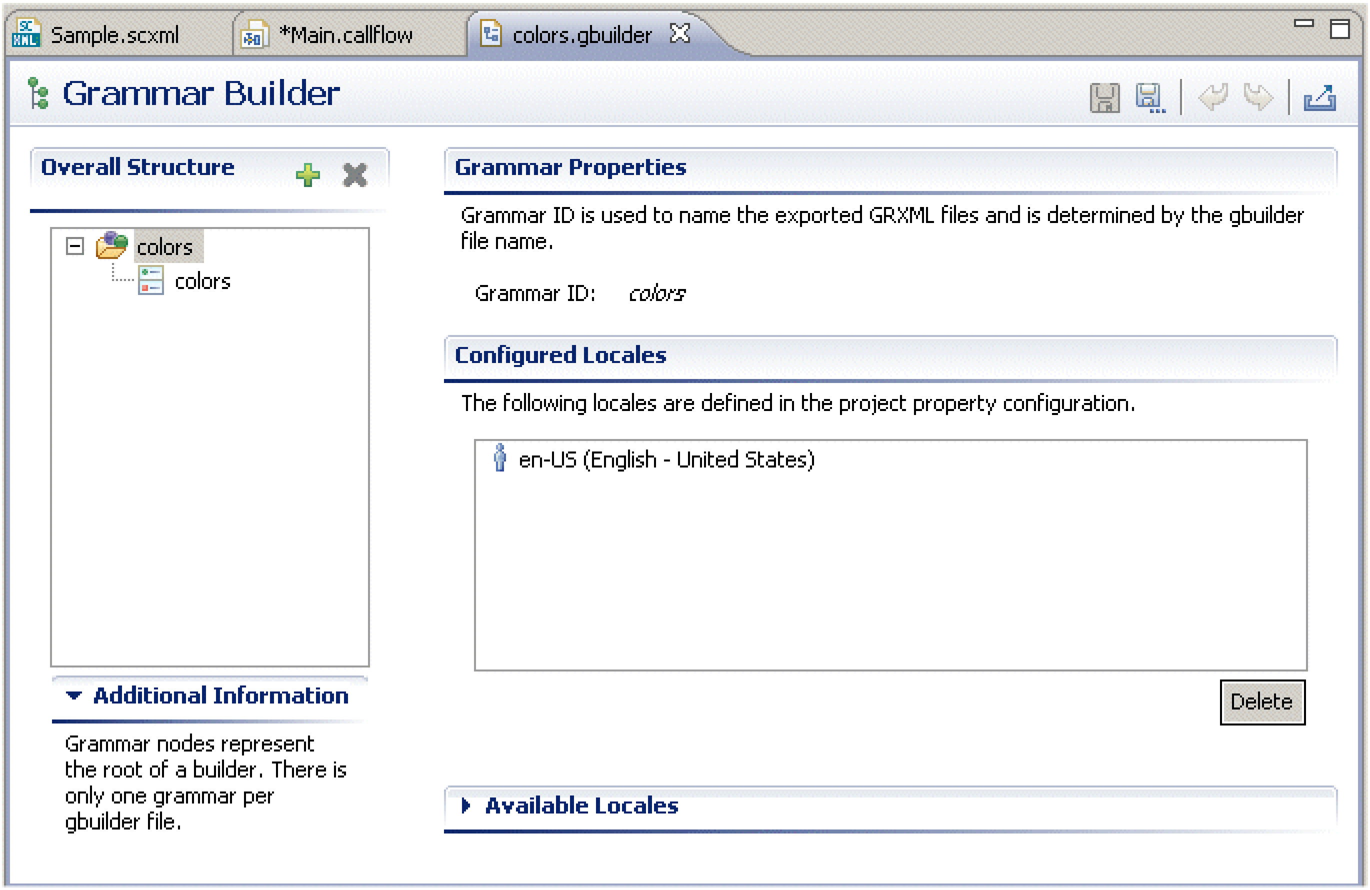
Note: The Grammar Menu block does not pick up changes automatically if you change your Gbuilder file. To synchronize the block with the latest changes, click on the **Gbuilder File** property of the Grammar Menu block. In the popup make sure that the correct **Gbuilder file** and **RuleID** are selected. Click **OK** to close the dialog. Your diagram will now reflect any menu options changes made in the Gbuilder file.

6. Next, set the initial default rule. Rules contain items which form a category. All grammar builder files must have at least one rule. Since our example grammar only deals with one such categorization, type `Colors` in the **Initial Default Rule** field.
7. *Locales* are languages that this grammar will support. By default, **English** and **digit input (DTMF)** are selected in the Initial locale(s) field. If you knew you would need to support additional languages for the grammar, you would select the appropriate check box(es). For our example, the default selections are adequate.

Note: Grammar Builder treats DTMF as a separate language (locale), even though technically it is not categorized as such.

8. After making the selections described above, click **Finish**.

The file is added to the selected project (as you can see in the Project Explorer), and the Grammar Builder opens as shown in the image below.



Grammar nodes represent the root of a builder. There is only one grammar per gbuilder file.

Grammar ID is used to name the exported GRXML files and is determined by the gbuilder file name.

The following locales are defined in the project property configuration.

Adding Keywords and Synonyms for a Rule

Grammar builder files are created with a default rule. The next step is to define keywords for this rule. Each rule can have any number of input-agnostic keywords. These keywords will be returned from either the speech or digit processor for use in your callflow.

By default, a keyword is not usable in an application. This is because multiple languages may use different words/sounds for your keyword. In our example, red may be an appropriate English pronunciation, but in Spanish this would not be true. Because of this, each configured locale must provide accepted input for the keyword. These inputs are called synonyms. Therefore, keywords consist of a logical identifier and a list of locale-specific synonyms.

Once you have defined keywords and synonyms for the default rule, you can then create additional rules and define keywords and synonyms for those rules as well.

To add a new keyword:

1. Select the **Colors** (default) rule in the **Overall Structure** tree.
    - The Rule Properties area shows the Public Visibility and Default Rule settings for the selected Rule ID.
    - **Default Rule**. This is selected only for the rule that has been set as the default (as is the Colors rule in this example).
    - **Note:** Not all aspects of Composer allow for specific rule targeting within grammar files (grxml). As such, it is highly recommended that you specify a default rule. This rule will be used by default when a reference to the grammar exists that does not target a specific rule. Considering that a default rule (e.g., root) is not mandatory in GRXML, no warning is given when one is not specified
    - **Public Visibility**. If selected, this indicates that this rule can be referenced by an external grammar (in a ruleref element in the grammar making the reference). A public rule can always be activated for recognition. If not selected, the rule is private, which indicates that the rule is visible only within its containing grammar. A private rule can be referenced by an external grammar if the rule is declared as the root rule of its containing grammar.
2. Click the ✚ (**Add**) button.
3. In the Add new keyword dialog box, type a name for this keyword, which is normally an instance of the category that the rule defines. In our example, type **Red** in the **Keyword ID** field and click **OK**.
4. You can repeat the steps above to add more keywords to this rule.

To add a new synonym:

1. Select a keyword from the **Overall Structure** tree. In our example, select **Red**.

The Synonyms area allows you to add synonyms for each of the locales you have defined (each locale is a tab at the bottom of the synonyms table). Note in our example that the window has both English - United States and DTMF as bottom tabs. This allows you to switch the synonym context for the selected keyword.

1. With the English - United States tab selected, click **Add ID** as **Synonym**. This button allows you to add a synonym that is identical to the keyword, thus allowing red to be spoken in English and associated with the keyword **Red**.
2. You may at this time add other values, such as **Crimson** for example, which will also be accepted as Red.

3. Select the **DTMF** tab. To associate the digit 1 with the keyword Red, type 1 in the Digits field and click the Add button.
4. You can repeat the steps above to add more synonyms to this keyword.

Note: If you are using locales representing other languages, the synonyms you create for each locale would represent acceptable values for the keyword in that language. In our example, if you also defined Spanish and French locales, you could create a synonym rojo for the Red keyword in the Spanish locale, and a synonym rouge for the Red keyword in the French locale.

Saving the Grammar Builder File

When you have finished building your grammar builder file, or periodically during the course of building the file, be sure to save the changes you make to the file.

5. To save the file, click (Save), or to save the file under a different name, click (Save As) and provide a new file name and location.

Exporting the Grammar to GRXML Format

Because the Grammar Builder saves your grammar to a non-standard GRXML format (denoted with a .gbuilder extension on the file name), you will want to export the grammar to the standard GXML format as follows:

1. Click (**Export)** , located at the top-right corner of the Grammar Builder editor.
2. If prompted to save, click **Yes**.

You will see a message indicating the file has been successfully exported. The exported GRXML file names are displayed in the success window, and the .grxml file will display in the appropriate locale folder(s) in the Project Explorer under `<voiceprojectname>/Resources/Grammars`. It's important to note that DTMF is considered a locale for the purpose of exportation. As such, an export result for a GBuilder resource with English and DTMF would be placed in `<voiceprojectname>/Resources/Grammars/en-US` and `<voiceprojectname>/Resources/Grammars/DTMF directories`, respectively. These files can now be edited in the GRXML Editor.

Locales and Grammar Builder

When using the Grammar Builder, you specify *locales*, which are the languages that a grammar file will support. The Grammar builder wizard uses the active locales for the Composer Project.

See Locales in Common Blocks & Functionality.

Dynamic Grammars

Dynamic grammars are used for automated speech recognition (ASR). They are generated "on-the-fly" based on information dynamically pulled out from data sources such as databases, web services, or the file system. Contrast this to using a static grammar file whose content is fixed. The ASR engine matches the user utterance with the grammar. Returned values are then passed back to the application based on any matches in the grammar.

There are several ways to include dynamic grammars in voice dialogs:

- Use a dynamic VXML page template that creates the dynamic grammar and insert it in-line into the VXML page. Using a dynamic VXML page will provide flexibility in terms of the data source used to generate the grammar.

- If data is being retrieved from a database, using the DB Input block may be another alternative. It generates a grammar based on data retrieved from a database using the DB Data block. It can also generate a grammar based on contents of a JSON array that may have been retrieved from alternate data sources e.g., a Web Service.

Working with CTI Applications

Composer provides CTI blocks for two CTI scenarios supported by GVP:

- SIP Server (SIPS) scenario, which uses the Genesys SIP Server component to gain access to CTI functionality.
- CTI Connector (CTIC) scenario, which uses GVP's CTI Connector component to access CTI functionality provided by Genesys Framework.

These two scenarios do not provide identical capabilities and key differences are highlighted later in these topics. Composer provides four CTI blocks for accessing CTI functions. It generates VXML for each of these blocks that can work in either CTI scenario (SIPS or CTIC), and does not ask the user to choose between the SIPS or CTIC scenarios at design time. The decision to use CTIC or SIPS is made at runtime based on the X-Genesys headers received from GVP's Resource Manager. Therefore, the Composer user interface does not need to expose a Project-level preference for specifying the CTI scenario. **Note:** The CTI Connector provides different capabilities depending on the configuration in which other Genesys components like the IServer are deployed. For more details, please refer to the GVP documentation. Also see GVP Debugging Limitations.

Design Paradigms for CTI Applications

There are two design paradigms for building CTI applications with GVP in which Composer can be used:

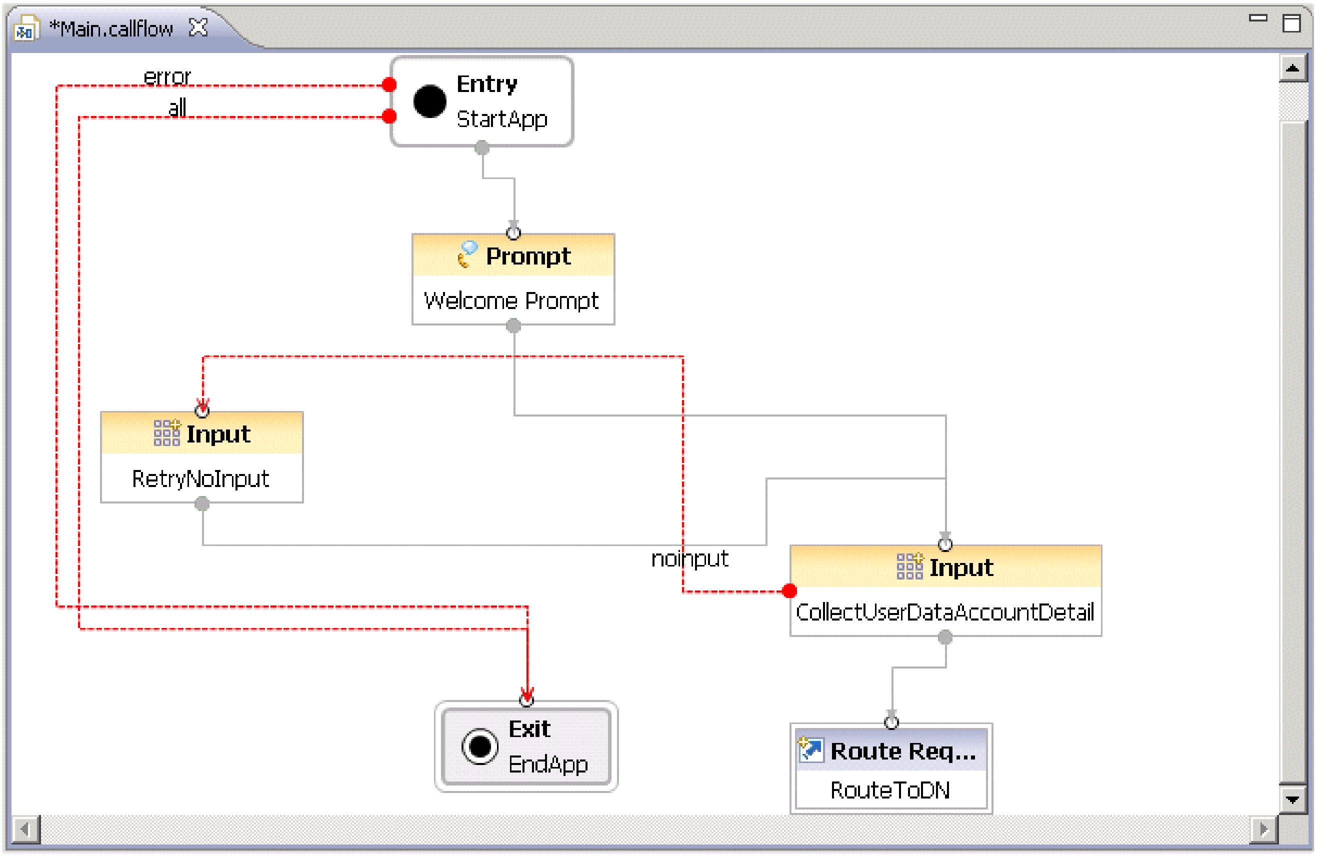
- Standard VXML Applications
- URS-Centric Applications

These paradigms differ in the extent to which the VXML application is involved in performing call control. **Standard VXML Applications** In this paradigm, the VXML application gets invoked first and can go through VXML interactions with the caller before using the <transfer> tag to transfer the call to another party such as queuing for an agent. At this point, the control of the call is passed to the SIP Server or CTI Connector while waiting for an agent. During this time, SIP Server or CTI Connector may invoke additional call treatments on GVP like playing music or invoking other applications. **URS-Centric Applications** In this paradigm, the VXML application is always invoked as a treatment by Genesys URS. The incoming call is controlled by Genesys URS and a strategy retains full control of the call. The strategy invokes specific treatments on GVP IVR as a media server to play prompts, play music, collect user input or execute a VXML application. In this paradigm, the VXML application does not use tags like <transfer> nor does any other kind of call control. Those decisions are left to the strategy. The VXML application returns user input collected during the call back to the strategy and lets the strategy make all call control decisions. Composer can be used to write VXML applications following either of the above paradigms.

Typical CTI Callflow

Before you start building a typical CTI application, the following information is required:

- The Genesys Virtual Route Point destination address. This is the address/location where the Genesys strategy is present (an integer number--for example, 5001).
- Strategy application on the Framework side (IRD) to find and transfers the call to an agent.



The following describes the interaction flow of this callflow:

1. GVP starts executing the generated VoiceXML application script.
2. The caller hears the Welcome prompt.
3. The caller is requested to enter the account details.
4. If the caller does not enter the required details within the maximum time frame provided, the caller is asked to retry.
5. The application issues a route request to the route DN configured in the Route Request block. (This occurs via the <transfer> tag, supported in both CTIC and SIP Server scenarios.)
6. The caller-entered data is sent as UserData to the routed DN, and the called strategy does the knowledge based transfer to the available agent based on the User Data .

7. This application ends after the Route Request has been issued.
8. The called strategy can play Voice treatments to the caller until the next available agent is available.
9. Finally, the caller will be transferred to the Agent.

Note: The Route Request block can be configured in various Transfer modes (Bridge / Consultation) to gain back the control of the callflow after the called strategy returns back the execution. Please check the Route Request topic block for more details.

CTI Scenarios

There are feature differences between the SIPS and CTIC scenarios. The following table gives a summary of the CTI blocks, and for each CTI block it lists the differences in behavior for the two CTI scenarios.

CTI Block Name	Supports CTIC Case?	Supports SIPS Case?	Comments
Interaction Data	Yes	Yes	Supported operations in each scenario: CTIC: • PUT • GET • DELETE • DELETEALL • REPLACE SIPS: • PUT • GET Types of interaction data supported: CTIC: • USERDATA SIPS: • USERDATA
Get access number	Yes	No	Get access number block can only be used in the CTIC scenario. Types of interaction data supported: CTIC: • USERDATA • EXTENSIONDATA

Statistics	Yes	No	Statistics block can only be used in the CTIC scenario.
Route Request	Yes	Yes	Types of interaction data supported: CTIC: • USERDATA • EXTENSIONDATA SIPS: • USERDATA Types of transfers supported: CTIC: • Blind • Bridge SIPS: • Consultation • Blind • bridge

In case a CTI block or feature is used in a CTI scenario in which it is not supported, appropriate exceptions will be thrown at runtime indicating that the feature is not supported. The table below gives a list of all exceptions that can be thrown by CTI blocks and other possible CTI-related exceptions that can be thrown if errors are encountered at runtime.

Block(s)	Exception	Error Message	Description
Interaction Data Get access number Statistics	error.com.genesyslab.composer.invalidkey	Missing <block name> key <key name>	This is the event error for handling an invalid key name.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.operationtimedout	Operation timed out.	This exception will be thrown when a <receive> operation, executed in the context of a CTIC specific operation, times out.
Interaction Data Get access number Statistics Route Request	error.com.genesyslab.composer.receiveerror	<Error string returned by CTIC>	If the <receive> fails and an error is reported by CTIC, this exception will be thrown.
Interaction Data	error.com.genesyslab.composer.unsupported	Delete operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata DELETE in the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupported	DeleteAll operation not supported in case of CTI	If the user wants to do a userdata DELETEALL in

		using SIPServer.	the CTI using SIPS scenario.
Interaction Data	error.com.genesyslab.composer.unsupported	Replace operation not supported in case of CTI using SIPServer.	If the user wants to do a userdata REPLACE in the CTI using SIPS scenario.
Get access number	error.com.genesyslab.composer.unsupported	AccessNumGet operation not supported in case of CTI using SIPServer.	If the user wants to do a AccessNumGet in the CTI using SIPS scenario.
Statistics	error.com.genesyslab.composer.unsupported	Statistics block not supported in case of CTI using SIPServer.	If the user wants to do a PeekStatReq or GetStatReq in the CTI using SIPS scenario.
Route Request	error.com.genesyslab.composer.unsupported	Consultation transfer is not supported in case of CTI using CTIConnector.	If user sets Transfer type to consultation in case of CTI using SIPS.

Script ID Usage in the GVP 8 Environment

In Genesys VoiceXML 2.1, ScriptId refers to the script identifier, as generated by the CTI Connector, to handle call treatments. The use of ScriptId is specific to GVP 7.x and was mandatory for treatments. Since the GVP 7.x design is "IVR-centric," the treatment would be invoked on the same VXML session. Things are a bit different with GVP 8.x and the Next Generation Interpreter (NGI) where APP_URI is used instead of ScriptId and the treatments are executed on different VXML sessions. **GVP 8 and NGI** In GVP 8.x, request for treatment execution comes in as a NETANN request with the APP_URI being passed in as a VoiceXML parameter. GVP executes the requested page to kick off the treatment. Unlike the GVP 7.x environment, treatments get invoked as separate VXML sessions and terminated at the end of the treatment execution. Hence, ScriptId switching is no longer needed here, unless an application wants to do branching based on ScriptId.

- Note: Composer provides support for both SIPS and CTIC scenarios for achieving the CTI functionality. However, SIPS may not support passing additional request-uri parameters like ScriptId, therefore, this option is limited only to CTIC scenarios.

Please refer to *GVP 8.x VXML Help* under Sample Voice XML Applications > CTI Interactions > Treatments for more details on this topic.

Accessing ScriptId in Composer

Use if you want your application to do ScriptId-based switching like GVP 7.x. **CTIC Scenario (IRD strategy + Composer Callflow)**

1. Use the APP_ID property in IRD's Play Application block.
2. Define a new Input type variable named ScriptId in the Entry block of your callflow to collect the ScriptId.

Composer Workflow + Composer Callflow)

1. On the VXML callflow side, define a new Input type variable named ScriptId in the Entry block to collect the APP_ID (i.e., ScriptId) passed from the workflow.
2. On the SCXML workflow side, use the Play Application block to invoke the callflow created using step#1. Then do an auto-synchronize for the parameters, and specify the ScriptId value.
3. The ScriptId (i.e., APP_ID) passed from the workflow will be automatically collected on the VXML side from the session.connection.protocol.sip.requesturi array.

SIPS Scenario

1. SIPS may not support passing additional request-uri parameters. Pass ScriptId as attached data on the strategy side (If using IRD) or on the SCXML side (If using Composer workflows).
2. Define a new Input type variable named ScriptId in the Entry block to collect the ScriptId.
3. The ScriptId (i.e., APP_ID) passed from the strategy will be automatically collected on the VXML side from the session.com.genesyslab.userdata array.

Working with Prompts

Note: There is both a Prompts Manager perspective and a Prompts Manager perspective. The Prompts Manager provides the ability to quickly review all prompts in a Composer Project from a central place. It displays the relevant information about all prompts in all callflows present in your workspace, one project at a time. It displays prompt item text, associated audio file(s) and allows you to play prompt resource files directly from the Prompts Manager view. The Prompts Manager also enables you to tweak your prompts by rearranging prompt items, changing prompt item text, and recording new audio files using microphone input and associating them with prompts in your Projects. It provides the ability to quickly jump to a specific prompt block in the callflow diagram with a simple right-click so that other changes can be made to the prompt. Prompt Manager works in conjunction with the prompts properties popup dialog.

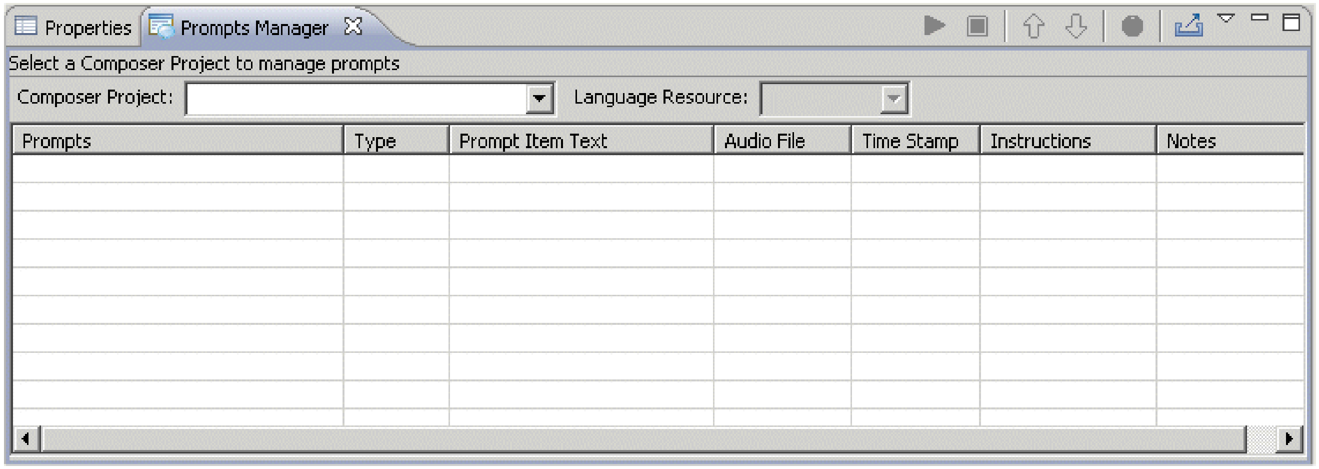
Opening the Prompts Manager

To open the Prompts Manager perspective:

- Select **Window** > **Open Perspective** > **Other** > **Prompts Manager**.
- Or select the Prompts Manager Perspective toolbar button.

To open the Prompts Manager view:

- Select **Window** > **Show View** > **Prompts Manager**.



Selecting a Composer Project

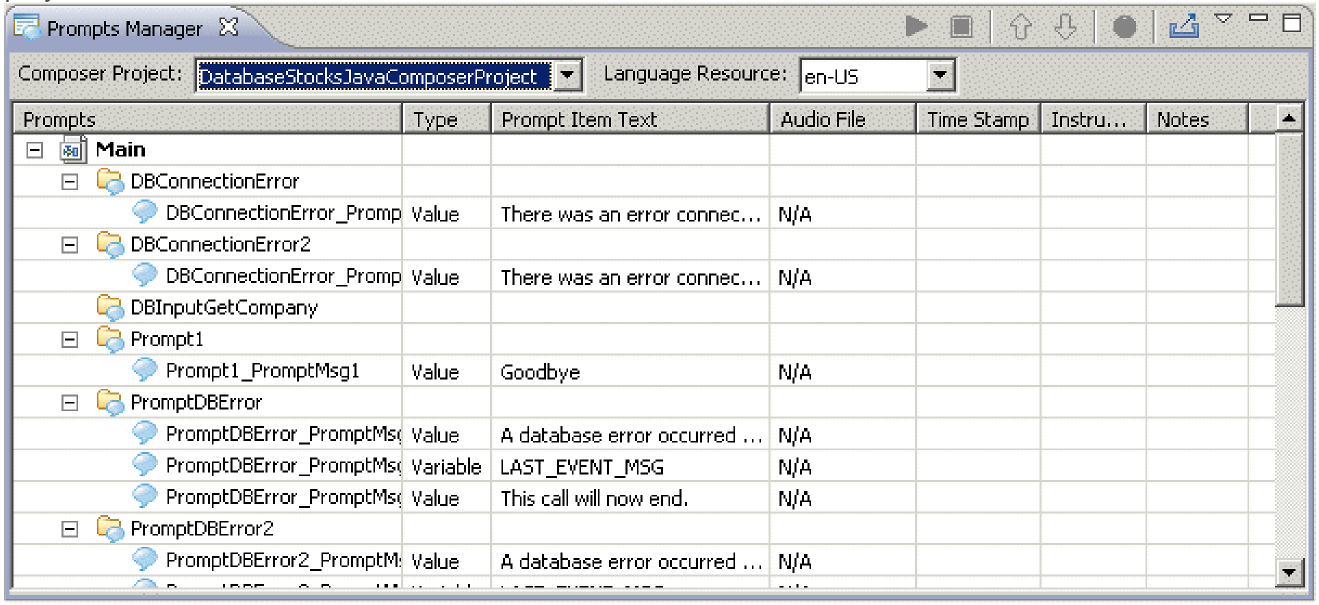
To select a Composer Project for which to manage prompts, the Prompts Manager view:

- Select a project from the Composer Project drop-down list.

Selecting a Language Resource

- Select a Language Resource. If not using en-US, see the special note on Non-US Locales.

The Prompts Manager displays all the prompt-related blocks for all the callflows in the selected project.



Composer Project: DatabaseStocksJavaComposerProject   Language Resource: en-US

Prompts	Type	Prompt Item Text	Audio File	Time Stamp	Instru...	Notes
Main						
DBConnectionError						
DBConnectionError_Promp	Value	There was an error connec...	N/A			
DBConnectionError2						
DBConnectionError_Promp	Value	There was an error connec...	N/A			
DBInputGetCompany						
Prompt1						
Prompt1_PromptMsg1	Value	Goodbye	N/A			
PromptDBError						
PromptDBError_PromptMs	Value	A database error occurred ...	N/A			
PromptDBError_PromptMs	Variable	LAST_EVENT_MSG	N/A			
PromptDBError_PromptMs	Value	This call will now end.	N/A			
PromptDBError2						
PromptDBError2_PromptM	Value	A database error occurred ...	N/A			

Notes

- The Prompts Manager will not work with non-Composer projects (such as hand-coded applications).
- Note the Language Resource dropdown selection box. The prompt audio resource is located in the appropriate language resource folder location (for example:`/Resources/Prompts/<locale>/audioResourceFile.vox`.
- Starting with 8.0.2, a Composer Project upgrade sets the default Project locale to en-US. If other than en-US, right-click the Composer Project in the Project Explorer, and select **Properties** > **Locales** to set the default and active locales.

Columns in the Prompts Manager View

A prompt item row in the Prompts Manager view displays the following column details:

- **Prompts** -- A tree hierarchy consisting of the following elements; Root elements, studio diagram callflow/sub-callflow file name. Studio diagram elements may have diagram block elements. These diagram block elements may have prompt/retry prompt item elements.
- **Type** -- The type of prompt item (audio resource/value/variable)
- **Prompt Item Text** -- Any associated text with the prompt item.
- **Audio File** -- The relative path of the audio file associated with this prompt item.
- **Time Stamp** -- The Date/Time stamp when the audio file was created or recorded. This helps in identifying the newer/older prompts.
- **Instructions** -- If additional item specific information needs to be given, such as a certain word needs to be emphasized when recording at the recording studio.
- **Notes** -- Any notes that you would like to associate with the prompt item for later reference.

Non-US Locales

By default, Composer provides prompts audio resources for the en-US locale. The supplied `PlayBuiltinType.js` under **Resources/Prompts** in the Project Explorer defines a global variable called **promptBaseUrl** with the value **en-US**. When using a different locale in a callflow (other than en-US in the Language Resource field in Prompts Manager), you must provide the associated audio files and `PlayBuiltinType.js`. Adjust the path with the associated prompt resource locale folder path.

Reviewing and Managing Prompts

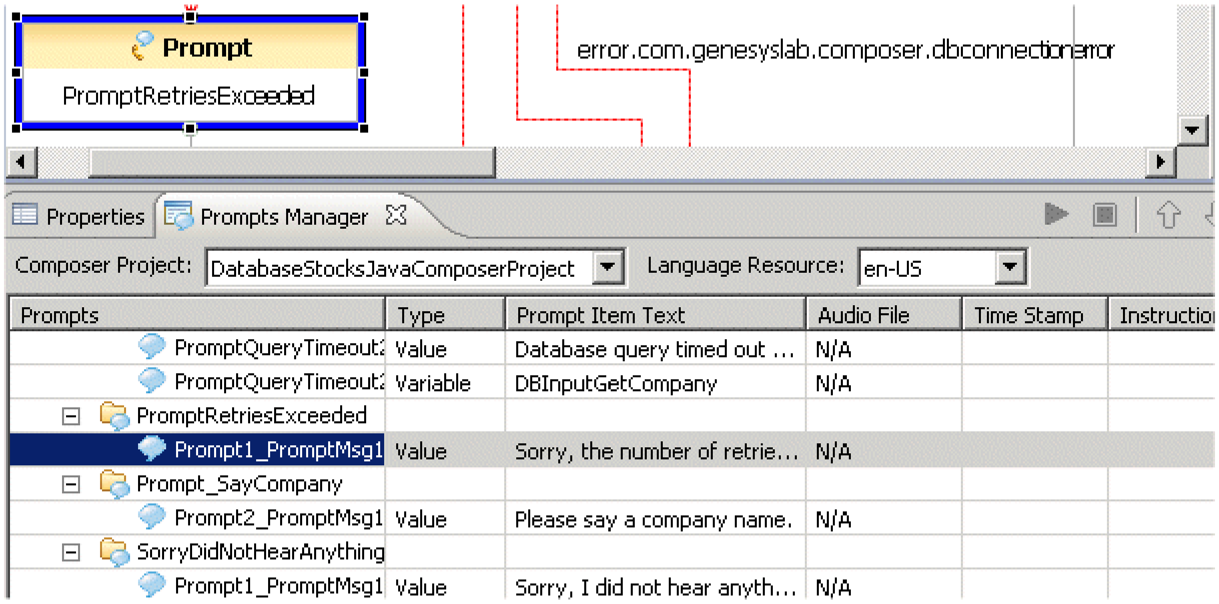
Once you have laid out your diagram and wish to review the flow of the application, you can use the Prompts Manager to do the review. It is useful to have the callflow and Prompts Manager view open together so that the flow of the application can be traced using the callflow while reviewing prompts using the Prompts Manager. Select the Composer Project(s) containing prompts you wish to manage. You can review and manage your prompts as follows:

1. Expand a Prompt block in the Prompts column of the Prompts Manager view to display all prompts associated with that block.
2. Select a prompt row. The Prompts Manager view displays detailed information about the prompt.
3. You can view prompt item text in the Prompts Manager.

    - For prompts that have an associated audio file, click the Play icon ▶ in the Prompts Manager view to hear the audio file.
    - Click the Stop icon ■ to stop playing the audio file.

Note: To play back VOX audio files in their correct encoding (U-Law/A-Law), you may need to set the encoding properties in the Composer Project settings. To change the settings, go to the Project Explorer, right-click the Composer Project folder, and select Properties. Select the Prompts Management section and set the Encoding property accordingly.

4. To modify the sequential order of the prompt items within a block, select a prompt item element row and click the Up or Down icon ⇧ ⇩ in the Prompts Manager view.
5. To locate the diagram block in the studio diagram callflow that is associated with a selected row, right-click a prompt row and select Display in callflow from the context menu. If the callflow (or studio diagram) is not currently open, it will be opened in the editor and the selected block will be highlighted with a blue outline as shown below.



Composer Project: DatabaseStocksJavaComposerProject   Language Resource: en-US

Prompts	Type	Prompt Item Text	Audio File	Time Stamp	Instructio
PromptQueryTimeout:	Value	Database query timed out ...	N/A		
PromptQueryTimeout:	Variable	DBInputGetCompany	N/A		
⊟ PromptRetriesExceeded					
Prompt1_PromptMsg1	Value	Sorry, the number of retrie...	N/A		
⊟ Prompt_SayCompany					
Prompt2_PromptMsg1	Value	Please say a company name.	N/A		
⊟ SorryDidNotHearAnything					
Prompt1_PromptMsg1	Value	Sorry, I did not hear anyth...	N/A		

6. To modify a value (for example, the prompt item name, the prompt item text, and so on) from within the Prompts Manager view, double-click the table cell, type a new value, and press Enter. Certain table cell values may not be modified.(for example, callflow diagram name, prompt type, and so on).

Supported Audio File Formats Audio files are encoded and outputted in various audio file formats. The following audio file formats are recommended and supported for playback and recording within Prompts Manager:

File Extension	Sample Rate	Sample Size	Bit rate (Bandwidth)	File Format	Encoding
.vox	8000 Hz	8-bit	64 bits/sec	Raw audio (mono)	u-law
.vox	8000 Hz	8-bit	64 bits/sec	Raw audio (mono)	a-law
.wav	8000 Hz	8-bit	64 bits/sec	Audio with .wav header (mono)	PCM

Refer to the VoiceXML 2.1 Reference Help on the Genesys Voice Platform Wiki for additional formats that GVP supports. Those additional formats can be played back and recorded using third party tools outside of Composer.

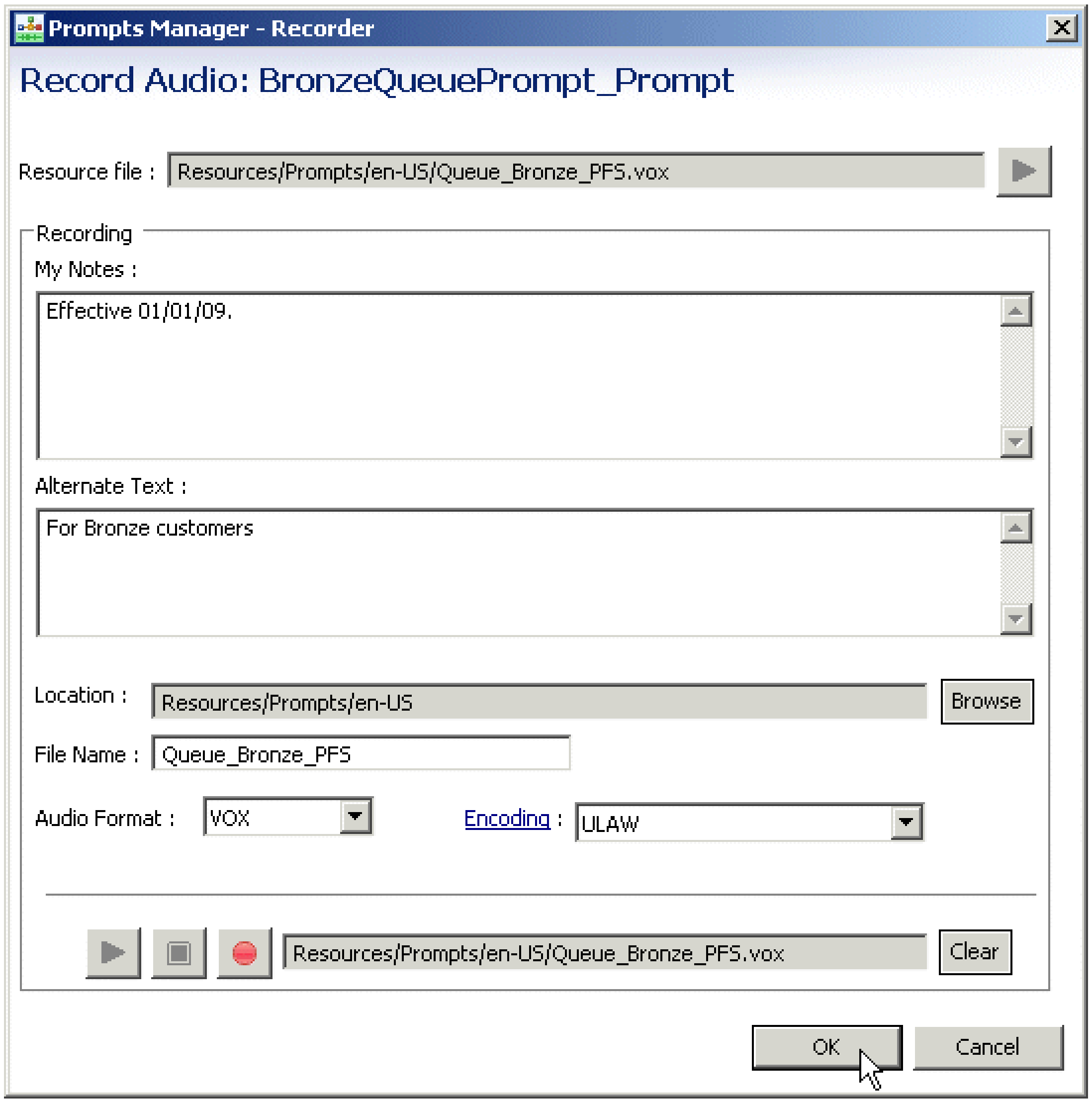
Recording Prompts

The Prompts Manager view provides a button to launch a recorder/player that can record and play back a single prompt item’s audio file. The newly-recorded file will replace any existing audio file associated with the highlighted prompt item. **Notes:**

- Related prompt audio settings are located in the Composer Project Settings. In the Project Explorer, right-click the Composer Project folder, and select Properties. Select the **Prompts Management** section for prompt audio settings.
- To record a prompt item using Prompts Manager, the prompt item must be of type Resource in the Prompts Manager view. If you do not want to specify an audio resource at this time or wish to record your own resource prompt using the Prompts Manager, you may instead define a value in the **Alternate Text** field shown below.
- Please note for recording prompts that are of type **Value** to interpret-as "Text," you will need to change the prompt type to **Resource**. Supply the prompt text value in the **Alternate Text** field for the Resource prompt type.

To record a prompt from within the Prompts Manager view:

1. Select an existing prompt row of type Resource.
2. Click the Record icon ● to open the Prompts Manager - Recorder dialog box as shown below:



Prompts Manager - Recorder

Record Audio: BronzeQueuePrompt_Prompt

Resource file : Resources/Prompts/en-US/Queue_Bronze_PFS.vox

Recording

My Notes : Effective 01/01/09.

Alternate Text : For Bronze customers

Location : Resources/Prompts/en-US

File Name : Queue_Bronze_PFS

Audio Format : VOX

Encoding : ULAW

Resources/Prompts/en-US/Queue_Bronze_PFS.vox

The Prompts Manager - Recorder dialog box assists in the recording, playback, and storing of the audio file.

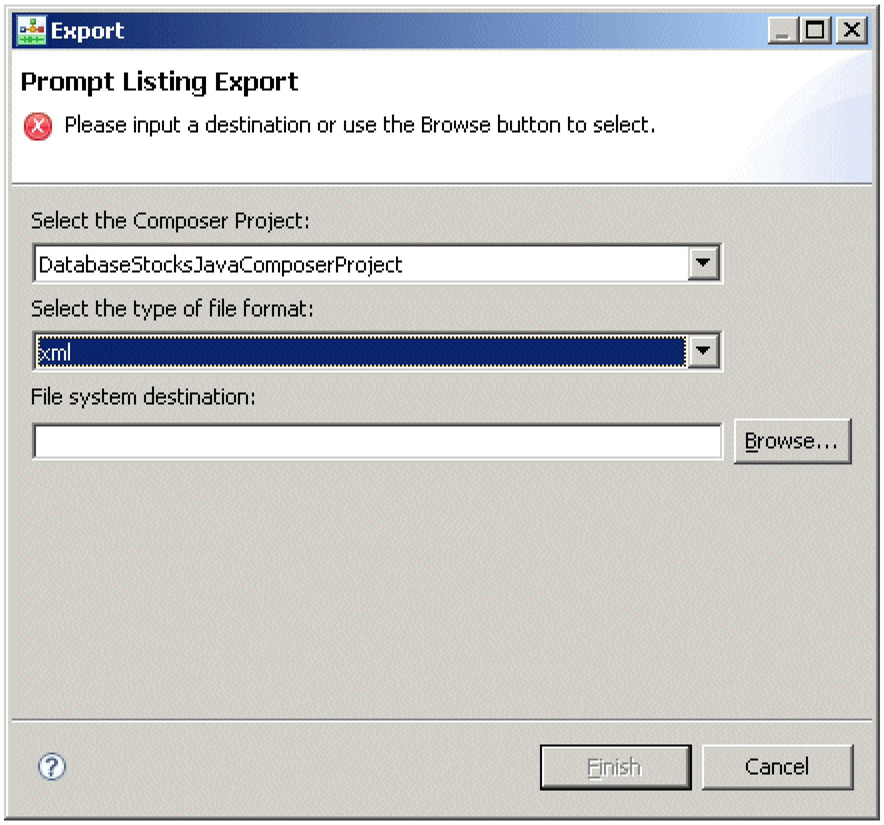
1. Type any notes for this prompt in the **My Notes** field.
2. Type an alternate text string in the **Alternate Text** field. This text is used to generate audio using #Text-to-Speech in place of the audio file should the audio file not be available at runtime.
3. Select the default recording location, or click **Browse** to navigate to an alternate location. Note: Genesys recommends that you keep your language specific audio files in the`/Resources/Prompts/`<language-code> folder.
4. Type a recording file name or keep the current name.
5. If the audio recording format displayed in the Recorder is not the format you wish to use, click **Audio Recording Format** to open the Properties dialog box for this Composer Project, from which you can change the audio format to **WAV** or **VOX**. You can also specify the encoding as **ALAW** or **MULAW**.

6. Click **OK** in the Properties dialog box to accept the new audio format setting for this Composer Project.
7. Click **Clear** if you want to clear the current resource file and create a new one.
8. Click the **Record** icon to record your audio prompt. A microphone should be connected and volume levels should already be set properly.
9. Click the **Stop** icon to stop the recording.
10. Click the **Play** icon to play back the new audio prompt. You can re-record if necessary.
11. Click **OK** when you are finished to close the Prompts Manager - Recorder dialog box. At this point, Prompt Manager will save any changes you have made. If you click Cancel, no changes are saved to the project.

Exporting a Prompt Listing

The Composer Prompts Manager provides the ability to export a prompt listing of all prompts in a Composer Project along with the attributes shown in Prompts Manager, such as instructions and notes. This facility is useful if you need to send your prompts out for professional recording and want to include instructions and text to be recorded along with prompt names. To export a prompt listing from within the Prompts Manager view:

1. Click the **Export Composer Project Prompts** icon in the Prompts Manager view, or
    - From the File menu, select **Export.** Expand **Composer** and select **Export Prompt Listing**, or
    - Right-click with any prompt or Prompt block selected, and select **Export Composer Project Prompts** from the context menu, to open the Export dialog box as shown below:


Export

Prompt Listing Export

Please input a destination or use the Browse button to select.

Select the Composer Project: DatabaseStocksJavaComposerProject

Select the type of file format: xml

File system destination:

2. Select the Composer Project whose prompts you wish to export from the drop-down list.

3. Select the file format for your exported data from the drop-down list. You may select either **xml** or **csv** format.

4. Click Browse to navigate to a destination location to hold your prompt export file. The exported file will have the name: <voiceprojectname>.xml or <voiceprojectname>.csv].

5. Click **Finish** to complete the export request.

XML Format Description Below is an example snippet from a prompt listing export in XML format: `<prompts project="JavaComposerProject_Voice_Business"> <prompt callflow="Main" block="WelcomePrompt" name="WelcomePrompt_Prompt1" type="Resource" interpret-as="Audio" value="Resources/Prompts/en-US/Brand_A.vox" format="" alternateText="" instructions="" notes="" /> … </prompts>`

XML Tag	Attribute Name	Description
<prompt>	project	The Composer project that is being exported.
<prompt>	callflow	The name of the callflow diagram where the prompt resides.
<prompt>	block	The name of the diagram block where the prompt resides.

<prompt>	name	The name of the prompt item.
<prompt>	type	The type of prompt, such as Value, Resource, or Variable.
<prompt>	interpret-as	The interpretation of the prompt value.
<prompt>	value	The value of the prompt item.
<prompt>	format	If applicable, the format of the value. Used for interpret-as, Date or Time. For example, 24 Hour or 12 Hour.
<prompt>	alternatetext	The alternate text for the prompt. Used for an invalid value. For example, if an audio resource does not exist or the variable data is invalid.
<prompt>	instructions	Text for additional or specific information instructions. For example, if a certain word needs to be emphasized when recording at the recording studio.
<prompt>	notes	Any further notes from the user. For example, identify if an associated audio file was recorded by the Prompts Manager or if the audio file was from a recording studio. Shows the source, which will be set by the user (Recorded/Imported/Unknown).

CSV Format Description The CSV format separates each prompt-related value by commas. The ordered values represents the following:

1. **Callflow**
2. **Block Name**
3. **Prompt Type**
4. **Interpret-As**
5. **Prompt Name**
6. **Value**
7. **Format**
8. **Alternate Text**
9. **Instructions**
10. **Notes**

The following is a snippet from the prompt listing Export in CSV format:
```
Main,WelcomePrompt,Resource,Audio,WelcomePrompt_Prompt1,"Resources/Prompts/en-US/
Brand_A.vox","",,"",""
```

Prompt Listing Usage For a Recording Studio

A recording studio may use the details in the sample exported prompt listing below when preparing an audio recording for a prompt item. This transcript-like format is intended to assist with producing professional sounding recordings. The five prompt items are in sequenced order to provide a sense of tone in relation to where the recorded message is at the beginning/middle/end of the overall message. The recorder:

- Uses the block name attribute to determine which set of prompt items belong together.. For example, the last five prompt items are from the same Menu1 prompt block.
- Is typically interested in prompts where type="Resource" and interpret-as="Audio", as these are the audio resources that are to be professionally replaced.
- Uses the value from the alternateText attribute to determine what should be said for the recording.
- Uses the instructions attribute for additional details from the developer, such as instructions to emphasize a certain word in the prompt message.

Sample Exported Prompt Listing

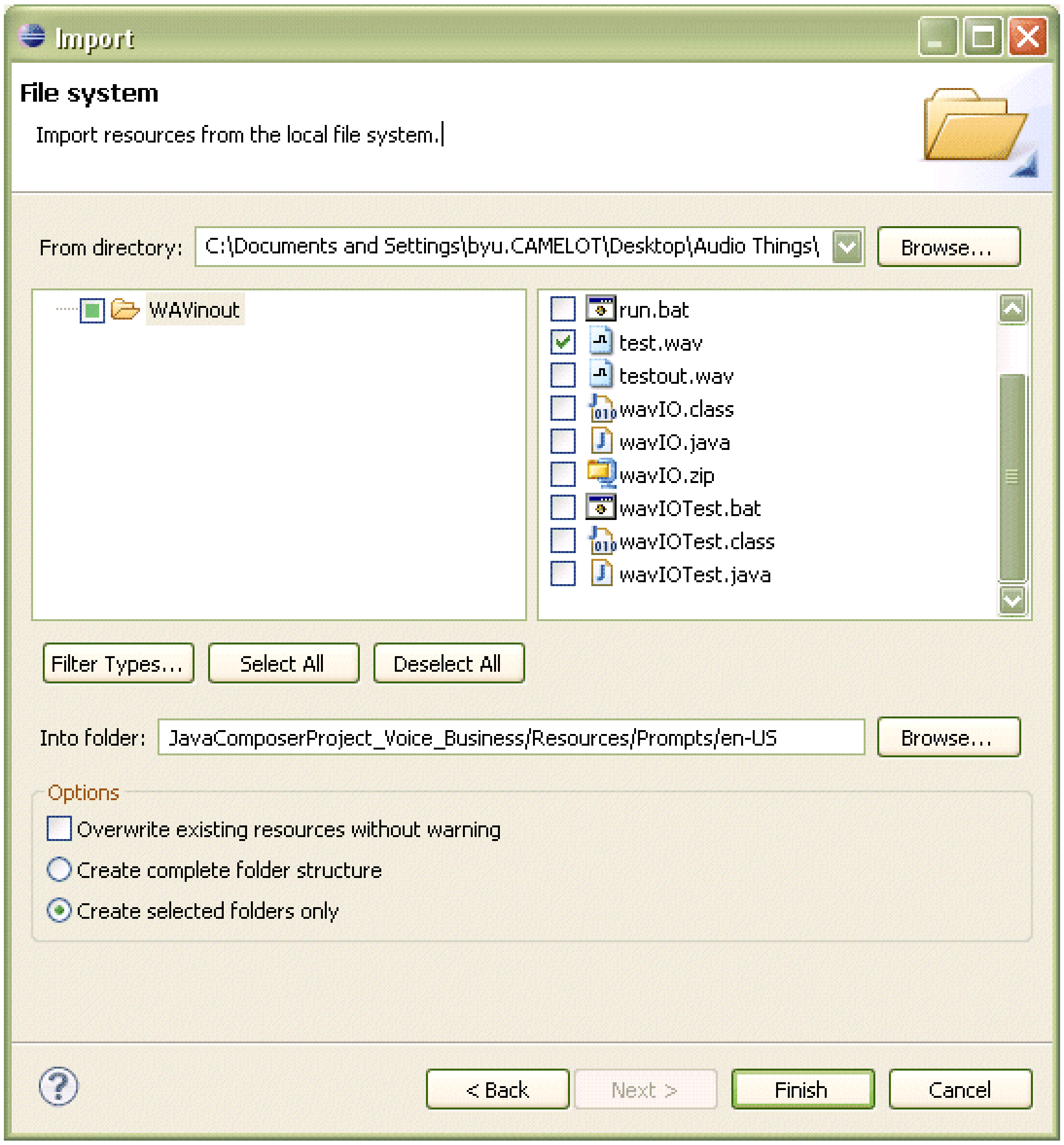
<prompts project="JavaComposerProject_Voice_Business"> <prompt callflow="CompanyABC" block="Prompt1" name="Prompt1_PromptMsg1" type="Resource" interpret-as="Audio" value="Resources/Prompts/en-US/Welcome.vox " format="" alternateText="Welcome to A B C bank." instructions="" notes="Prompts Manager recorded file." /> <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg1" type="Resource" interpret-as="Audio" value="Resources/Prompts/en-US/MainMenu_A.vox" format="" alternateText="Main menu." instructions="" notes="Default Composer audio file." /> <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg2" type="Resource" interpret-as="Audio" value="" format="" alternateText="To check your balance press one or say check balance." instructions="Place emphasis on 'check balance'" notes="" /> <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg3" type="Resource" interpret-as="Audio" value="" format="" alternateText="To make a bank to bank transfer press two or say transfer." instructions="Place emphasis on the word 'transfer'" notes="" /> <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg4" type="Resource" interpret-as="Audio" value="" format="" alternateText="To repeat these options press five or say repeat." instructions="Place emphasis on the word 'repeat'" notes="" /> <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg5" **Naming** For ease of importing the new audio recordings into Composer, Genesys recommends making the name the same as the attribute value of the respective prompt entry. For example, MainMenu_A.vox in the below snippet. This avoids having to rename the files when they are imported into Composer as described in the Importing Prompt Resources topic. <prompt callflow="CompanyABC" block="Menu1" name="Menu1_PromptMsg1" type="Resource" interpret-as="Audio" value="Resources/Prompts/en-US/MainMenu_A.vox" format="" alternateText="Main menu." instructions="" notes="Default Composer audio file." />

Importing Prompt Resources

See the Sample Exported Prompt Listing, which should be used as a transcript by the recording studio. After receiving the prompt audio resources from the professional audio recording studio, be sure to place the audio files in the correct Composer Project resource path. This ensures that the resources will work properly with the existing callflows that will use them. The Composer prompt resources are stored under the Composer Project folder ../Resources/Prompts. Example: ../Resources/Prompts/en-US/Brand_A.vox For a prompt item with audio resource Resources/Prompts/en-US/Brand_A.vox, the new professionally recorded audio file must be identically located and named Resources/Prompts/en-US/Brand_A.vox. If you do not do this, you must go to the callflow diagram block properties to set the new prompt resource path, or rename the file to match existing prompt settings. **Note**: When importing multilingual prompts, be sure to place the audio resource files in their corresponding prompt resource locale folder. For example,

- **English** -- United States ../Resources/Prompts/en-US
- **Spanish** -- Spain ../Resources/Prompts/es-ES

To import file resources to the target Composer Project, use the Project Explorer. Or simply copy and paste the files to the target prompts resource folder location of the Project Explorer. As an alternative, importing may be achieved by using File > Import… Expand and select **General** > **File System**. In the Import dialog, set the **Fro**m directory field and Into folder fields, select the desired files, and click **Finish**. A sample is shown below.


Import

File system

Import resources from the local file system.

From directory: C:\Documents and Settings\byu.CAMELOT\Desktop\Audio Things\

WAVinout

- [ ] run.bat
- [✓] test.wav
- [ ] testout.wav
- [ ] wavIO.class
- [ ] wavIO.java
- [ ] wavIO.zip
- [ ] wavIOTest.bat
- [ ] wavIOTest.class
- [ ] wavIOTest.java

Into folder: JavaComposerProject_Voice_Business/Resources/Prompts/en-US

Options

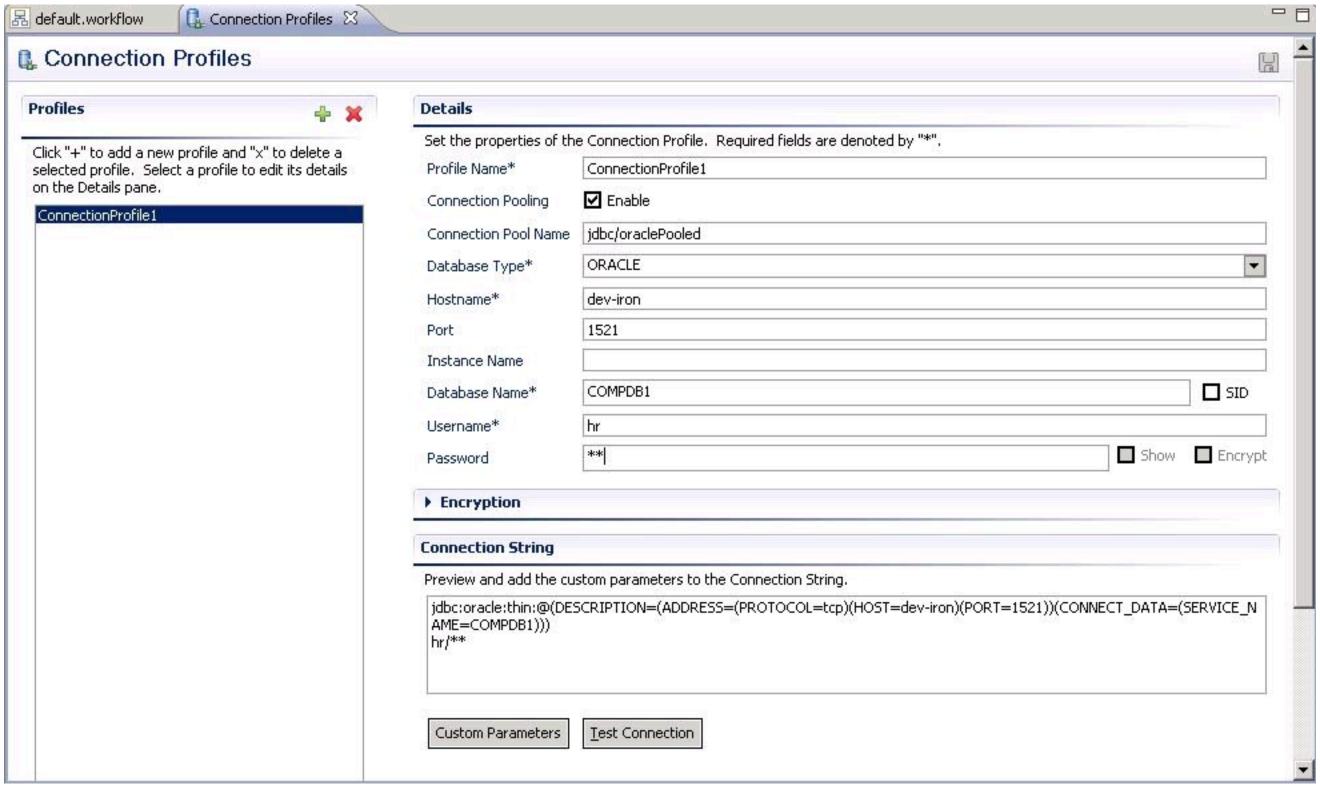
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Working with Database Blocks

This page contains general information on working with the Database blocks.

Database Connection Profiles

Before you can connect to a database in your application, you need to define a database connection profile that will maintain all information necessary to connect to a particular instance of a database. The figure below shows an example connection profile.



Click "+" to add a new profile and "x" to delete a selected profile. Select a profile to edit its details on the Details pane.

ConnectionProfile1

Set the properties of the Connection Profile. Required fields are denoted by "*".

Profile Name*	ConnectionProfile1
Connection Pooling	☑ Enable
Connection Pool Name	jdbc/oraclePooled
Database Type*	ORACLE
Hostname*	dev-iron
Port	1521
Instance Name	
Database Name*	COMPDB1 ☐ SID
Username*	hr
Password	** ☐ Show ☐ Encrypt

Preview and add the custom parameters to the Connection String.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=dev-iron)(PORT=1521))(CONNECT_DATA=(SERVICE_N
AME=COMPDB1)))
hr/**
```

The DB Data block requires that you specify the name of a connection profile in its properties so that it can use that information to connect to the database at runtime. Multiple connections profiles can be defined in one Project and these profiles can be shared by multiple DB Data blocks even if they are in different callflows. A connection profile consists of the basic information required to connect to a database. The information provided in a connection profile includes the following:

- **Profile Name**. The internal name that Composer uses to identify connections uniquely.
- **Connection Pooling**. Select to enable connection pooling, which maintains a set of database connections that can be reused for requests to databases. You can use this feature to enhance performance by avoiding time-consuming re-establishment of connections to databases.

- **Connection Pool Name**. Specify a Java Naming and Directory Interface (JNDI) name for the pooled data source. Composer applications can use any JNDI data source exposed by the web server. The .war files exported by Composer contain configuration files to support connection pooling with JBoss and WebSphere; other configuration changes to the web application may be required for other web servers
- **Database Type**. The type of database from the list of supported databases
- **Hostname**. The host on which the database server is running. In case of Database Cluster, Virtual IP/ Cluster Alias/SCAN Name is specified here.
- **Port**. The TCP port on which the database server is listening for connections. The most commonly used defaults for supported database types are pre-populated by Composer. If your database server uses custom ports, you will need to specify them here.
- **Instance Name**. The MSSQL Instance that need to connect in SQL Server. Port will take precedence if specified. This field is disabled when Database Type is selected as ORACLE.
- **Database Name**. The name of the database/catalog for SQLServer and the SID in case of Oracle.
- **SID**. The check box to specify if value provided in "Database Name" is SID. This check box is disabled when "Database Type" is MSSQL
- **Username**. The username that should be used to access the database
- **Password**. The password that should be used to access the database
- **Encrypt**. Select the encrypt the password.
- **Show**. Select to show the password
- **Custom Parameters**. The supported custom parameters can be included in connection string along with other parameters. To define custom parameters click on the button "Custom Parameters". In the dialog opened add the parameter name and value, in the order that need to be appended to connection string.

Configuration for Database Cluster:

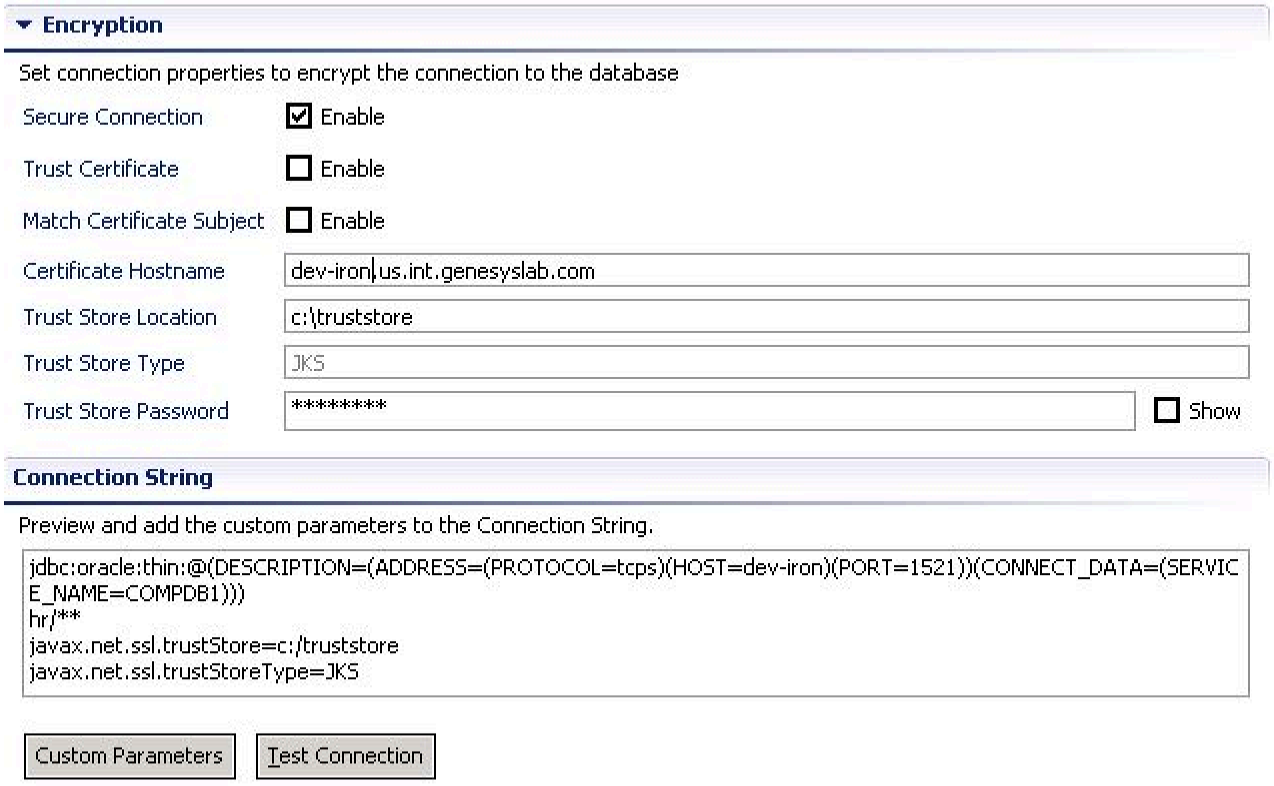
- For MSSQL Cluster, Virtual IP/Cluster Alias is specified in Hostname field of Connection Profile. To connect to particular named instance in cluster, Instance parameter is configured.
- For ORACLE Cluster, Cluster Alias/SCAN Name is specified in Hostname field of Connection Profile.

Additionally, to enable TAF functionality in ORACLE clusters, connection pool is created similar to pooling capability in other application servers. Connection pool can be created as the example below (This need to be added in Tomcat server.xml present in Composer installed path) <Resource name="jdbc/oraclePooled" auth="Container"

```
 type="com.mchange.v2.c3p0.ComboPooledDataSource"
factory="org.apache.naming.factory.BeanFactory"
driverClass="oracle.jdbc.driver.OracleDriver"
user="scott"
password="tiger" jdbcUrl="jdbc:oracle:oci:@(DESCRIPTION=(LOAD_BALANCE=on)(FAILOVER=on)
(ADDRESS=(PROTOCOL=tcp)(HOST=172.21.184.70)(PORT=1521))(ADDRESS=(PROTOCOL=tcp)
(HOST=172.21.184.71)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=rac.genesyslab.com)
(FAILOVER_MODE=(TYPE=session)(METHOD=basic))))" />
```

Encryption:

Parameters under "Encryption" tab allows you to configure SSL encryption and server authentication for Database connections made during Design time (Query Builder, Stored Procedure) and Runtime. When security is enabled, SSL encryption is used for all data sent between composer and SQLServer, if the SQL server has a certificate installed.


Encryption

Set connection properties to encrypt the connection to the database

Secure Connection	☑ Enable
Trust Certificate	☐ Enable
Match Certificate Subject	☐ Enable
Certificate Hostname	dev-iron.us.int.genesyslab.com
Trust Store Location	c:\truststore
Trust Store Type	JKS
Trust Store Password	******** ☐ Show

Connection String

Preview and add the custom parameters to the Connection String.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=dev-iron)(PORT=1521))(CONNECT_DATA=(SERVIC
E_NAME=COMPDB1)))
hr/**
javax.net.ssl.trustStore=c:/truststore
javax.net.ssl.trustStoreType=JKS
```

To establish a Secure Database connection from Composer, following parameters are to be configured under encryption tab:

- **Secure Connection**. Enabling this check box will make all connections from Composer to Database Server encrypted with a choice of server authentication
- **Trust Certificate**. Enabling "Secure Connection" and "Trust Certificate" will be sufficient to establish SSL Connection. When "Trust Certificate" is disabled, other optional attributes are enabled to validate server certificate,
- **Match Certificate Subject**. This is enabled in order to force the matching of the certificate subject available in Server Certificate and client's trusted copy.
- **Certificate Hostname**. This parameter is specified in case the client certificate carries a different subject name than the server certificate and user wishes to ignore the difference by providing the subject name expected in the server certificate explicitly.
- **Trust Store Location**. Location where the Trust Store file is present. The trust store file contains all the certificates trusted by the client, including the certificate that the server uses to autheticate itself.
- **Trust Store Type**. JKS truststore is supported when Database Type is ORACLE. This parameter is not editable. This is not applicable when Database Type is MSSQL
- **Trust Store Password**. Password to access the trust store.

Certificate configuration for Secure Connection:

- For Java Composer Projects, when "Secure Connection" is enabled and "Trust Certificate" is disabled, certificates are placed in "TrustStore Location" specified in connection profile.
- For .NET Composer Projects Design time (i.e. for Query Builder and Stored Procedure Builder), certificates are placed in "TrustStore Location" specified in connection profile.
- For .NET Composer Projects Runtime and MSSQL database, certificates are installed in "Certificate Windows Snap-In" accessed from MMC console in Windows.
- For .NET Composer Projects Runtime and ORACLE database, certificates are installed in Oracle wallet both in client and server. tnsnames.ora configuration will have service name with TCPS protocol. Example is given below.

SSLTEST =

```
  (DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCPS)(HOST = dev-rose.us.int.genesyslab.com)(PORT = 2484))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = SSLTEST)
  )
)
```

Notes:

To establish a connection profile, you must be working with a Project file that was upgraded to Composer 8.0.2 or higher from an earlier Composer release. Connection profiles are not available in Projects created using Composer 8.0. They become available after the Project is upgraded. The method for specifying additional pooling parameters varies based on the database being used and the Project type. Java Composer Projects use the c3p0 library for both SQLServer and Oracle databases. Otherwise, in the case of Oracle databases, Composer uses the c3p0 library and the library exposes its own configuration parameters for pooling via an XML file. In case of SQLServer, additional pooling parameters can be specified in the connection string.

Creating/Editing a Connection Profile

To create (or edit) a connection profile:

1. Select the Project for which you are creating a connection profile in the Project Explorer, and expand your project folder set.
2. Expand the db folder.
3. Double-click the connection.properties file. The Connection Profiles view opens.
4. To create a new profile, click the **Add Profile** ✚ icon in the Profiles pane. (If you wish to edit an existing profile, you can select an existing profile in the Profiles pane.)
5. In the Details pane, enter (or update) the appropriate information in each field (fields containing the *

character are required).

6. Click the **Save Profile** icon in the upper-right of the Connection Profiles window. You must save the profile in order for it to be available for selection in the Select Connection Profile dialog box.
7. Test the connection profile by clicking the **Test Connection** button to connect to the database.
   - The message `Database connection was successful` indicates your connection profile successfully connected to the intended database.
   - The message `Database connection failed` followed by additional details indicates a problem with your connection profile. Update the profile, save it, and test it again.

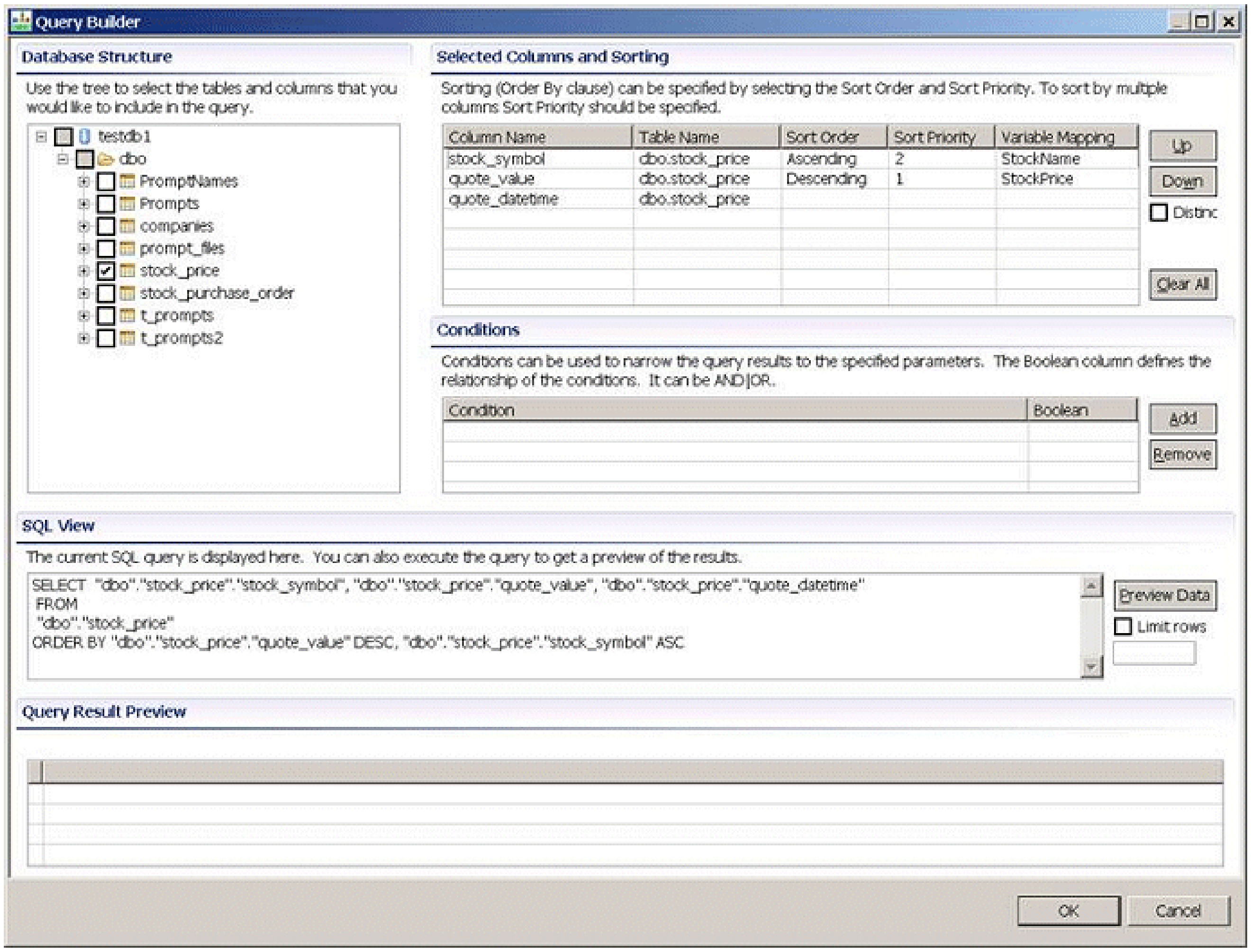
Note: For information on creating the configuration for the connection pool on the application server side, see Connection Pooling.

Preview Connection Strings

The connection to the database with the specified parameters in the connection profile can be previewed and tested in the Connection profile editor. In case of Java project as the design and runtime connections use JDBC connection , JDBC connection string is available to preview and test. In case of Dotnet projects as the design time uses JDBC connection and runtime uses OLEDB conneciton, both strings are available to preview and test. **Note:** The Dotnet project must be deployed correctly in IIS to preview the OLEDB connection string. The parameters apart from ones explicitly collected in the editor can be added using the **custom parameters** dialog which takes the parameters as a name value pair.

Using the Query Builder

The Composer Query Builder provides a visual method of building a database query without the need to type SQL code. The Query Builder is accessed through the Query Type property in the DB Data block. It can be used for both voice callflows and routing workflows. **Note:** The Query Builder can only be accessed when a valid connection profile has been created and selected in the Connection Profile property of the DB Data block. The Query Builder with an example query is shown below.



Use the tree to select the tables and columns that you would like to include in the query.

Sorting (Order By clause) can be specified by selecting the Sort Order and Sort Priority. To sort by multiple columns Sort Priority should be specified.

Column Name	Table Name	Sort Order	Sort Priority	Variable Mapping
stock_symbol	dbo.stock_price	Ascending	2	StockName
quote_value	dbo.stock_price	Descending	1	StockPrice
quote_datetime	dbo.stock_price			

Conditions can be used to narrow the query results to the specified parameters. The Boolean column defines the relationship of the conditions. It can be AND|OR.

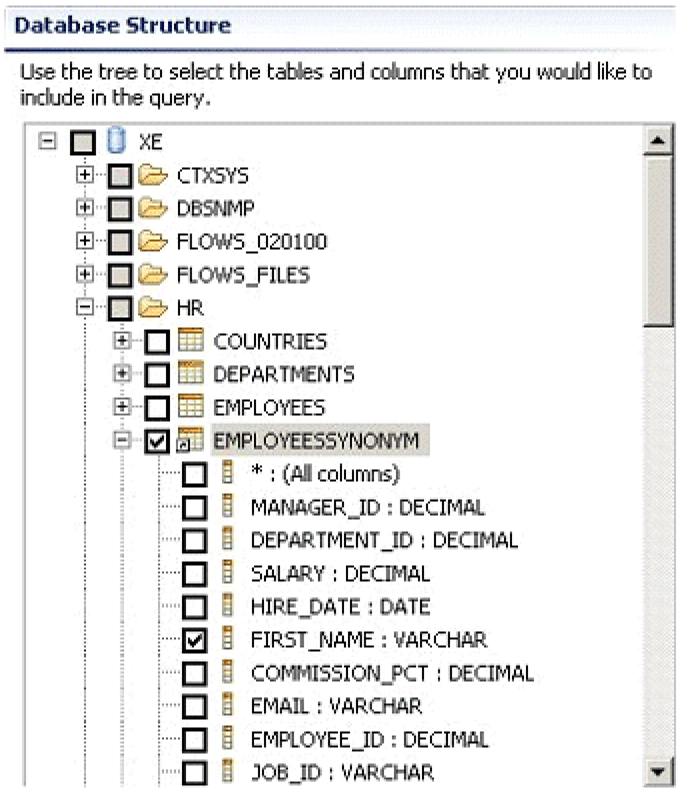
Condition	Boolean

The current SQL query is displayed here. You can also execute the query to get a preview of the results.

```
SELECT  "dbo"."stock_price"."stock_symbol", "dbo"."stock_price"."quote_value", "dbo"."stock_price"."quote_datetime"
 FROM
 "dbo"."stock_price"
ORDER BY "dbo"."stock_price"."quote_value" DESC, "dbo"."stock_price"."stock_symbol" ASC
```

Building a Database Query

The Query Builder opens when Composer is successfully able to connect to the database specified in your connection profile. Any schemas, tables (and table synonyms) and columns of the database accessible from the specified user account are shown in hierarchical format in the Database Structure pane of the Query Builder. In the example below, EMPLOYEESSYNONYM is a table synonym.


Database Structure

Use the tree to select the tables and columns that you would like to include in the query.

TableSyn.gif

Note: MSSQLServer table synonyms are read from the system table sys.synonyms. Oracle table synonyms are read from the system table user_synonyms. To build a query:

1. Specify which table columns are returned as query results.

    - Select the tables and columns to include in your query by checking appropriate items in the **Database Structure** pane. Expand table entries to see the columns. To select all columns in a table, select the appropriate (**All columns**) check box under the appropriate table.
    - Selected columns and tables appear in the **Selected Columns** pane. To alter the order in which selected columns are returned in query results, use the **Up** and **Down** buttons to reorder columns within the list.
    - To specify the order in which query results should be sorted, click on the **Sort Order** field for a column and select a Sort option (**ascending** or **descending**). This will automatically fill in the Sort Order, which indicates the sequence in which multiple sort criteria will be applied. It is possible to sort by multiple columns and you can change the sorting sequence by clicking on the **Sort Priority** column. For example, you might sort a query of names by last name and then sort by first name for those people with the same last name. In that case, last name has Sort Order 1, and first name has Sort Order 2.

Note: The order in which columns appear in the Selected Columns list does not affect the sort order.

- To specify the variables into which the column values need to be copied, click on the **Variable Mapping** field for a column and select a variable. If a variable is specified for a column, DB Data block execution will result in the column values of the first record being copied into the specified variable. If more than one record is returned by the query, then use the Looping block along with the DB Data block to iterate over records and populate the variables specified for the columns.

2. Specify filter criteria. In the **Conditions** pane, you build the search or filter criteria to identify the data you want to retrieve from the database. You can can specify multiple conditions.
    - Click **Add** to create a new condition. A new row will be added to the Conditions list. Click on the **Condition** column, and then click the ... to open the Condition Builder.
    - Select a column from the **Select Column** drop-down list which the search condition will operate on.
    - Select the operator (=, <>, <, >, and so on) from the **Operator** drop-down list. This operator will be used to compare the specified column with the value specified in the next step.</li>
    - In the **Value** field, type or select your value for the condition depending on the value type option:
        - **Column Reference**: a table column that you can select from a drop-down list. This option will compare the two selected columns based on the specified operator.
        - **Application Variable**: a variable defined in your application that can be selected from a drop-down list. At runtime the current value of the selected variable will be used for comparing the column's value based on the specified operator.
        - **Custom Value**: a value that is not validated by the query builder and is added directly to the query. It can be used to specify SQL functions or more complex expression.
        - **Literal**: a value that is interpreted as a string or a number. Type in the literal value. The value will be enclosed in quotes automatically if it is a string. If the literal value represents a number, you will need to enclose it in quotes depending on the data type of the selected column. This option will compare the selected column's value to the specified literal using the specified operator.
    - Click **OK** to complete the condition.
    - Using the above steps, you can define multiple conditions. These conditions can be combined using logical operators to further refine your search criteria. You can select **AND** or **OR** in the **Boolean** field to specify the logical operator.
3. Test your query.
    - To test the query, you can click the **Preview Data** button. This executes the query against the appropriate database. If the database tables contain data and if any records match the specified conditions, they will be displayed in the Query Results Preview pane. A message will also show the number of records returned as a result of the query.
    - If you expect that the number of matching records will be large and want to preview a subset of returned data, click the **Limit Rows** check box and enter a numeric value to limit the number of returned results.

Note: The message will now show the number of records displayed rather than the actual number of matching records. The query results preview is shown in the Query Result pane.

4. Click **OK** to save your query and update the DB Data block with the new query. If you click Cancel, all changes are discarded and no changes are made to the DB Data block.

Specifying Custom Queries

The DB Data block can use queries specified in a SQL (.sql) file in your Project instead of a query created using the Query Builder. To use a custom query:

- Create a .sql file in your db folder and specify the filename in the Query File property of the DB Data block. Make sure that the operation type is **SQLScriptFile**. Composer will read this file at runtime and use it to query the specified database.

The ability to use custom queries is useful in cases where the SQL query is already created using other tools, or if the query uses features not supported by the Visual Query Builder. The next topic describes limitations of the query builder.

Application Variables

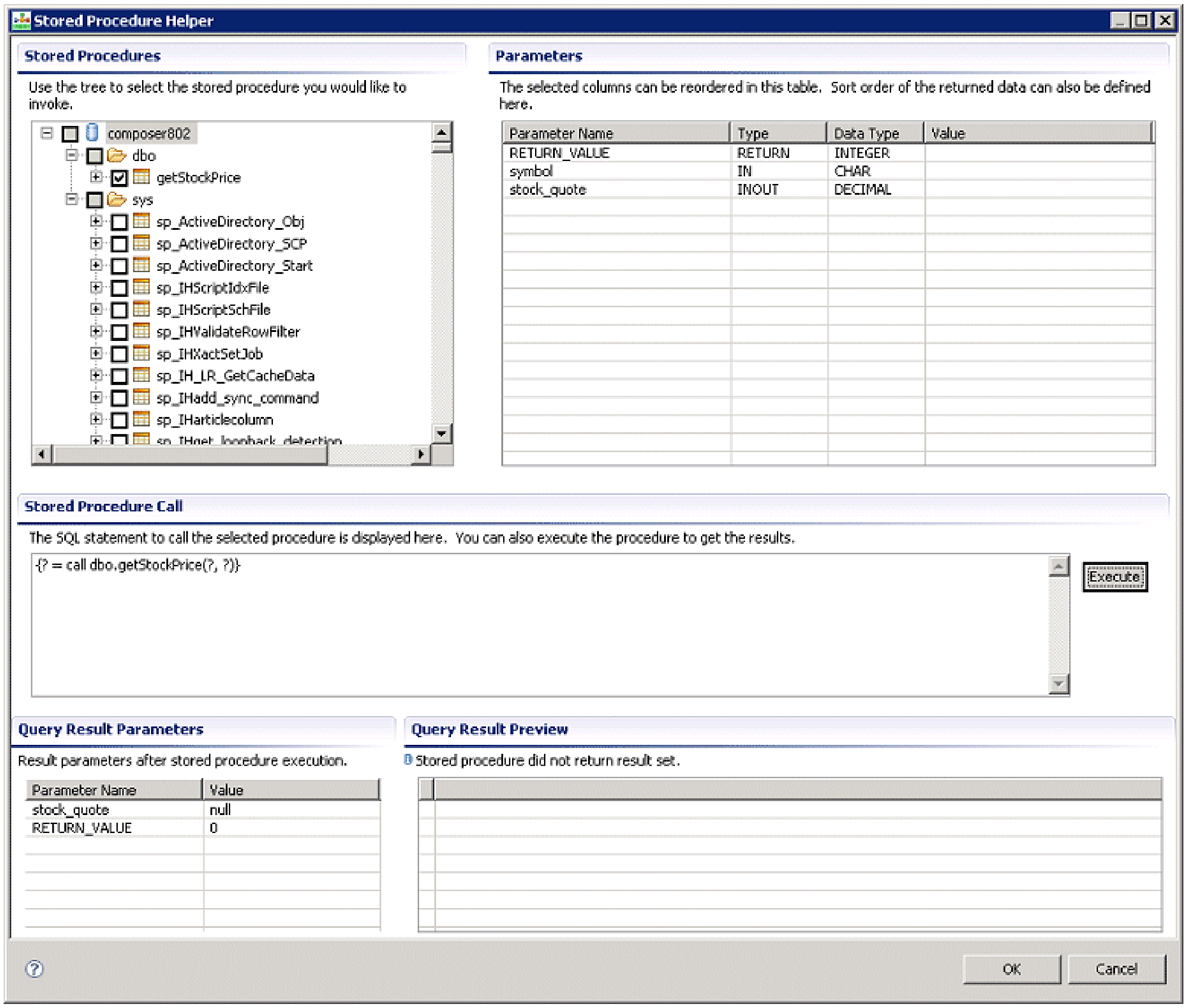
You can use Application variables in custom query files as part of the SQL statement. To use a variable, include its name within curly braces without the AppState. prefix. For example, the following statement uses varname1 and varname2. Their values will be substituted at the time the DB Data block queries the database. `SELECT name_of_function({varname1}, {varname2}) from dual` Results of the query are stored in a variable as a two-dimensional JSON array. This data can then be accessed via a Looping block or via scripting in the Assign or ECMAScript block. For example, if the database result set looks like this in tabular form:

Vegetables	Animals
lettuce	chicken
broccoli	lion

The JSON for the result will look like this: `{"db_result":[["lettuce", "chicken"], ["broccoli", "lion"]],"db_result_columns":["vegetables", "animals"]}` **Note:** An example of custom queries is in the Database Stocks Template application.

Stored Procedure Helper

If you select **StoredProcedure** for the Query Type property in the DB Data Block, you can click the ⋯ button on the property row to open the Stored Procedure Helper dialog box. Here you can select a stored procedure, execute it, and get query results. A completed example is shown below.



Stored Procedure Helper

Stored Procedures

Use the tree to select the stored procedure you would like to invoke.

```
composer802
  dbo
    [✓] getStockPrice
  sys
    sp_ActiveDirectory_Obj
    sp_ActiveDirectory_SCP
    sp_ActiveDirectory_Start
    sp_IHScriptIdxFile
    sp_IHScriptSchFile
    sp_IHValidateRowFilter
    sp_IHXactSetJob
    sp_IH_LR_GetCacheData
    sp_IHadd_sync_command
    sp_IHarticlecolumn
    sp_IHget_loopback_detection
```

Parameters

The selected columns can be reordered in this table. Sort order of the returned data can also be defined here.

Parameter Name	Type	Data Type	Value
RETURN_VALUE	RETURN	INTEGER	
symbol	IN	CHAR	
stock_quote	INOUT	DECIMAL	

Stored Procedure Call

The SQL statement to call the selected procedure is displayed here. You can also execute the procedure to get the results.

```
{? = call dbo.getStockPrice(?, ?)}
```

Query Result Parameters

Result parameters after stored procedure execution.

Parameter Name	Value
stock_quote	null
RETURN_VALUE	0

Query Result Preview

Stored procedure did not return result set.

Setting up a Stored Procedure Call

The Stored Procedure Helper opens when Composer is successfully able to connect to the database specified in your connection profile. Any stored procedures in the database accessible from the specified user account are shown in hierarchical format in the Database Structure pane of the Stored Procedure Helper.   To set up a stored procedure call:

1. Specify which stored procedure should be executed.
2. Select the stored procedure to execute by checking appropriate item in the Database Structure pane.
3. Parameters and Return Value appear in the Parameters pane. Specify the value (application variable) for each of the parameter into which the output value is stored after the stored procedure has executed.

4. To test the stored procedure, click the **Execute** button. This executes the stored procedure in the appropriate database. If the stored procedure returns any records, they are displayed in the Query Results Preview pane. Any output values are displayed in the Query Result Parameters pane. A message shows the number of records returned as a result of the query.
5. Click **OK** to save your query and update the DB Data block with the new query. If you click Cancel, all changes are discarded and no changes are made to the DB Data block.

Note: Composer does not support the REF CURSOR return type in a stored procedure.

Password Encryption

Composer can now encrypt the database connection profile passwords so that they are not written in the clear to the connection.properties file.

Encryption Key

In order to enable encryption, you must first create an encryption key. Composer requires a 128-bit (16 bytes) key, in hex-encoded format. This can be randomly generated by the OpenSSL tool, using the following command line:

```
$ openssl rand -hex 16 75b8ec9a3ce60a21c4f94236a1b55fb2
```

Any random source will do. Another example is `http://www.random.org/cgi-bin/randbyte?nbytes=16&format=h` (With this example, you will have to remove the spaces in the output.)

Save the encryption key to a text file. Note that this file should be securely stored, so that it can only be read by the Composer process and the backend Tomcat/IIS processes.

Configuring Composer Preferences

In the **Composer** > **Security** preference page, set the Encryption Key Location preference to point to the encryption key file created in the previous step.

Encrypting the Database Connection Profile Password

In the Connection Profile Editor, next to the Password field, enable the **Encrypt** checkbox. Now, when you save the Connection Profile, the password will be scrambled in the connection.properties file.

Enabling Decryption in the Backend

When the application runs, the application server will need to be able to decrypt the password so that it can connect to the database. For this, the application needs to be configured with the location of the encryption key file.

Java Composer Projects

If it doesn't already exist, create the file WEB-INF/composer.properties inside the project. Inside the file, enter the following line:

```
composerEncryptionKey=C:\\secrets\\encryption-key.txt
```

(Note that the backslashes here must be escaped.)

.NET Composer Projects

Edit the web.config file's appSettings entry:

```
<appSettings>

   <add key="composerEncryptionKey" value="C:\secrets\encryption-key.txt" />
 ...


</appSettings>
```

(Backslashes here are fine.)

Limitations and Workarounds

The Query Builder supports creating SELECT statements. The following is a list of limitations along with suggested workarounds:

- INSERT, UPDATE, and DELETE statements cannot be created using the Query Builder. Advanced SQL features, such as outer joins, subqueries, and unions are also not supported. A custom query can be used to overcome these limitations.
- if you rename a DB Data block, its corresponding SQL statement file in the db folder will not be updated and will not be valid until you generate code again.
- For details on SQL datatypes supported by Composer, see Supported SQL Datatypes.

Oracle Client Setup for IIS

To set up an Oracle client for Internet Information Services:

1. Install the Oracle client components on the application server.
2. Create a `tnsnames.ora` file in the `C:\oracle\ora81\network\ADMIN` folder where C:\oracle is the installation folder of Oracle client components.
3. Add the following lines to `tnsnames.ora` where COMPDB1 is any alias of choice, XYZ is the Oracle server, COMPOSER is the **Service Name** as configured on the Oracle listener (server). After doing this, you should be able to connect to Oracle using `sqlplus user/pwd@COMPDB1` as the command at the command prompt.

```
COMPDB1 =   (DESCRIPTION =     (ADDRESS_LIST =       (ADDRESS = (PROTOCOL = TCP)(HOST =
XYZ.us.int.genesyslab.com)(PORT = 1521))     )     (CONNECT_DATA =       (SERVICE_NAME =
COMPOSER)     )   )
```

4. Create a System DSN using the Data Sources (ODBC) under **Administrative Tools**.
5. Make sure that **Data Source Name** specified above is exactly same as the **Database Name** specified in the Composer database connection profile and **TNS Service Name** is the same as the alias in step 3.
6. Click on **Test Connection** in the database connection profile. The connection should be successful and the Composer VXML application should be able to connect to the database.

Steps 4, 5 and 6 can be avoided if the alias used in the `tnsnames.ora` file is same as the database name specified in Composer.

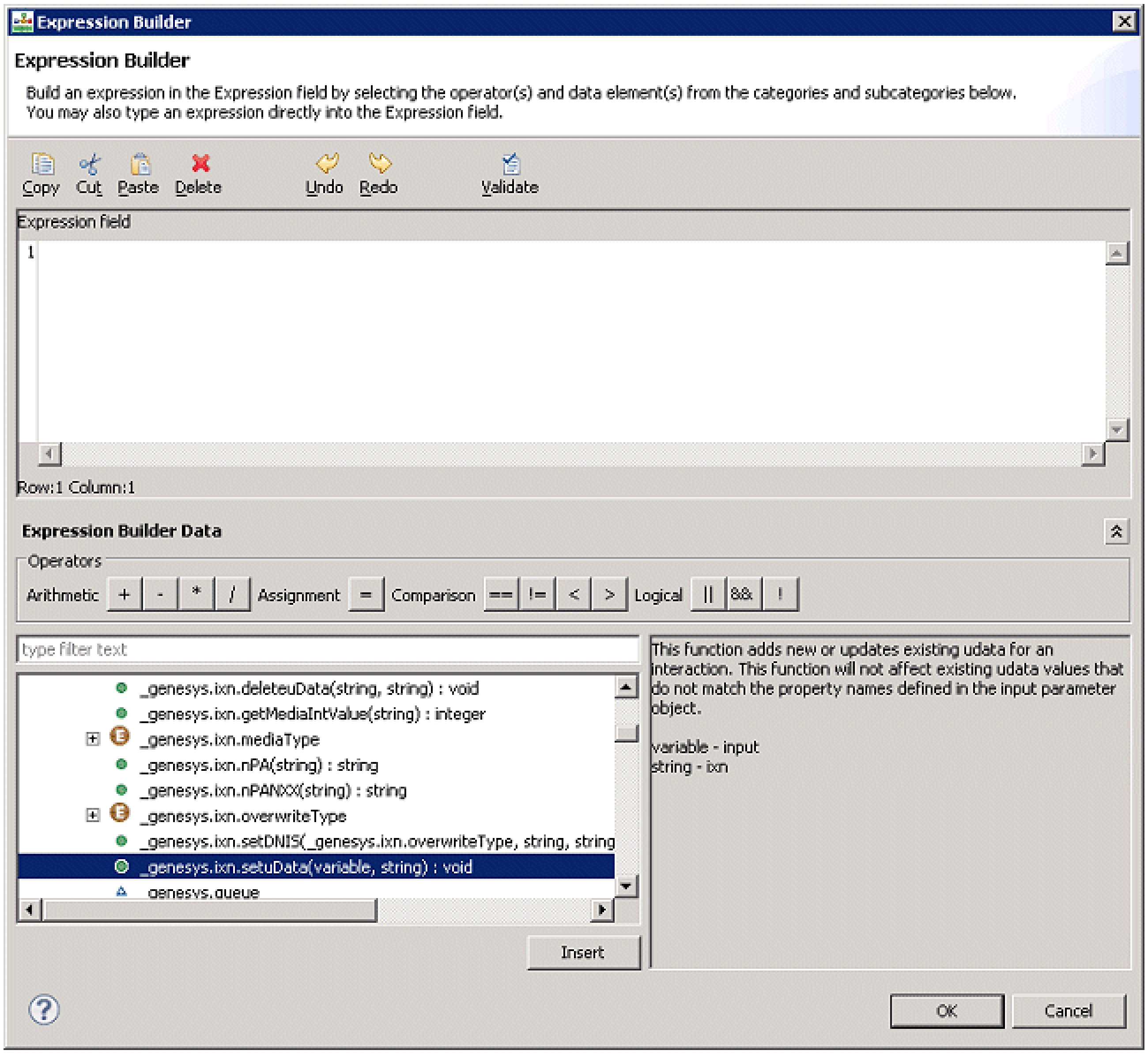
User Data

To work with User Data in Composer, you can use:

- The Interaction Data block for voice applications.
- The User Data property of the External Service block if you wish to pass User Data to an external service (routing and voice).
- The Entry block to access User Data (routing and voice).

For routing applications, you can use:

- The User Data block.
- The Create Email block, which lets you pick up standard response text from User Data.
- The Create SMS block, which lets you pick up message text from User Data.
- The Identify Contact block, which provides an option to update the interaction’s User Data with the parameters returned by Universal Contact Server (Contact attribute data).
- The Create Interaction block, which lets you create a new interaction record in UCS database based on User Data.
- The ECMAScript block. The Script property lets you use Universal Routing Server User Data functions. Open Expression Builder. Select URS Functions and then _genesys.ixn.deleteuData (to add a User Data property or delete all properties) or _genesys.ixn.setuData (to add new or update existing User Data).


Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below. You may also type an expression directly into the Expression field.

_genesys.ixn.deleteuData(string, string) : void
_genesys.ixn.getMediaIntValue(string) : integer
_genesys.ixn.mediaType
_genesys.ixn.nPA(string) : string
_genesys.ixn.nPANXX(string) : string
_genesys.ixn.overwriteType
_genesys.ixn.setDNIS(_genesys.ixn.overwriteType, string, string
_genesys.ixn.setuData(variable, string) : void
_genesys.queue

This function adds new or updates existing udata for an interaction. This function will not affect existing udata values that do not match the property names defined in the input parameter object.

variable - input
string - ixn

Hints

- A specific variable ‘xyz’ can be accessed directly; for example: _genesys.ixn.interactions[0].udata.xyz
- To write to User Data, use the setuData() function in an ECMAScript snippet. Usage is similar to the example below.

```
var input = new Object();

input.xyz = InputValue1; // Specify a value for the key ‘xyz’.

input[‘my-key-nname’] = ‘value’; // Use this notation if the key or property name has
a hyphen in it. Note that‘my-key-nname’has hyphens.
```

```
_genesys.ixn.setuData(input);
```

- Reading User Data is easier using the Assign block than with the ECMAScript block.

Mandatory Data for UCS Blocks

When working with the Update Contact and Render Message blocks (which map to Universal Contact Server services), certain properties must exist in the interaction User Data.

For the Update Contact block, ContactId must exist.

For the Render Message block, ContactId (if some contact-related Field Codes (as described in the *eServices 8.1 User*'*s Guide*) are used in the text to render). Also InteractionId (if some interaction-related Field Codes are used in the text to render)and OwnerEmployeeId (if some agent-related Field Codes are used in the text to render).

As is the case with IRD, these properties are not set in the blocks themselves. Instead, the properties are assumed to be put in the interaction's User Data by some other block earlier in the workflow, such as the Identify Contact block or Create Interaction block with the Update User Data property set to true. In case no other block does this, the User Data block may be used for this purpose.

If these properties are not available, an explicit UCS error message (missing parameter) shows in the Orchestration Server log.

Callflow User Data

Also see the following blocks used for callflows:

- Interaction Data
- Route Request

Connection Pooling

When defining a database connection profile, you can use connection pooling, which maintains a set of database connections that can be reused for requests to databases. This feature can enhance performance by avoiding time-consuming re-establishment of connections to databases.   While Composer does not support specific application servers, this topic presents information on configuring Tomcat, JBoss, and Websphere application servers to expose a pooled data source as a JNDI resource. This topic also contains information on creating a JDBC provider for an Oracle database.

Connection Pooling for Tomcat Application Servers

For Tomcat, a JNDI resource is defined in a Context configuration. Do this in the global scope, at `$TOMCAT_HOME/conf/context.xml`. Here is a sample:

```
<Context> ...          <Resource
name="jdbc/pooledDS" auth="Container"
        type="com.mchange.v2.c3p0.ComboPooledDataSource"
        factory="org.apache.naming.factory.BeanFactory"
        driverClass="com.microsoft.sqlserver.jdbc.SQLServerDriver"
        user="john"        password="doe123"
        jdbcUrl="jdbc:sqlserver://dbserver1:1433;databaseName=composer1" />
     <Resource name="jdbc/oraclePooled" auth="Container"
        type="com.mchange.v2.c3p0.ComboPooledDataSource"
        factory="org.apache.naming.factory.BeanFactory"
        driverClass="oracle.jdbc.driver.OracleDriver"        user="jane"
        password="doe456"
        jdbcUrl="jdbc:oracle:thin:@dbserver2:1521:composer2" /> ... </Context>
```

Important Items

- `name`--should match the Connection Pool Name parameter given in the Connection Profile in Composer.
- `user`, `password`--these are the login credentials to the database.
- `jdbcUrl`--specifies the host, port and database name. Can be copied from the Connection Profile editor in Composer. The JDBC URL can also use advanced options that might not be otherwise exposed by Composer. For example, to enable Transparent Application Failover for a connection to an Oracle database, the URL can be given as:

```
jdbcUrl="jdbc:oracle:oci:@(DESCRIPTION=(LOAD_BALANCE=on)(FAILOVER=on)(ADDRESS=(PROTOCOL=tcp)(HOS
(ADDRESS=(PROTOCOL=tcp)(HOST=host2)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=dbcluster)
(FAILOVER_MODE=(TYPE=session)(METHOD=basic))))"
```

Additional Pooling Parameters

Additional pooling parameters can be customized here as well, for example:

```
<Resource name="jdbc/
pooledDS" auth="Container"        type="com.mchange.v2.c3p0.ComboPooledDataSource"
        factory="org.apache.naming.factory.BeanFactory"
        driverClass="com.microsoft.sqlserver.jdbc.SQLServerDriver"
        user="john"         password="doe123"
        jdbcUrl="jdbc:sqlserver://dbserver1:1433;databaseName=composer1"
        maxPoolSize="20"         acquireRetryAttempts="0" /
```

For a full list of available settings, refer to the c3p0 documentation, which is the third-party connection pooling library used by Composer (http://www.mchange.com/projects/c3p0/index.html).

Connection Pooling for JBoss Application Servers

To define connection pooling for JBoss:

1. Add the c3p0 and JDBC driver JARs to JBoss's `global lib` directory (`$JBOSS_HOME/server/<instance>/lib`). This is because JBoss will initialize the connection pool upon startup regardless of what applications are deployed. This is in contrast to Tomcat, which creates the connections on demand.
2. Next, define the JNDI resources in a file called `c3p0-service.xml`. Copy the file into `$JBOSS_HOME/server/<instance>/deploy`.

Sample:

```
 <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE server> <server> <mbean
code="com.mchange.v2.c3p0.jboss.C3P0PooledDataSource"
name="jboss:server=SQLServerDS">      <attribute name="JndiName">java:jdbc/
pooledDS</attribute>      <attribute
name="JdbcUrl">jdbc:sqlserver://dbserver1:1433;databaseName=composer1</attribute>
     <attribute
name="DriverClass">com.microsoft.sqlserver.jdbc.SQLServerDriver</attribute>
     <attribute name="User">john</attribute>      <attribute
name="Password">doe123</attribute> </mbean> <mbean
code="com.mchange.v2.c3p0.jboss.C3P0PooledDataSource" name="jboss:server=OracleDS">
     <attribute name="JndiName">java:jdbc/oraclePooled</attribute>      <attribute
name="JdbcUrl">jdbc:oracle:thin:@dbserver2:1521:Composer2</attribute>      <attribute
name="DriverClass">oracle.jdbc.driver.OracleDriver</attribute>      <attribute
name="User">jane</attribute>      <attribute name="Password">doe456</attribute>
</mbean> </server>
```

Pooling Parameters

Specify pooling parameters are specified by adding more <attribute> elements, e.g.,

```
<mbean
code="com.mchange.v2.c3p0.jboss.C3P0PooledDataSource" name="jboss:server=OracleDS">
     <attribute name="JndiName">java:jdbc/oraclePooled</attribute>      <attribute
name="JdbcUrl">jdbc:oracle:thin:@dev dbserver2:1521:Composer2</attribute>
     <attribute name="DriverClass">oracle.jdbc.driver.OracleDriver</attribute>
     <attribute name="User">jane</attribute>      <attribute
name="Password">doe456</attribute>       <!-- note that the attribute names must be
capitalized -->      <attribute name="MaxPoolSize">20</attribute>      <attribute
name="AcquireRetryAttempts">0</attribute> </mbean>
```

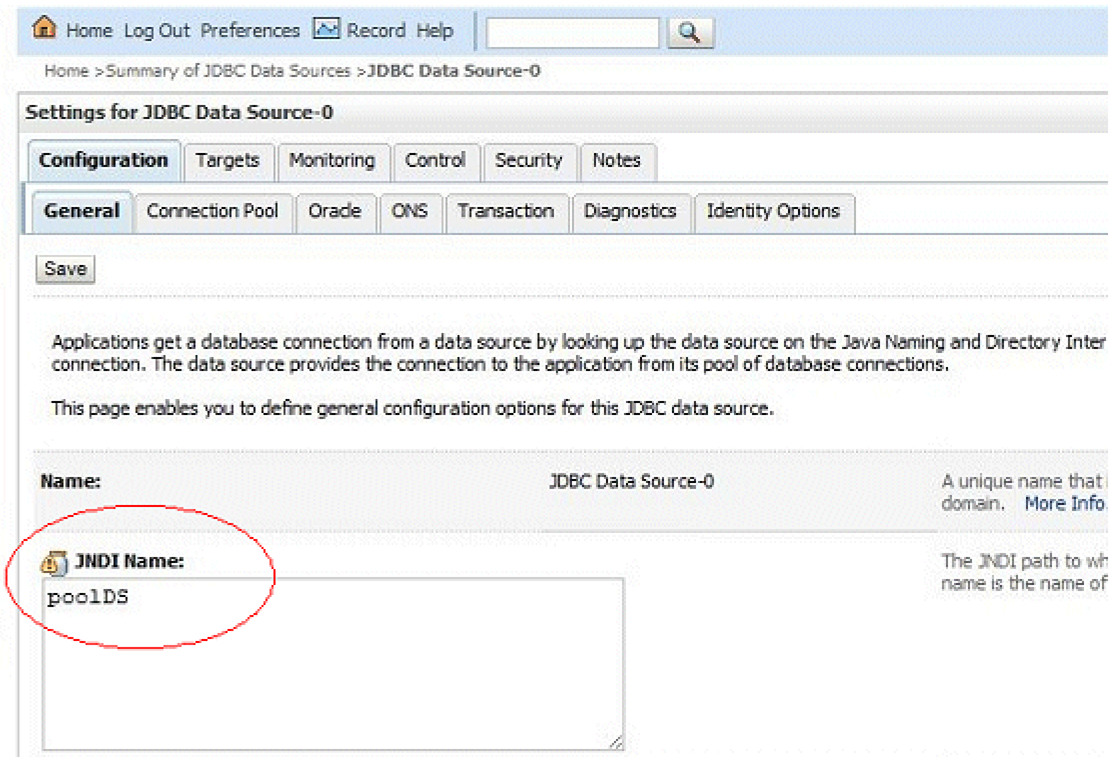
For a full list of available settings, refer to the c3p0 documentation, which is the third-party connection pooling library used by Composer (http://www.mchange.com/projects/c3p0/index.html).

Configuration Files

The following configuration files are automatically generated by Composer's WAR export functionality and do not require any user action: web.xml and jboss-web.xml **web.xml** In the web application itself, the deployment descriptor (`WEB-INF/web.xml`) needs to specify a resource reference: `<resource-ref>            <res-ref-name>jdbc/pooledDS</res-ref-name>            <res-type>javax.sql.DataSource</res-type>            <res-auth>Container</res-auth> </resource-ref>` **jboss-web.xml** This special JBoss-specific configuration file (WEB-INF/jboss-web.xml) is required to map the resource-ref to the globally defined resource. `<?xml version="1.0" encoding="UTF-8"?>` `<jboss-web>      <resource-ref>            <res-ref-name>jdbc/pooledDS</res-ref-name>            <res-type>javax.sql.DataSource</res-type>         <jndi-name>java:jdbc/pooledDS</jndi-name>      </resource-ref>` `</jboss-web>`

Connection Pooling for WebLogic Application Servers

When the application Server is WebLogic, there must be an extra configuration file in WEB-INF called `weblogic.xml`. First, though, confirm that the following is present in `web.xml` in the exported .war file: `<resource-ref>                  <res-ref-name>jdbc/poolDS</res-ref-name>                  <res-type>javax.sql.DataSource</res-type>                  <res-auth>Container</res-auth>` `</resource-ref>` `res-ref-name` should match the pool name in the connection.properties file, and it should be prefixed by `jdbc/`. **weblogic.xml File** The `weblogic.xml` can be added to the Composer Project in WEB-INF. Afterwards, you will have to export the .war file from Composer again and redeploy. The `weblogic.xml` should contain: `<?xml version="1.0" encoding="UTF-8"?>` `<wls:weblogic-web-app xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app"; xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"; xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd http://xmlns.oracle.com/weblogic/weblogic-web-app http://xmlns.oracle.com/weblogic/weblogic-web-app/1.2/weblogic-web-app.xsd";>   <wls:resource-description>   <wls:res-ref-name>jdbc/poolDS</wls:res-ref-name>   <wls:jndi-name>poolDS</wls:jndi-name>   </wls:resource-description>` `</wls:weblogic-web-app>` Note the following: The `wls:res-ref-name` should match `res-ref-name` in `web.xml`. `wls:jndi-name` should be the JNDI Name in the WebLogic configuration.



Settings for JDBC Data Source-0

Applications get a database connection from a data source by looking up the data source on the Java Naming and Directory Inter
connection. The data source provides the connection to the application from its pool of database connections.

This page enables you to define general configuration options for this JDBC data source.

Name: JDBC Data Source-0 A unique name that
domain. More Info

JNDI Name: poolDS The JNDI path to wh
name is the name of

.

Connection Pooling for WebSphere Application Servers

WebSphere has its own connection pooling capabilities, so you won't be using c3p0. The data sources are defined in the WebSphere management console.

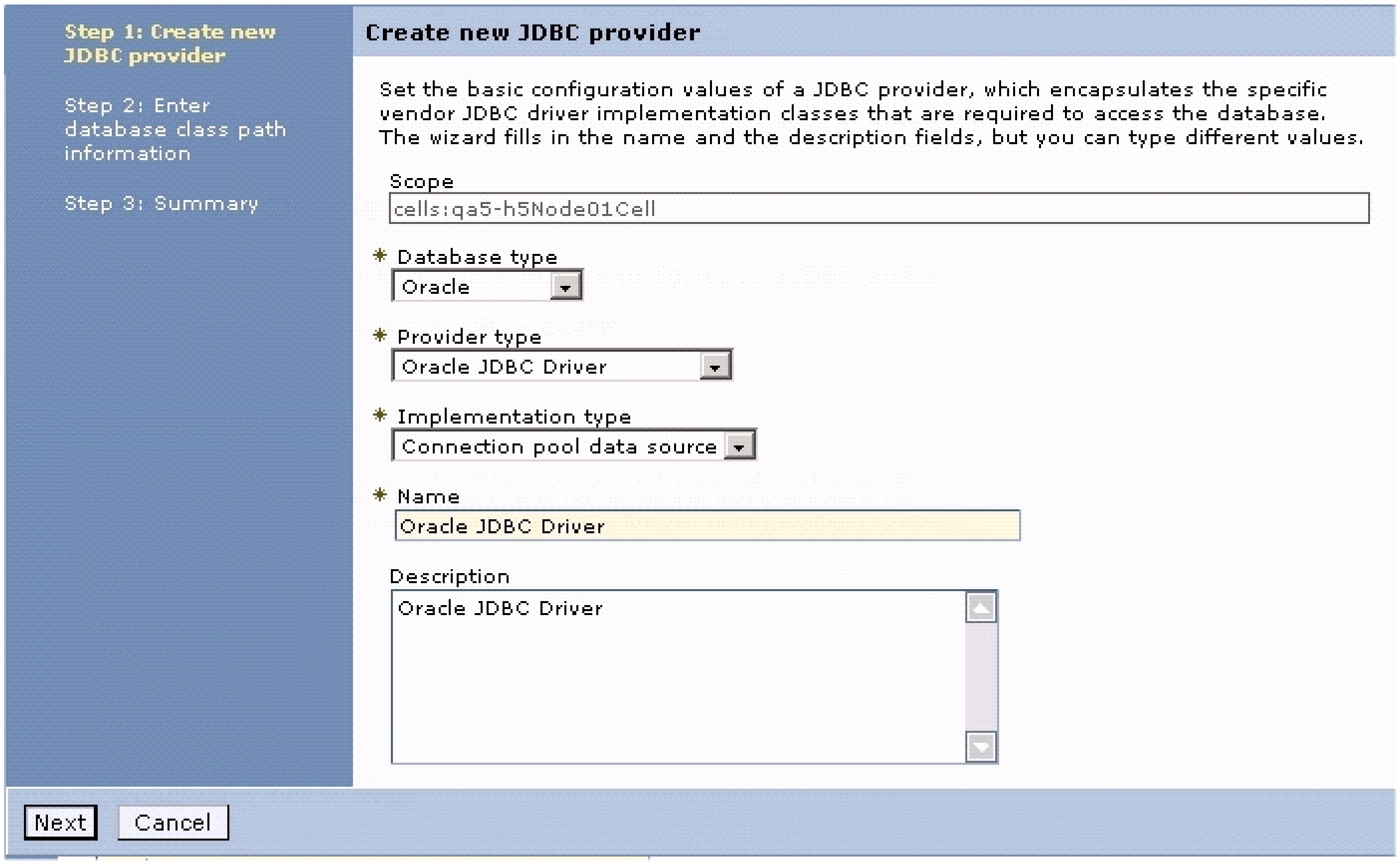
Configuration Files

The following configuration files are automatically generated by Composer's WAR export functionality and do not require any user action: web.xml and ibm-web-bnd.xmi

Creating a JDBC Provider for an Oracle Database

SQL Server driver is built-in for WebSphere. However, the Oracle driver must be configured as a JDBC provider.

1. From the left-hand side panel, open **Resources** > **JDBC** > **JDBC Providers**.
2. Click **New**.
3. In Step 1, choose the following:


Step 1: Create new JDBC provider

Step 2: Enter database class path information

Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:qa5-h5Node01Cell

* Database type: Oracle

* Provider type: Oracle JDBC Driver

* Implementation type: Connection pool data source

* Name: Oracle JDBC Driver

Description: Oracle JDBC Driver

JDBBProvider.gif

4. In Step 2, specify the location of the `ojdbc14.jar` file. The JAR can be copied from Composer's tomcat/lib directory to a location local to the WebSphere server.

Creating Data Sources

1. On the left-hand side panel, open **Resources** >**JDBC** > **Data sources**.
2. Click **New**.
3. Enter anything you like under **Data source name**.
4. Under **JNDI name**, enter the name that matches the one given in the Connection Profile Editor. Hit **Next**.
5. For the **Select JDBC** provider step, choose **WebSphere embedded ConnectJDBC** driver for MS SQL Server for SQL Server, or **Oracle JDBC** driver for Oracle. Hit **Next**.
6. Enter the database name, host name and port of the database server. Click **Next**. Click **Finish** on the summary page.
7. Next , you must specify the username and password for the database connection. Click on the data source that was just created and then click on the **Custom Properties** link.
8. Create two new properties, called user and password, and specify the credentials for the database.
9. After saving the data source, use the **Test Connection** button to test.

10. Use the **Connection Pool Properties**, link to customize the pooling settings. Refer to the WebSphere documentation for details.

The following items are generated by Composer's WAR export functionality and require no user action. `WEB-INF/web.xml is required, similar to JBoss.      <resource-ref id="ResourceRef_1276009394684">     <res-ref-name>jdbc/pooledDS</res-ref-name>     <res-type>javax.sql.DataSource</res-type>     <res-auth>Container</res-auth>     </resource-ref> WEB-INF/ibm-web-bnd.xml` does the same thing as `jboss-web.xml` does for JBoss... `<?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>`

Route Applications and Workflows

This section contains the following:

- Getting Started with Routing Applications
- Preferences for Routing Applications
- Creating Routing Applications
- Routing Block Palette Reference
- Event and Exception Handling

Also see Interaction Process Diagram Blocks.

Getting Started with Route Applications

The information in this book will help you get started using Composer to build SCXML-based strategies (hereafter called routing applications) which can be comprised of one or more workflows). It assumes you have reviewed the topics in the general Getting Started with Composer section.

Preparation

Composer provides a wide range of tools to satisfy the needs of a diverse developer population. Ideally, you will be already be familiar with SCXML, XML, and HTML. If you do not wish to write code or use existing code templates, you can build routing workflows using Composer's designer where you place, configure, and connect routing blocks.

Routing FAQs

This page provides answers to common questions that IT personnel might have when planning or considering the addition of Genesys Routing to their site. The information on this page applies to 8.1.x versions of Composer.

What is Genesys Customer Experience Routing and how is it unique?

Genesys Customer Experience Routing is computer software that helps organizations better manage customer journeys. Routing prioritizes and matches the right interaction with the right resource at the right time. Our approach is unique in the industry because it’s:

- SIMPLE to support the 80% of customer interactions that are routine
- DYNAMIC to automatically adapt to fluctuations within the 80% (so this variability doesn’t consume 100% of resources)
- POWERFUL to drive the 20% of interactions that are not routine but are the most valuable (across time, channels, multimedia, front and back office)

We help companies create better customer experiences. Our DYNAMIC routing frees you to do more than just what is SIMPLE. And that gives you bandwidth to apply the full POWER of Genesys to those moments that truly matter.

What is a routing application? What are the basic elements?

Routing provides instructions about how to handle and where to direct interactions under different circumstances.

Conceptually, a routing application is like a series of prioritized instructions ”if&ldots;, then&ldots;, else&ldots;” that take into account various factors to determine the optimal routing target, and what to do next if that action is not possible within the specified constraints.

Routing applications are made up of a number of different elements, described here at a conceptual level:

- Data can come from various sources and may include customer, contextual, operational, or analytical data. Attached Data, which is included in call messaging as Key Value Pairs (KVPs), is what you know about a specific interaction. Attached data can be added and updated throughout the life of the interaction (e.g., as a call flows through the IVR, routing, agent desktop, and reporting).
- Skills are what you know about an agent. To identify the best available resource to handle a particular interaction, routing looks for desired combinations of Skills at the individual level (per agent), at the team level (per skill group, or ’queue’), or across a virtual pool of resources (per ’virtual queue’).

Skills should not represent absolutely everything about agents, but simply the minimum needed to accurately route and report on interactions. Because of the combinatorial power of Skills, it is best not to get too granular. Modify an agent’s Skills only when the agent acquires new job functions, training, or capabilities; do not change agents’ skills merely to redirect traffic.

Each Skill can optionally have a Proficiency, which rates how relatively good an agent is at a Skill (e.g., Spanish level 5 vs. 10). This allows an organization to route to the best-skilled available agent, and then if no agents at that proficiency level are available within a certain amount of time, expand the target to agents with a lower proficiency level and/or an alternative combination of skills.

Logic provides the overall routing decisioning or instructions. Logic specifies the conditions under which the routing applies and the method of target selection. The logic can be based on a number of different considerations, such as skill targeting, service level, load balancing, percentage allocation, statistics, or workforce. (See below for more details.)

Certain aspects of routing can be configured and saved as Reusable Objects. There are various types of reusable objects, including subroutines, list objects, interaction data, etc. Reusing these building blocks within and across routing applications improves the efficiency, quality, and simplicity of the routing.

A well designed and implemented routing solution should be able to handle most of the ongoing routing needs in a dynamic and automated fashion. However, there may be some situations where the business needs or wants to make changes on a frequent or ongoing basis. These select elements can be exposed to business users either as Operational Parameters or as Business Rules to facilitate greater business agility while maintaining system stability:

- Operational Parameters are simple conditional variables that give business users limited control (e.g., After Hour Messages, Hours of Operation, Emergency Status, etc.). Users can make changes to these parameter settings through the Genesys Administrator Extensions (GAX) interface. (Alternatively, this can also be done via list objects in Interaction Routing Designer (IRD).) The business user cannot change the underlying logic (only the pre-specified values of the exposed parameters), and does not require any specialized technical training.
- Business Rules are logical representations of underlying routing that are written in plain language (i.e., meta-language, not code). They are useful when the business user (typically a business analyst) wants greater control over the conditions, logic, and actions associated with the routing (e.g., create differentiated customer service treatments based on segmentation, marketing campaigns, etc.). Users can make updates to the business rules, but only for those parts of the routing that have been exposed through the business user interface within Genesys Conversation Manager. Although the business user isn’t actually viewing or changing the code directly, they still require a clear understanding of the business logic and potential impact of changes.

What are some of the most common types of routing?

The table below lists the most common types of routing.

TYPE	DESCRIPTION
Agent Group	sRouting interactions to a specified group of agents. This may be based on job type (e.g., Tier1Agents), location or site (e.g., MiamiAgents), etc.
Auto Attendant	Routing implemented to support simple menus

	(e.g., audio prompts and touchtone selections), mimicking the functionality of a basic IVR.
Blended	Routing which allows the same agent or select resources to handle more than one type of interaction (e.g., Inbound/Outbound, multimedia). Blending should be used to make use of underutilized resources and to prevent service level fluctuations (e.g., forcing agents to log off a voice queue due to an influx of Social Media interactions). Consider how many interactions of each type an agent can handle at a time and define capacity rules according. Also, increment and/or cap priority values based on interaction types, so voice interactions don’t always take precedence over non-voice ones, or vice-versa.
Business Case	Routing to provide differentiated customer service treatments for specific business processes or use cases (e.g., marketing campaigns, account status, payment due, collections, regulatory, etc.).
Callbacks/Virtual Hold	Routing that accounts for the prioritization and targeting when a call back to a customer is required, requested, or scheduled.
Cascading Routing	Routing that uses multiple tiers of prioritized routing decisioning, such that if the conditions for the highest priority routing instructions are not met, the routing automatically overflows to the next level of routing instructions. Conditions can also be checked in parallel, so that time is not wasted waiting to execute the first tier of decisioning before considering the next one.
Concierge Routing/Hunt Groups	Routing to a specific agent or a small group of individuals when specialized or personalized service is required. Typically the interaction is first directed to the primary agent assigned to a particular customer account. However, if that agent is unavailable, the routing will search for the next available team member within a small hunt group.
Cross-Channel	Routing based on what a customer was just doing on another channel (e.g., a customer is on the company’s website or mobile application and then calls in).
Default Routing	Routing an interaction to the default destination that is to be used when none of the conditions for the previous tiers of routing decisioning have been met. This typically occurs when traffic volumes spike for some reason and the timeout thresholds for the previous tiers have been exceeded, so the interaction overflows to the final default destination.
Dynamic Routing	Routing that automatically adjusts based on pre-specified priorities and conditions. Examples include: cascading routing, target expansion,

	timeout thresholds, data dips, holidays, emergencies, service outages, etc. Dynamic routing is an efficient and valuable alternative to reskilling agents on the fly, an inefficient and costly practice that is often used in legacy contact center environments to manually redirect traffic.
Enterprise Workload Management	Routing of work items across the enterprise. The same Genesys routing capabilities that can be used to direct customer-facing interactions (calls, emails, chat, etc.) can also be leveraged to schedule, assign, distribute and track work activities across the back-office.
Escalations	Routing of interactions which require the support or intervention of a more highly skilled agent (e.g., 'Tier 2') or manager. This may be handled as a transfer, or it may involve a conference call or consultative support with the specialist.
eServices/Multimedia	Routing of various types of non-voice interactions (e.g., email, chat, text, social, video, open media). Different media types may require unique skills (e.g., +Written could be a skill type for email, chat, and text). Consider how many interactions of each type an agent can handle at a time and define capacity rules according (e.g., 1-4 chats per agent).
Interaction Type (a.k.a. Call Type)	Routing based on the type of customer and/or ccustomer's intent. This is typically determined based on the number dialed (DNIS), from the caller's menu selection or activity within the IVR, or from content analysis on an email or chat.
Interactive Voice Response Integration	Routing a call to the appropriate target based on what the caller did or selected within an IVR. Based on integration with Genesys Voice Platform (GVP) or a third-party IVR.
Last Agent Routing	Routing to the last agent the customer interacted with. Especially useful for routing to a single point of contact (such as a case owner) or for dropped calls that call back in within a specified timeframe.
Outbound	Routing of interactions that are initiated by the organization and directed outward to the customer (e.g., outbound calls, marketing campaigns, collections, outbound emails, text messaging, proactive contacts, etc.).
Overflow/Sharing Agents	Routing to an alternative queue or agent group, when the primary target is unavailable or over-utilized. Lending and borrowing of resources can be contingent upon certain predetermined business conditions being met, so that spikes in one team's volume does not unduly impact another team's availability or service levels.
Percentage Allocation	Distributing interactions between queues based on a percentage of total volumes (e.g., 60% to Site A and 40% to Site B).

Priority Queuing	Routing which uses priority values to give preference for one queue or interaction over another. Priorities can be incremented over time, so if a lower-ranked interaction has been waiting longer, it will be serviced before a higher-ranked interaction that has just arrived. This ensures that no interaction ever waits too long for service.
Queue Treatments	Routing that plays audio (e.g., music, ads, messages) or provides certain functionality while callers are waiting in queue or on hold.
Ring No Answer/Redirect on No Answer (RONA)	Routing to an alternative target if the original target fails to answer (e.g., agent failed to log out). The agent will be targeted the first time, but after that an action can be specified (e.g., log out) so that agent isn’t targeted again subsequently.
Segmentation	Routing based on the type of customer, the value of the opportunity, or other marketing segmentation data.
Skills-Based	Routing to the best-skilled available agent based on a combination of skills specified in the routing. This is sometimes called ’agent-level routing,’ since Genesys routing is capable of looking down to an individual agent’s unique set of skills. However, in practice routing typically looks for the desired skill set across a ’universal queue,’ to optimize utilization across a large pool of resources.
Statistical Routing	Routing based on various database lookups and operational conditions, such as Estimated Wait Time (EWT), queue depth, service levels (SLAs), performance goals, agent occupancy, skill utilization, seasonality, special events, business processes, etc.
Target Expansion	Routing that expands its targets to increase the pool of agents able to handle an interaction, be it after a time period or triggered by the Estimated Wait Time (EWT) being greater than a defined threshold. The highest skill level is first targeted until the time limit is reached, and then routing expands to include the next level of skills, cascading down until all skill levels are included in the targeting. This ensures that if the best suited pool of agents are unavailable, then after the expansion timeout the next best pool of agents are included in the targeting.
Transfers	Routing to handle transfers. Need to consider the routing for transfers that are directed either into or out of the contact center. The routing priority may vary depending on whether it is an internal transfer (within the contact center) or external transfer (to/ from an outside group or entity).
Workforce	Routing that factors in various workforce considerations such as schedules, shrinkage, absenteeism, training, skill development, desktop/

	tools, new-hires/career paths, agent affinity for particular interactions, outsourcers, unions, labor laws, etc.
Voicemail	Routing of inbound calls to voicemail (e.g., after hours group voicemail inboxes). Or outbound routing which addresses what to do if a voicemail is reached (i.e., leave a message or not).

How many skills total does an organization typically have?

It depends on the size and requirements of the organization, but generally we see a range somewhere between 20-75 skills total. Once you start to approach 100 or more skills, you need to question if you are really taking advantage of the combinatorial power of Genesys skills (i.e., where agents can be multi-skilled and Genesys routing can look for multiple combinations of skills).

The average agent is typically highly proficient in 3-4 skills each, but may have lower proficiency in other skills to provide backup. Expert agents may be highly proficient in 10 or more skills.

Skills and proficiencies grow and change over time, which is useful for staff development and retention. Skills need to be monitored and aligned across staffing and routing.

If you find there are certain skills &endash; a, b, c &endash; that every agent has, then maybe you’ve dissected the skills too granularly. Try renaming/regrouping these into one mega-skill (e.g., A). At the same time, you don’t want to group so many skills together that you’ve gone back to queue-based routing, where each skill maps to a separate queue.

If an organization requires many skills, rather than hard-coding each one separately directly into the routing logic, a better and simpler approach may be to reference the skills as variables within the routing logic. Then do a data-dip into a database or table look-up from a separate file. That way when skills need to be modified, this can be done in the external data source housing the skill information, without having to change the actual routing logic itself. Soft-coding skills is an effective approach if you find that skills change frequently over time, but the core routing does not. Certain industries demand a high level of subject matter expertise (e.g., finance, insurance, healthcare), so there are more total skills the organization needs. At the same time, since each agent requires more specialized expertise to handle these inquiries, each agent typically handles fewer call types than in other industries where agents may be more of generalists.

Don’t confuse Skills with Attached Data. For instance, consider situations in which many corporate clients need to be supported, or there are state-specific licensing requirements (e.g., 401ks, insurance plans). The specific account or plan can be identified based on the phone number dialed (DNIS) or other information gathered in the IVR and attached to the call. There may be hundreds of these possibilities. However, this doesn’t necessarily mean there need to be hundreds of different skills corresponding to each. An individual agent might be trained to handle a more generalized skill (e.g., 401Ks in general), and a particular plan’s specifics can be screen-popped through to the agent’s desktop based on the Attached Data.

How many routing applications should an organization have?

As a rule of thumb, a large contact center solution (a major line of business) should not need more than 10 routing applications and subroutines (not counting reusable objects and subroutines used across applications).

It’s important to encompass two key design considerations when planning routing &endash; Flexibility and Simplicity. This can be done by creating generic components and modularizing parts for reuse. A routing model which is data-driven and accommodates the logic shared across applications and lines of business helps to eliminate duplicated logic or code. Functionality which is replicated should be separated out into a sub-routine to minimize the need to change multiple applications for feature enhancements and/or defect fixes. This minimizes the number of applications required and still meets the demands of complex routing requirements.

What are skill proficiency levels, and what are they used for?

Proficiency is an optional way of reflecting how relatively good an agent is at a particular skill (e.g., Spanish level 5 vs. 10). Following the ’Simplicity’ design principle, it’s best to keep to three (or fewer) levels of skill proficiencies &endash; for instance, High = 9, Medium = 6, and Low = 3. This allows additional proficiency levels to be added in between if required in the future.

Proficiency enables Target Expansion &endash; e.g., first target agents with skill of Sales ≥ 9 proficiency for 15 seconds, then target Sales ≥ 6 for 15 seconds; then target Sales > 0). This circumvents agents having to log off one agent group/queue and log into another, which is a common issue with legacy ACD-based solutions and can be avoided using Genesys routing.

How many tiers of cascading routing should there be?

With basic Skills-Based Routing, 4 tiers are typical &endash; 3 for the three skill proficiencies and the forth tier for emergency (e.g., breached threshold, all agents log off).

When using additional soft skills to provide an extra level of customer experience, then an additional tier will be required before the 4 tiers previously mentioned.

What Reporting considerations need to be taken into account?

First, the Reporting requirements need to be well defined. What are the business goals of the solution? How will success be measured? What are the KPIs? How does the business need to slice and dice the data? How will reporting be represented? What needs to be monitored in near real-time vs. historically? Who are the different consumers of reporting and what do they want/need to see? What business intelligence is needed &endash; analytics, trends, outliers, outcomes, actionable insights, alerts?

Routing must then be aligned with those Reporting needs. This is typically supported through Attached Data associated with each interaction (e.g., line of business, customer segment, routing point/agent, service type, disposition code, business result, etc.). Decide on a flexible approach for attaching data. Don’t attach too much (as it may have a performance impact). Consider codifying values to reduce the total data overhead. And be very clear about what data represents at the point it was attached.

What Workforce Management considerations need to be taken into account?

Genesys routing allows an interaction to be serviced by the best-skilled available agent across a virtualized pool of resources, and to expand the target (to lower proficiency level and/or a different skill set) if the desired target isn’t available. Altering the original target (such as in target expansion) will always affect Workforce Management (WFM), so it’s important to include WFM into the routing considerations.

- For instance, sometimes an agent might be working on a call type that is outside of what they normally work on. So supervisors/team leads need the right insight to know their people are working on the right thing at the right time.

Genesys Routing works with a variety of industry WFM solutions, but there are additional advantages to using Genesys Workforce Management:

- The Genesys WFM solution provides historical data collection and real-time analytics for all interaction types being monitored by the Genesys environment.
- Genesys WFM integrates with the Genesys suite to utilize all of the Site, Agent, Skill, and Skill level information contained therein.
- Genesys also provides the ability to base routing on agents’ specific future schedule states in Genesys WFM. For instance, if an agent is scheduled to go on break soon, routing will not direct an interaction to them, to stay in adherence.

Skills should not be changed to re-route traffic, due to absenteeism or overflow.

- Frequent ad hoc re-skilling of agents (to redirect traffic flow) is inefficient, fails to leverage dynamic routing, and can wreak havoc with the accuracy of WFM forecasting for skill types.
- Agents should already have their skills and proficiencies in their profiles, but they may be scheduled to take particular call types based on their scheduling and routing logic. Re-skilling of agents typically only happens if they have acquired new skills (after training) or taken on a new job role.
- Most interaction flows should be handled via dynamic routing (such as target expansion). If traffic must be manually redirected, then rather than re-skilling agents, keep agent skills the same and redefine the ’activity set’ object within Genesys WFM. This reschedules agents to work on different activities during a given time period. That way you are rescheduling the types of work they are handling, rather than changing the agents’ actual skills. This approach is based on doing schedule-based routing (not just skills-based routing), and has a dependency on Genesys WFM, thus taking advantage of the interoperability across the Genesys suite of solutions.

What are the best practices for migrating from traditional queue-

based routing to Genesys Customer Experience Routing?

The most common mistake that organizations make when moving away from legacy ACD environments is trying to replicate a like-for-like solution. While this is sometimes inescapable as an interim step (e.g., due to end-of-life equipment), it should be avoided at all costs as the end state. Seize the opportunity to re-evaluate your current customer experience and create an optimal solution:

- Start by identifying the business goals and customer experiences you want to deliver.
- Segment your customers and determine an appropriate customer service strategy for each (e.g., Elite Customers, High Value, Mass Market, and Low Value).
- Consider the various channels and contact drivers of customer interactions. Rather than treating these as siloed touch points, craft them into seamless customer journeys. (These journeys will likely vary per segment.)
- Evaluate your workforce and identify their hard and soft skills. Determine which skill sets and proficiencies are needed to deliver the desired customer journeys. Are there gaps? Do job roles, teams, or training need to change?
- Prioritize (rank) desired customer journeys and match with optimal skill targets for each. Then consider the next best treatment and target if these conditions cannot be met.

As a rule of thumb, routing should be designed so that:

- Your most valuable customer interactions (top 10-20%) receive the best service most of the time.
- The majority of your customer interactions (60-80%) receive good service (e.g., slightly longer wait, less skilled agents) much of the time.
- Your costly customer interactions, overflows, or exceptional situations (bottom 5-20%) receive adequate service and the minority of the time.

Routing My Interactions

You can use Composer's predefined blocks and/or write your own SCXML code to create routing applications that route based on various criteria such as: **Agent, Agent Group, ACD Queue, Place, Place Group, Route Point, Skill, or Variable**, **last called agent**, **date and time**, **the value of a statistic**, **dialed number (DNIS)**, **originating number (ANI)**, **percent and conditional routing**, and **load balancing**.

IRD Functionality Included in Composer

Composer enables you to create SCXML-based routing applications to run on the Universal Routing 8.x platforms and, as such, it includes functionality that was previously provided through Genesys Interaction Routing Designer (IRD). The information below is provided for existing Genesys customers transitioning to Composer, who are familiar with creating strategies in IRD.

Composer Blocks and IRD Objects

Composer refers to the fundamental element of a workflow as a "block" whereas in IRD documentation, this element is referred to as an "object." The tables below group IRD objects based on their IRD toolbar category name and point to the corresponding functionality in this release of Composer. Summary information is presented below. For more information, see Composer on the Genesys Documentation Wiki

Data & Services

IRD Object Name	Composer Block Name	Description
Database Wizard	DB Data	DB Data retrieves information from the database. Uses a Query Builder.
Web Service	Web Service	Invokes Web Services. GET, POST and SOAP over HTTPS are supported.
	Web Request	Invoke any supported HTTP web request or REST-style web Service. See sample: Routing Based on Web Request.

Miscellaneous

IRD Object Name	Composer Block Name	Description
Assign Multi-Assign	Assign	Assigns a computed value/ expression or a literal value to a variable. Variables are defined in the Entry block. Capable of multiple assignments.
Call Subroutine	Subroutine	Creates reusable sub-modules.
Entry	Entry	Sets global error (exception) handlers. Defines global variables (see Variables section

		below).. All routing strategy diagrams must start with an Entry block.
Exit	Exit	Terminates the strategy and returns control back to calling workflow in case of a subroutine.
Error Segmentation	Multiple error output ports can be created in Composer blocks based on each block's Exception property.	
Function Multi-Function	ECMAScript	Builds an ECMAScript expression using the Expression Builder. Many URS functions are available as Genesys Functional Modules described the Orchestration Server Documentation Wiki Can invoke multiple functions.
If	Assign, Branching, ECMAScript blocks all open Expression Builder	Expression Builder can be used to create IF expressions.
Multi-Attach	ECMAScript	Can be used for attaching data to an interaction.

Routing

IRD Object Name	Composer Block Name	Description
Selection	Target	Routes an interaction to a target, which can be Agent, AgentGroup, ACDQueue, Place, PlaceGroup, RoutePoint, Skill, or Variable. Skill target uses Skill Expression Builder.
Percentage	Target	Statistics Order property in Target block, lets you perform percentage allocation. Also see sample: Routing Based on Percent Allocation.
Default	Default Route	Routes the interaction to the default destination. Can be overrridden by the Set Default Route block.
Routing Rule		Orchestration Server 8.1 does not support service level routing rules.
Switch to Strategy		Orchestration Server 8.1 does not support switch to strategy routing rules.

Force Route	Force Route	Not exposed as a routing rule in Composer.
Statistics	Target	Although statistical routing rules are not yet supported as in IRD's Statistics routing object, users can use the Target object Statistic property to route based on the value of a statistic. A Statistics Manager and Builder let you create your own statistics from URS predefined statistics.

Segmentation

IRD Object Name	Composer Block Name	Description
ANI	Branching	See Your First Application: DNIS Routing for an example.
DNIS	Branching	See Your First Application: DNIS Routing for an example.
Date	Branching	See the sample Routing Based on Date & Time.
Day of Week	Branching	See the sample Routing Based on Date & Time.
Time	Branching	See the sample Routing Based on Date & Time.
Classification Segmentation	Branching	For classification segmentation, an ECMAScript function determines if a particular category name or ID exists in the array of category objects represented by an application variable.
Generic	Branching	Use as a decision point in a workflow. It enables you to specify multiple application routes based on a branching condition.

Also see Context Services Blocks.

Voice Treatment

See Composer Equivalent to IRD Treatment.

eServices Multimedia

See Composer Equivalent to IRD Multimedia.

Outbound

See Outbound Common Blocks

Context Services

See Context Services Blocks

Business Process

See Interaction Processing Diagrams Overview and Interaction Process Diagram Blocks. Reusable Objects

- IRD List Object: See Composer's List Object Manager.
- IRD Variable List Dialog Box: See Entry block Variables property.

In contrast to IRD, which defines variables in a special dialog box outside of the strategy, Composer defines both workflow and Project variables.

Workflow Post Installation

Post-installation configuration tasks include:

Tomcat

This step is necessary for both voice and routing applications. For Tomcat settings:

1. Select Window > Preferences, then expand Composer and select Tomcat.
2. Provide the same port number that you specified during installation. The default user name and password for the bundled Tomcat is admin.
3. To start Tomcat, click the button on the main menu. If necessary, see Tomcat Service Failed to Start.

If you already have Java Composer Projects in the workspace and did not perform the Tomcat configuration earlier, perform the following steps to deploy the project on Tomcat:

1. From the Project Explorer, right-click on the Java Composer Project and select Properties.
2. Select Tomcat Deployment and click the Deploy button.

Note: This also needs to be done if a Java Composer Project is imported or renamed as well.

Configuration_Server

Routing applications may be developed either:

- With a connection to Configuration Server
- Or in an "offline" mode, without connecting to Configuration Server

Whether or not to connect depends on what you wish to do. For example, you would need to connect to Configuration Server in order to access configuration objects through the Target block. You can connect to Configuration Server now or wait until strategy design time. To bring up the Connect Configuration Server dialog box:

1. From the main menu, select **Configuration Server > Connect**. Or use the keyboard shortcut: Alt+I+C. (To disconnect, keyboardshortcut is: Alt+I+D).
2. Enter **Username**, **Password**, **Application**, **Host**, and **Port** information for the Configuration Server used in your environment.
3. Select **Use Secure connection** for Transport Layer Security (TLS) when connecting to Configuration Server.

4. Click **Next:**

    - If authentication with the supplied User Name and Password is unsuccessful, Composer displays informational text in a Configuration Server Connection Error dialog box.
    - If a secure connection cannot be made, or if Transport Layer Security is not configured, a Configuration Server Connection Error dialog box appears.

In both of the above scenarios, click the Details button for more information.

1. Select the **Tenant**. For a single-tenant environment, select **Resources**.
2. Click **Finish.** Composer can now access Configuration Server data during validation (if configured to do so) and other operations.

Notes:

- You can configure an inactivity timeout for the connection to Configuration Server as well as the time for the timeout warning dialog. For information on these features, see the *Genesys 8.1 Security Deployment Guide*.
- For making live calls, you must manually configure the Routing Point in the Configuration Database as described in the chapter on creating SCXML-based strategies in the *Universal Routing 8.1 Deployment Guide*. You must also configure other Universal Routing Server options as described in that guide.
- Routing applications are not stored in Configuration Server as in 7.x and earlier. They are stored in the Workspace that you specify.

MIME_Types

MIME (Multipurpose Internet Mail Extensions) refers to a common method for transmitting non-text files via Internet e-mail. By default the SCXML MIME type is already configured in the Tomcat server bundled with Composer. If you are using the Internet Information Services (IIS) Application Server to deploy SCXML strategies, add the following MIME type extensions through the IIS Manager of your webserver:

.json	text/json
.scxml	text/plain
.xml	text/xml

Predefined_Statistics

There is an option to control whether or not to create Universal Routing Server predefined statistics. You will want to do this if you plan to route based on the value of a statistic (for example, statistic `StatTimeInReadyState`).

1. Select **Window** > **Preferences**.

2. Expand **Composer** > **Configuration Server**.
3. Check the box: **Create router predefined statistics** when connecting to Configuration Server.

Orchestration

In addition to specifying the HTTP request parameters, both Universal Routing Server (URS) and Orchestration Server (ORS) must be properly configured outside of Composer using Configuration Manager or Genesys Administrator. In addition to specifying HTTP request parameters, the URS configuration option strategy must be set to ORS. This ensures that URS is prepared to process interactions according to requests received from ORS. For more information, including information on additional options that must be set, consult the following: *Composer 8.1 Deployment Guide*, Post-Installation Configuration chapter. *Universal Routing 8.1 Deployment Guide*, Orchestration Support chapter. *Orchestration Server 8.1 Deployment Guide*, SCXML Strategy Support and Configuring Orchestration Server chapters. Important! if you have both Composer and IRD set up in the same environment, check in Interaction Routing Designer's Loading View that you have not loaded an IRD 7.x routing strategy on the same Route Point DN where the built-in strategy is loaded. This will create a conflict and cause your SCXML application not to launch.

Stream_Manager

Perform these steps in Configuration Manager or Genesys Administrator if using Stream Manager to play treatments via the Composer treatment blocks (such as PlaySound). After installing Stream Manager as described in the *Framework 7.6 Stream Manager Deployment Guide*:

1. Set up a SIP Switching Office and a SIP Switch.
2. Set up a SIP T-Server with an association to the SIP Switch.
3. For your SIP T-Server, ensure that the sip-port option under the TServer section is unique in your environment.
4. Make sure there is a connection between your SIP T-Server and Stream Manager.
5. For Stream Manager options, in the contact section, make sure the SIP port is unique in your environment.
6. On your SIP Switch, create a DN of type Voice over IP Service to enable Stream Manager to properly play the treatments. For information on Stream Manager and the Voice over IP Server type DN, refer to the *Voice Platform Solution 8.1 Integration Guide.*
7. In the Annex tab of this DN, add a section called TServer with the following options:
   - Name: contact, Value: *:*<IP Address of Stream Manager>:<SIP Port of Stream Manager>
   - Name: service-type, Value: treatment

Optional

8. You may also need a DN of type `Trunk` for your SIP softphone. In the Annex tab, add a section called `TServer`.

- Name: contact, Value:<IP address of where SIP softphone is running>

Defining Preferences

You can configure Preferences for SCXML-based routing applications now or later.

ORS_Debugger

You can configure Preferences for the ORS Debugger now or later. To set ORS Debugger preferences:

1. Select **Window** > **Preferences**, then expand **Composer** and select **Debugging.**
2. Specify the following settings:
    - **Network Interface**. Composer debugging uses this setting to make the socket connection for the Debugger control channel. Select the interface that is applicable to your scenario. The debugging server (GVP or ORS) must be able to access the Tomcat server, bundled as part of Composer, for fetching the Voice or Routing application pages. If you have multiple NIC cards of multiple networks (such as Wireless and LAN) select the interface on which GVP or ORS will communicate to your desktop. In case you are connected over VPN, select the VPN interface (such as PPP if connected via a Windows VPN connection).
    - **Client Port Range**. Enter a port range to be used for connection to ORS for SCXML debugging sessions.
3. Expand **Debugging**, select **ORS Debugger**, and specify the fields below. You can change this information when creating a launch configuration.
    - **ORS Server Host Name**. Enter the IP address for the ORS Server.
    - **ORS Server Port**. Enter the debugger port for the ORS Server.This is defined in the ORS configuration as [scxml]:debug-port, and defaults to 7999. ORS must have `debug-enabled` set to `true`.

Note: New launch configurations are pre-populated with the above host name and port information, which can be changed.

- **Use Secure Connections**. Check to enable secure communications (SSL/TLS) between the Composer client and ORS, for SCXML debugging sessions. The connection between Composer and ORS is mutually-authenticated TLS if implemented on the ORS side. Note: As of the Composer 8.1.1 release date, this feature is not yet implemented on the ORS side.

Upgrading Workflows

Upgrading workflow diagrams created in the 8.0.4 and later releases of Composer is supported as described in Upgrading Projects/Diagrams.

Upgrading Workflows Prior to 8.0.4

Some previously created workflow diagrams cannot be upgraded:

- Composer 8.0.2 began support for the creation and testing of SCXML-based workflows for inbound voice use cases. Upgrading workflow diagrams created in the 8.0.2 release of Composer, which introduced this new feature, is therefore not supported.
- Composer 8.0.3 began support for the Context Services option of the Universal Contact Server Database and the processing of multimedia interactions. This release also introduced interaction process diagrams, which are roughly the equivalent of IRD business processes. Upgrading workflow diagrams created in the 8.0.3 release of Composer, which introduced these new features, is therefore not supported.

Migrating IRD Strategies into Composer Projects

Starting with Composer 8.1, you can migrate routing strategies created with Interaction Routing Designer 8.0+ into Composer Projects as SCXML-based workflow diagrams. For more information, see the IRD to Composer Migration Guide.

Preferences for Routing Applications

Composer preferences apply to all Projects within the workspace. You can set preferences for the following:

- Business Rules
- Composer Diagram
- Configuration Server
- Gax Server
- Context Services
- Customizer
- ORS Debugger
- Time Zone
- SCXML File
- Security
- XML Preferences

To open the Preferences dialog box, select **Window** > **Preferences**.

Business Rule Preferences

See Business Rule Preferences in Rules Preferences Business Rule Common block.

Configuration Server Preferences

To set Configuration Server Preferences:

1. Select **Window** > **Preferences** > **Composer** > **Configuration Server**.
2. Select or clear the check box for **Create Router predefined statistics when connecting to Configuration Server**.

Set this preferences if you plan to route based on the value of a URS predefined statistic. For more information on URS predefined statistics, see the chapter on routing statistics in the *Universal Routing 8.1 Reference Manual.*

3. Select or clear the check box for Validate Skill Expressions. You have the option of clearing the check box when using complex skill expressions that use both literal expressions and variables, for which skill expression validation fails.
4. Specify Configuration Database object preferences:
    - **No validation**. You may wish to select this option if objects that will used in routing have not yet been configured.You may also wish to select this option if you do not have the required Configuration Database permission as described in the Genesys 8.1 Security Deployment Guide. For the Configuration Database, permissions and security are defined in the Security tab of the properties dialog box in Configuration Manager or (for web access) Genesys Administrator. The No Validation setting also allows application development to continue when access to Configuration Server is not currently available. Composer can still validate strategies with the Configuration Server items excluded.
    - **Validate if connected**. If you will not always be connected to Configuration Server, you may wish to select this option.
    - **Validate**. Select to have Composer validate that the objects exist in the Configuration Database.
5. **Published interaction process diagram when it is saved**.If checked, Composer will publish an interaction process diagram when it is saved. It will not publish or prompt to connect to Configuration Server if disconnected. Note: This auto-publish does not display a message when publishing is successful. However, it will display message if publishing fails.
6. Check or uncheck **Prompt to save before Publishing Interaction Process Diagram**.
7. uncheck **Delete published objects when Interaction Process Diagram is deleted**.
8. uncheck **Delete published objects when Project is closed or deleted**.
9. Set the **Inactivity Timeout** preference to have Composer automatically close the Configuration Server connection when the user does not interact with Composer in any way for the inactivity-timeout period as described in the Inactivity Timeout chapter of the *Genesys 8.1 Security Deployment Guide*. Composer displays a warning dialog two minutes in advance of this time. By default, the inactivity timeout is to be set to 0 (turned off).

Diagram Preferences

Select **Window**> **Preferences** > **Composer** > **Composer Diagram**. The following preferences for diagrams can be set in the Preferences dialog box:

Global Settings

1. Select or clear the check box for each of the following diagram global settings:
    - **Show Connection Ports**. If enabled, connection ports (both exception ports and out ports) are always displayed on blocks. This makes it convenient to draw links between blocks and to get immediate feedback on how many ports each block provides. However, in this case, the ability to reposition connections on a block is not available. If switched off, connection ports are not displayed by default, but repositioning or finer control over connection link placement becomes available. Note: This preference applies to all projects and is not available for individual projects.)
    - **Show popup bars**. If enabled, this setting displays basic blocks from the blocks palette in a pop-up bar if you hover your mouse on the diagram for one or two seconds without clicking. Note: blocks are shown in icon view only.)
    - **Enable animated layout**. If enabled, causes diagrams to gradually animate to their location when the Diagram \> Arrange \> Arrange All menu option is clicked.
    - **Enable animated zoom**. If enabled, while using the zoom tools, shows a gradual transition between the initial and final state of the diagram on the canvas. If off, the zoom is instantaneous. Similar behavior for animated layout when the Diagram \>\> Arrange \>\> Arrange All menu option is clicked.
    - **Enable anti-aliasing**. If enabled, improves the appearance of curved shapes in the diagram. You can see its effect on the circles in the Entry and Exit blocks.
    - **Show CodeGen success message**. If unchecked, then the confirmation dialog at the completion of code generation will not be shown.)
    - **Prompt to Save Before Generating Code**. If checked, when you generate code for an unsaved diagram, a prompt appears indicating the diagram has been modified and asking if you want to save the changes before generating code. The dialog box also contains a checkbox: Automatically save when generating code and do not show this message again.
    - **Show Validation success message**. If unchecked, then the confirmation dialog at the time of Validation will not be shown.)
    - **Enable Validation for Prompt Resources**. This preference is used for voice applications. If unchecked, then a validation check for missing prompts is not performed at the time of Validation.
    - **Interaction Process Diagram**. If unchecked, Composer will save Interaction Process Diagrams before publishing.
    - **Prompt to delete Published objects when Interaction Process Diagram is deleted**. If unchecked, Composer will attempt to delete any Published objects when an Interaction Process Diagram is deleted. If Composer is not connected to Configuration Server, object

deletion will not work.

2. Click Apply.

Colors and Fonts

1. Select **Appearance** under Composer Diagram.
2. Click **Change** and make selections to change the default font if you wish.
3. Click the appropriate color icon beside any of the following and make selections to change color:
    - **Font color**
    - **Fill color**
    - **Line color**
    - **Note fill color**
    - **Note line color**
4. Click **Apply**.

Connections

1. Select Connections under Composer Diagram.
2. Select a line style from the drop-down list:
    - **Oblique**
    - **Rectilinear**
3. Click **Apply**.

Pathmaps

1. Select **Pathmaps** under Composer Diagram.
2. Click **New** to add a path variable to use in modeling artifacts, or If the list is populated, select the check box of a path variable in the list.
3. Click **Apply**.

Printing

1. Select **Printing** under Composer Diagram.
2. Select **Portrait** or **Landscape** orientation.
3. Select units of Inches or Millimetres.
4. Select a paper size (default is Letter).
5. Select a width and height (for inches, defaults are 8.5 and 11; formillimeters, defaults are 215.9 and 279.4).
6. Select top, left, bottom, and right margin settings (for inches, defaults are 0.5; for millimeters, defaults are 12.7).
7. Click **Apply**

Rulers and Grid

You can make use of rulers and grids when creating diagrams. Rulers and grids can provide a backdrop to assist you in aligning and organizing the elements of your callflow diagrams.

1. Select Rulers and Grid under Studio Diagram.
2. Select or clear the **Show rulers for new diagram** check box (not selected by default).
3. Select ruler units from the drop-down list:
    - **Inches**
    - **Centimeters**
    - **Pixels**
4. Select or clear the Show grid for new diagrams check box (not selected by default).
5. Select or clear the Snap to grid for new diagrams check box (selected by default).
6. Type a value for grid spacing (for inches, the default is 0.125; for centimeters, the default is 0.318; for pixels, the default is 12.019).
7. Click **Apply**.

GAX Server

See GAX Server Preferences for voice applications.

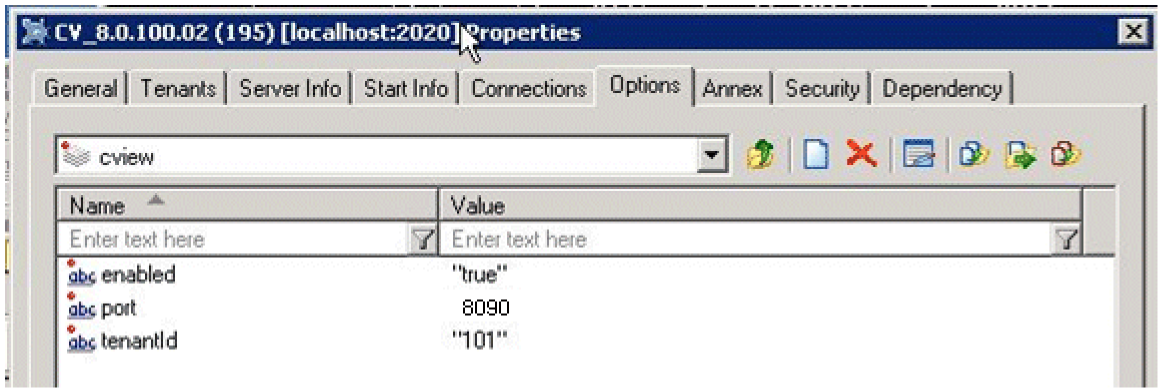
ORS Debugger Preferences

Select **Window** > **Preferences** > **Composer** > **Debugging** > **ORS Debugger**. ORS Debugger preferences are usually set during **post-Installation configuration**, when you first run Composer. A Debugger Cheat Sheet (**Help** > **Cheat Sheets** > **Composer** > **Routing Applications**) also provides detailed configuration instructions.

Setting Context Services Preferences

When working with Context Services blocks, you may wish to use online mode. In this mode, Composer fetches data from Universal Contact Server during design phase to help you configure the blocks. For example, Composer can fetch customer profile attribute names, extension attribute names, and so on. You can enable/disable this behavior in the Context Services Preferences page. If the Context Services capability is enabled at your site, set preferences as follows:

1. Go to **Window** > **Preferences** > **Composer** > **Context Services**.
2. Check the following box to specify online or offline mode when connecting to Context Services: **Connect to the Universal Contact Server when designing diagrams**. This enables the fields below.
3. Under **Universal Contact Server**, enter the server host name in your Configuration Database, which is the name (or IP address) of the Universal Contact Server. Also see the Runtime Configuration topic.
4. Enter the **Server Port number** for Universal Contact Server. **Note:** For the port number, open the Universal Contact Server Application object in your Configuration Database, go to Options tab, select the cview section, and the port option. Example settings are shown below.


CV_8.0.100.02 (195) [localhost:2020] Properties

General | Tenants | Server Info | Start Info | Connections | Options | Annex | Security | Dependency

cview

Name	Value
enabled	"true"
port	8090
tenantId	"101"

5. Enter the **Base URL** for the Context Services server (UCS).

The GVP Debugger passes all host, port, and base URL parameters to the VXML platform. It uses the parameters to make an url made of: `[http:// http://]<host-parameter>:<port-parameter>[/<base-url>`.

6. Under **Security Settings**, **Use secure connection**, select **never** or **TLS if Transport Layer Security** is implemented as described in the *Genesys 8.1 Security Deployment Guide*. Also see *Debugging Transport Layer Security*.
7. Select **Use Authentication** to require a user name and password when connecting to Universal Contact Server. If selected, enter the User and Password fields.
8. On the Context Services Preferences page, click the **Test Connection** button. Clicking should cause connection successful to appear. If not, check that Universal Contact Server is running and that the entered host/port values are correct. Other sources of error could be:

    - Base URL parameter value is incorrect

- UCS version is not 8.1 or higher

Note: Composer can successfully communicate with UCS at design stage whatever the UCS mode is (production or maintenance). However, UCS needs to be in production mode at runtime stage (when running Context Services SCXML or VXML applications, even when using GVP Debugger).

9. Under **Context Services object Validation,** select one of the following:
    - **No validation**
    - **Validate if connected**
    - **Validate**

SCXML File Preferences

To set SCXML file preferences:

1. Select **Window** > **Preferences** > **Composer** > **SCXML Files**.
2. Select the suffix that Composer should add to SCXML files. The default is scxml.
3. Select the encoding for the file. The encoding attribute in an SCXML document specifies the encoding scheme. The encoding scheme is the standard character set of a language. The SCXML processor uses this encoding information to know how to work with the data contained in the SCXML document. UTF-8 is the standard character set used to create pages written in English. Select from the following:
    - ISO 10646/Unicode(UTF-16LE) Little Endian
    - US ASCII
    - ISO Latin-1
    - Central/East European (Slavic)
    - Southern European
    - Arabic, Logical
    - Arabic
    - Chinese, National Standard
    - Traditional Chinese, Big5
    - Cyrillic, ISO-8859-4
    - Cyrillic, ISO-8859-5
    - Greek
    - Hebrew, Visual
    - Hebrew
    - Japanese, EUC-JP
    - Japanese, ISO 2022
    - Japanese, Shift-JIS
    - Japanese, Windows-31J
    - Korean, EUC-KR
    - Korean, ISO 2022
    - Thai, TISI
    - Turkish
4. Check the box to warn if no grammar is specified when validating SCXML files (not selected by default).
5. Click **SCXML Templates**.

SCXML Templates

Composer provides a set of predefined templates when writing SCXML code to create routing strategies. You can either start off from scratch in the SCXML editor or use one of the available templates as a starting point. Use this preference to create, edit, or remove templates as well as to import and export templates.

1. Select **Window** > **Preferences** > **Composer** > **SCXML Templates**.
2. In the resulting Templates dialog box, do one of the following:
    - To create a new template click **New**... The New Template dialog box opens for naming and describing the new template.
    - To edit an existing template, select its row and then click **Edit...**The Edit Template dialog box opens for editing.
    - To remove a template, select its row and click **Remove**. If you change your mind, click **Restore Remove**d,
    - To import a template, click the **Import...** button, navigate to the folder containing the SCXML file, select the file, and click **Open**.
    - To export one or more templates, select the row(s) and click the **Export...** button. In the Export Templates window, click the target location and click **Save**.
    - If you change your mind after editing a predefined template, click **Restore Defaults**.
3. Click **Source**.

Source

Source preferences are based on the XML Editor preferences. See that topic for more information.

Syntax Coloring

Syntax coloring preferences are based on the XML Editor preferences. See that topic for more information.

Security Preferences

Use the Security preferences page to specify the location of the Trust Store that holds security certificates.

Window > **Preferences** > **Composer** > **Security**

The certificates are used, for example, when Composer connects to Universal Contact Server with a secured connection. By default, Composer uses the Java Runtime Environment (jre) Trust Store. The default password for the Trust Store is `changeit`.

Time Zone Preferences

Composer displays all date/time elements in the user-preferred time zone with the time zone identifier. You can change the preferred time zone in **Window** > **Preferences** > **Composer** > **Context Services**.

Tomcat Preferences

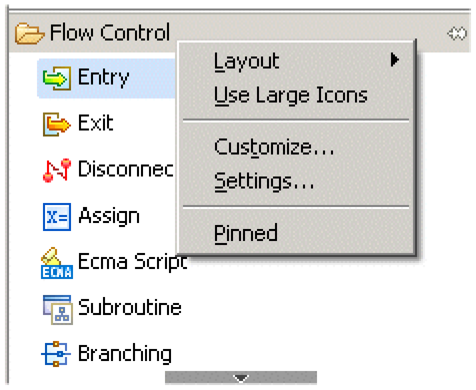
Select **Window** > **Preferences** > **Composer** > **Tomca**t Tomcat preferences are usually set during post-installation configuration, when you first run Composer. Detailed post-installation configuration instructions are provided in the Configure Tomcat and Debugger Settings Cheat Sheet (**Help** > **Cheat Sheets** > **Composer** > **Building Voice Application**s), and also in Callflow Post-Installation Configuration or Workflow Post Installation Configuration.

XML Preferences

You can also set XML File Preferences for both routing and voice applications: **Window** > **Preferences** > **XML** > **XML Files**. When specifying Encoding formats in the XML Preference page: encoding formats are applicable only for new File creation using the Template option: (**File** > **New** > **XML** > **XML File** > **Create XML File from an XML Template** > **Select XML Template**). This applies only to new XML, VXML, CCXML and SCXML files. Existing files within the Project will not get impacted.

Palette Group Menu

When creating a callflow or workflow in Composer or Composer Design perspective, a shortcut menu opens when you right-click on a palette title bar. The figure below shows an example:



The Palette Group menu contains the following items:

Layout	Allows you to specify how the blocks in this palette group should be displayed: • Columns • List • Icons Only • Detail
Use Large Icons	Allows you to increase the size of the icons representing the callflow or workflow blocks.
Customize	Opens a dialog box where you can change block names and descriptions, hide/unhide blocks from the palette, configure the block drawer to open upon Composer startup, and pin the block drawer open upon Composer startup.
Settings	Opens a dialog box where you can change the font, layout, and palette drawer options.
Pinned	Allows you to prevent a block drawer from closing when you switch to a different palette group.

Introduction to Routing Workflows

This section gives a high-level overview of creating SCXML-based routing strategies ("workflows") in Composer's integrated development environment.

What is a Routing Workflow?

What is Routing?

In the simplest terms, *routing* is the process of sending an interaction to a target, for example, sending an incoming telephone call to an agent. In practice, an interaction must undergo various types of processing between the time it arrives at the contact center and the selection and routing to the appropriate target. Each processing-point is an opportunity for some sort of processing to take place or for Universal Routing Server (URS) to make a decision based on the current situation—with the goal of getting the interaction delivered to the most appropriate target.

What is a Routing Workflow?

A routing workflow is a set of decisions and instructions that tells Universal Routing Server how to handle and where to direct interactions under different circumstances.

At any given processing-point in the workflow, only one of several possible outcomes can be optimal. Universal Routing Server uses the workflow instructions to determine which outcome is optimal and sends the interaction along a specified route accordingly.

There are various ways to create an SCXML-based workflow in Composer:

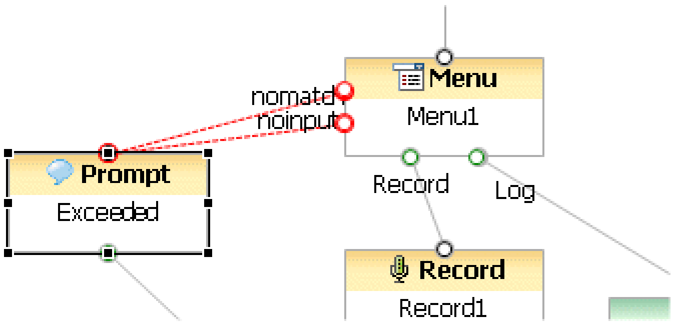
1. By working with blocks to create a workflow in Composer or Composer Design perspective.
2. By writing code in the SCXML editor (which may include using templates).
3. By Options importing an existing workflow.

Routing Blocks and Ports

Each processing-point is represented graphically in a workflow by a routing block, which has:

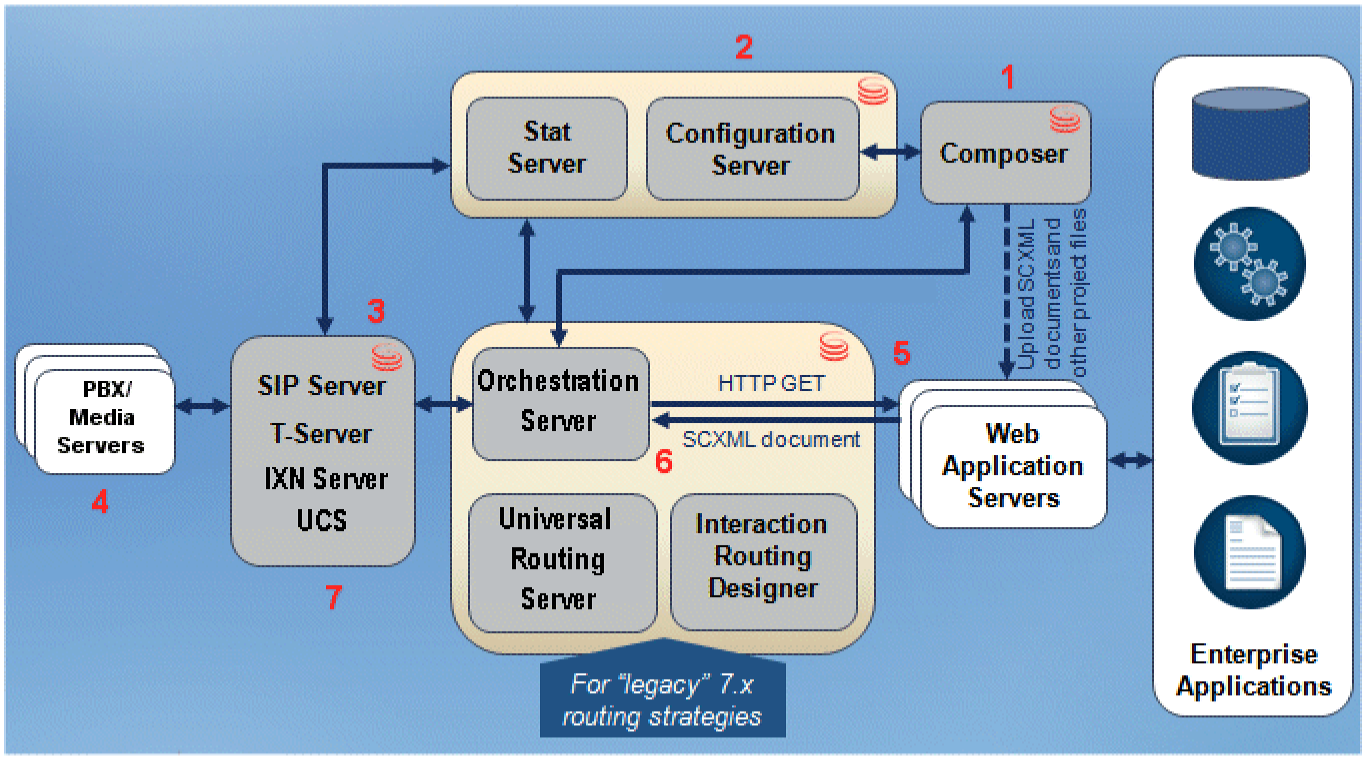
- One entry port
- One or more red exception ports
- One or more green exit ports.

The figure below shows how these ports appear for a block in Composer's workflow designer.


Menu
Menu1
nomatch
noinput
Prompt
Exceeded
Record
Log
Record
Record1

Architecture Diagram for Workflows

To give the "Big Picture" and show the various Genesys components used for SCXML-based workflows, below is a high-level diagram of the Universal Routing 8.x architecture.



The SCXML-based strategy creation and processing steps above are keyed to the numbers below.

1. The developer (or other persona) creates and tests the workflow application and manually deploys it on a Web application server.

    - For Web application server requirements related to Composer, see the *Composer 8.1 Deployment Guide*.
    - For developing SCXML applications and related resources that will be executed on the Genesys Universal Routing Server/ Orchestration Server platform, only specific Web application servers are supported. Refer to the *Orchestration Server 8.1 Deployment Guide* for more information.

The developer specifies Configuration Server objects as routing targets when creating the workflow application in Composer. For example, a routing target could be an agent (Person object) or Agent Group or an interaction can be sent to a queue. The developer may also create workflows that route based on the value of a Stat Server statistic or a statistic that you defined in Composer's Statistics Builder.

2. The developer or other persona configures:

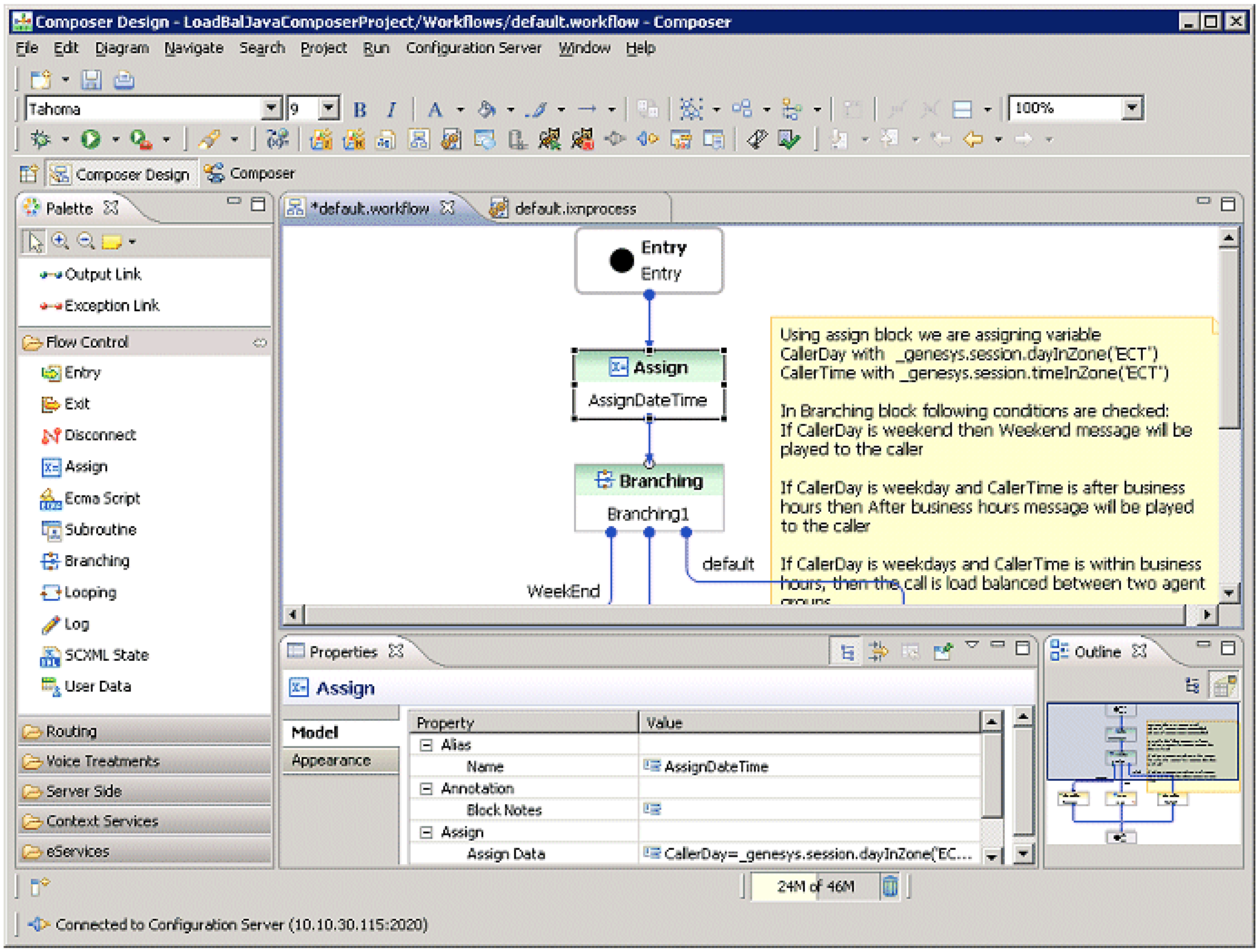
- Universal Routing Server (URS) and the Routing Point as described in the *Universal Routing 8.1 Deployment Guide.*
- Orchestration Server as described in the *Orchestration Server 8.1 Deployment Guide*.

Once this is done, the runtime life cycle of the routing logic can begin with a routing request.

3. URS gets an EventRouteRequest message from T-Server or another media server.
4. The request goes to the Routing Point. URS gets the URL of the Application Server to be contacted about SCXML strategy to be provided, which is a property of the Routing Point. After the URL is obtained, ORS issues the HTTP GET (or POST) request to the application server.
5. The application server executes the request and returns the SCXML document back to ORS.
6. Upon getting the response, ORS passes the received SCXML document to the SCXML engine. The SCXML engine runs the SCXML application, invoking ORS/URS services as needed.
7. ORS or URS issues a RequestRouteCall message to the media server.

Workflow Example and Palette

The figure below shows an example routing workflow in Composer Design perspective.



Using assign block we are assigning variable
CallerDay with _genesys.session.dayInZone('ECT')
CallerTime with _genesys.session.timeInZone('ECT')

In Branching block following conditions are checked:
If CallerDay is weekend then Weekend message will be played to the caller

If CallerDay is weekday and CallerTime is after business hours then After business hours message will be played to the caller

If CallerDay is weekdays and CallerTime is within business hours, then the call is load balanced between two agent

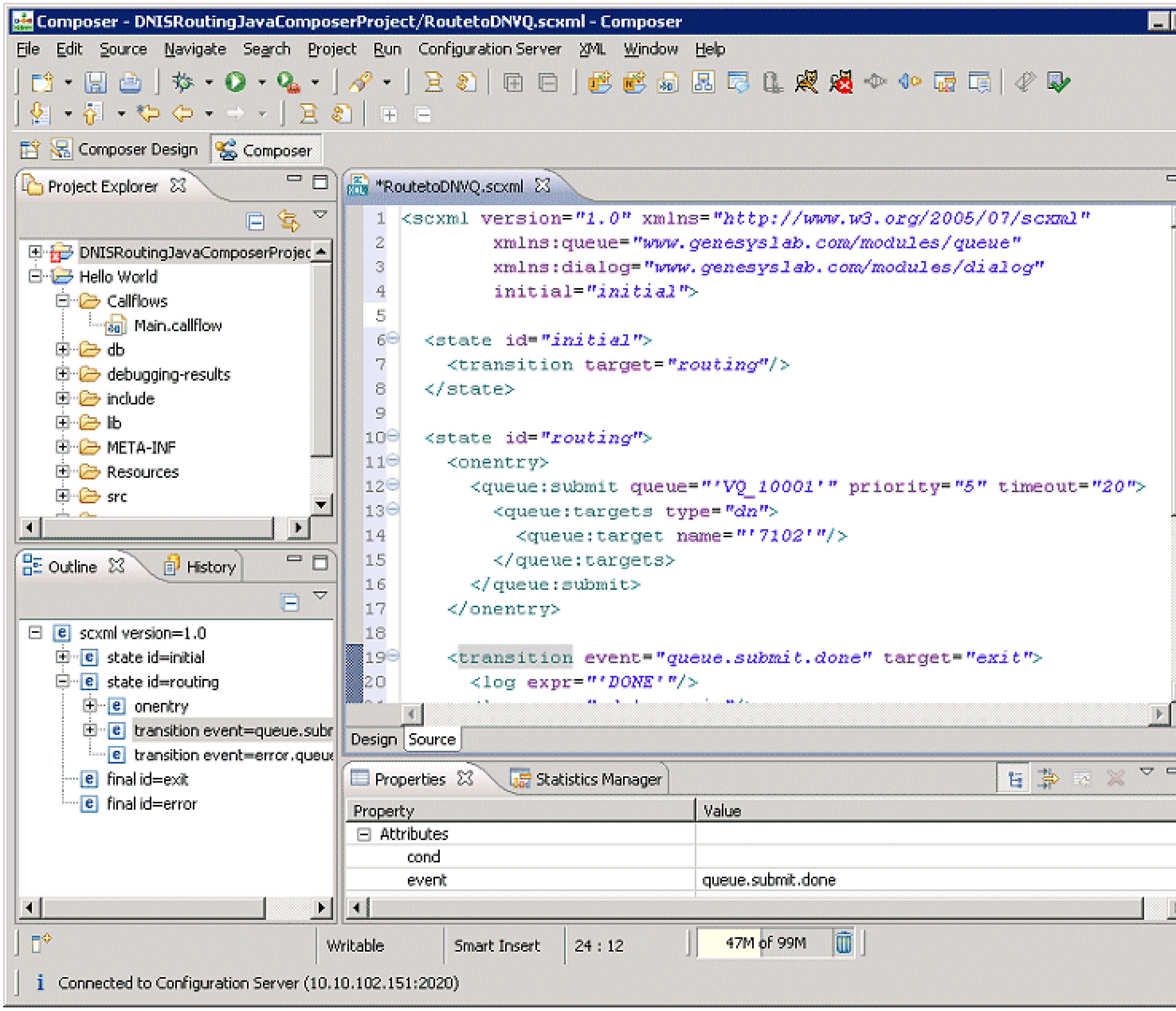
Property	Value
⊟ Alias	
Name	AssignDateTime
⊟ Annotation	
Block Notes	
⊟ Assign	
Assign Data	CallerDay=_genesys.session.dayInZone('EC...

Connected to Configuration Server (10.10.30.115:2020)

As shown above, in Composer Design perspective, the palette of blocks appears on the left. When creating workflows, you drag blocks from palette, drop them in the center canvas area, configure their properties in the lower Properties tab, and connect them.

SCXML File Editor

Composer also has an editor for those who like to write their own code. The figure below shows example strategy code in Composer's SCXML editor.



```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">

  <state id="initial">
    <transition target="routing"/>
  </state>

  <state id="routing">
    <onentry>
      <queue:submit queue="'VQ_10001'" priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
```

Property	Value
Attributes	
cond	
event	queue.submit.done

Writable   Smart Insert   24 : 12   47M of 99M

Connected to Configuration Server (10.10.102.151:2020)

- SCXML File Preferences control colors and other aspects of SCXML editing.
- For editing functionality, see Accessing the Editors and Templates.

- For sample strategies, see Universal Routing 8.1 SCXML Strategy Samples.

Sessions and Interactions

Orchestration Server introduces the concept of sessions. An understanding of these concepts is important to effectively use Routing functionality in Composer. Please consult the *Orchestration Server 8.1 Deployment Guide* for references to detailed technical information.

Composer generated SCXML applications make sure that an SCXML session is created whenever an interaction is pulled from an Interaction Queue and presented to the SCXML application. This behavior applies to:

- All new interactions received by Orchestration platform.
- Interactions that have been placed in the interaction by an SCXML application.
- New interaction created by Orchestration platform as a result of an action (e.g., E-mail Response and Create SMS blocks).

However, under certain circumstances, use of the SCXML State block or a hand-coded subroutine may result in a new interaction being created. In such cases, special care must be taken to process the new interaction since the default behavior is to take action on the original interaction that resulted in the session being created. One of the following approaches is recommended in such cases.

- If the Interaction ID is not known, retrieve the Interaction ID from the `_genesys.ixn.interactions[x].g_uid` list.
- Place the new interaction in a different queue using the Queue Interaction block. When the interaction is presented to the workflow connected to the queue, a new session is created, thereby all further actions are performed on the presented interaction.

Interaction Process Diagrams

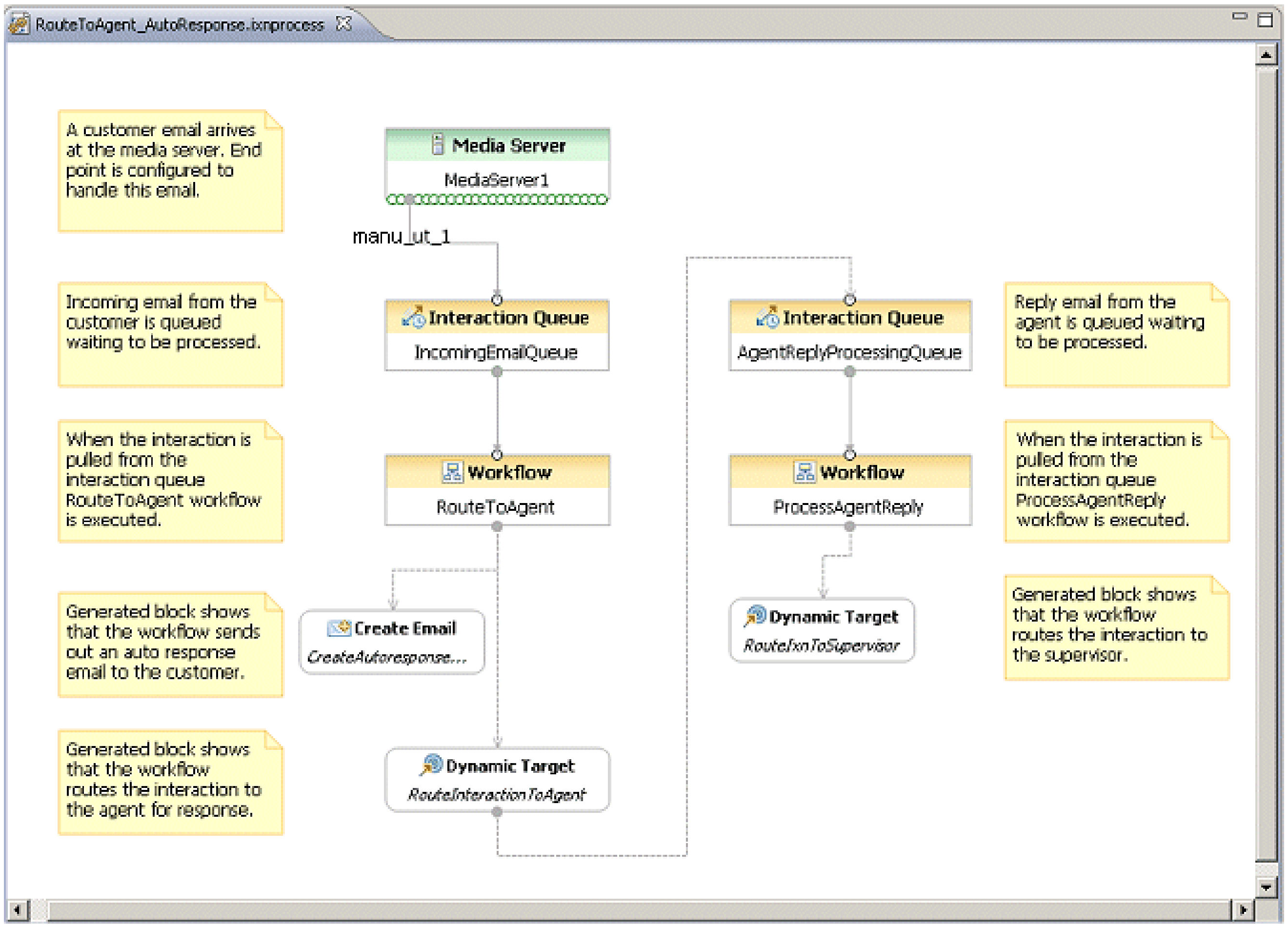
What is an IPD?

An interaction process diagram (IPD) provides a high level view of how multimedia interactions flow through various components like media servers, interaction queues, workbins, and workflows.

Starting SCXML Page

In addition, an IPD functions as the starting SCML page for both voice and multimedia interactions and results in using the same session across the entire interaction process. It also updates objects in Configuration Server and activates the linkages specified in the IPD.

- When used for voice interactions or interaction-less processing, an IPD contains a single Workflow block.
- When used for multimedia interactions, an IPD contains at least one and possibly multiple Media Server, Interaction Queue, and Workflow blocks. The example shown below is available as a template (Routing After Sending Autoresponse) when you create a new Project.


RouteToAgent_AutoResponse.ixnprocess
A customer email arrives at the media server. End point is configured to handle this email.
Media Server
MediaServer1
manu_ut_1
Incoming email from the customer is queued waiting to be processed.
Interaction Queue
IncomingEmailQueue
When the interaction is pulled from the interaction queue RouteToAgent workflow is executed.
Workflow
RouteToAgent
Generated block shows that the workflow sends out an auto response email to the customer.
Create Email
CreateAutoresponse...
Generated block shows that the workflow routes the interaction to the agent for response.
Dynamic Target
RouteInteractionToAgent
Interaction Queue
AgentReplyProcessingQueue
Reply email from the agent is queued waiting to be processed.
Workflow
ProcessAgentReply
When the interaction is pulled from the interaction queue ProcessAgentReply workflow is executed.
Dynamic Target
RouteIxnToSupervisor
Generated block shows that the workflow routes the interaction to the supervisor.

An IPD used for multimedia interactions:

- Visually presents and directs the flow of interactions through various processing blocks (media servers, queues, and workflows).
- Defines what happens to customer multimedia interactions from the point of arrival at your contact center to the point of completion.
- Can implement the procedures used by agents, supervisors, quality assurance, and other personnel in your company to accomplish your company's business objectives related to incoming multimedia interactions.

An IPD also enables visual configuration of the associated Configuration Server objects. When connected to Configuration Server, you trigger the actual Configuration Database update by using Composer's Publish operation. Publishing causes Composer to push out relevant subsets of the configuration information to Configuration Server. Here platform components can query for the information and use it for processing interactions.

SCXML File Creation

Composer expects IPDs (*.ixnprocess files) to be present in the Interaction Processes folder. This folder is accessible from the Project Explorer. The code generation step creates SCXML files for the following types of diagrams:

- Interaction process diagram: One SCXML file is created for:
    - Each Interaction Queue block present in the diagram with the name IPD_<Diagram Name>_<Interaction Queue Block name>.scxml (Multimedia interaction scenarios).
    - Each Workflow block present in the diagram, that is not connected to an Interaction Queue block with the name IPD_<Diagram Name>_<Workflow Block name>.scxml (voice or interaction-less scenarios). Each of these files represents an entry point into this application and can be specified in the EnhancedRoutingScript object. This Script object should never point directly to a workflow or sub-workflow SCXML file.
- Workflow diagram, sub-workflow diagram: Both types of files have the .workflow extension. Code generation creates an SCXML file for each diagram with the same name as the diagram name.

The IPD SCXML file includes one or more workflow SCXML files which, in turn, can contain a hierarchy of sub-routine SCXML files. Orchestration Server combines all these related files into a single document and then executes the resulting combined SCXML document. Due to this reason, the line numbers mentioned in Orchestration Server logs may not match the line numbers in individual Composer SCXML files. See the Orchestration Server documentation for information on how to obtain access to this merged document.

Starting a New IPD

See Starting a New IPD when creating a routing application.

Executing an IPD

The following Genesys servers work together to execute an interaction processing diagram:

- For multimedia interactions, handles queue processing, delivery of interactions to selected resources, and invoking multimedia services. For information on Interaction Server, start with *eServices/ Multimedia 8.1 Deployment Guide*.
- (ORS) interprets the top-level SCXML document created as a result of the interaction processing diagram. For information on ORS, start with *Orchestration Server 8.1 Deployment Guide*.
- Universal Routing Server provides the Genesys Functional Modules (Queue, Interaction, Voice Interaction, Dialog, Resource), which are described in the Orchestration Server Documentation Wiki: http://developerzone.genesyslab.com/wiki/index.php?title=Orchestration_Server

ORS sends requests to URS, invoking actions from these modules based on its SCXML interpretation. URS then determines available resources (which may be another queue or the final target, such as an agent) where interactions can be delivered to and returns responses back to ORS with the resultant action information. For information on Universal Routing Server, start with *Universal Routing 8.1 Deployment Guide*.

Creating Routing Applications

This section contains step-by-step instructions for creating routing applications, both in the workflow designer and in the editor.

IPD Planning & Preparation

This topic discusses preparation and planning for creating an interaction process diagram (IPD) that will process multimedia interactions. An IPD for voice interactions contains only a single workflow block. When planning an IPD for multimedia processing, start by considering the basic stages in the interaction life-cycle. You can then design an IPD that encompasses all stages, just one stage, or multiple stages. This topic presents four basic stages, which are especially applicable to e-mail processing, but could apply to other media types as well. The stages are:

1. Pre-Route
2. Route-to-Agent
3. Review
4. Pre-Send

Each stage is summarized below. **Pre-Routing Stage** The main activities in the pre-routing stage of e-mail handling can potentially include:

- Determining whether an e-mail has already been processed by Genesys. This can be accomplished via the absence or presence of an Interaction Subtype Business Attribute assigned by Interaction Server.
- Classifying e-mails based on content analysis, which can assign a category code. Once a category code is assigned to an e-mail, you can configure other types of processing to occur based on the category code.
- Screening e-mails for certain words or patterns of words. Once a screening rule match occurs, you can configure other types of processing based on the match.
- Sending an acknowledgement and/or automatic standard response to the customer originating the e-mail.
- Determining the agent (if any) who previously handled the interactions related to this service.

Route-to-Target Stage This may or may not be an agent target. For example, the e-mail may be:

- Sent to a queue for submittal to other routing strategies and further processing.
- Sent to a queue for failed interactions.
- Forwarded outside the contact center to an expert with the expectation of getting a response back.
- Redirected to another agent without the expectation of getting a response back.
- Routed to an agent target for construction of a response.

Review Stage The reviewer could be a manager, supervisor, or QA Person. You may want to have two different types of quality assurance review:

- A supervisor review that checks the skills of the agent who constructed the response.
- An analysis that performs a sanity check; for example, to prevent sending out a bank account password in an interaction or to screen interactions for inappropriate language.

Pre-Send Stage The cycle of going from queue to routing workflow to queue can continue until the interaction reaches some final outbound queue. The pre-send stage performs last-minute quality checking and allows for attaching additional information to interactions when needed.

IPD Preparation

This section summarizes the preparatory steps for creating an IPD prior to actually creating, configuring, and placing blocks in Composer. It also describes the Configuration Database and Universal Contact Server Database objects that must exist first so they can be selected from Composer blocks.

1. Determine the interaction life cycle at your contact center.
2. Determine the media server, which will be referenced by the Media Server block. Check that the required Endpoints have been defined.
3. Determine which Composer blocks will be used in routing workflows reference by the Workflow block to perform the various processing required at each stage of interaction processing as described in the IPD Planning topic.
4. For each interaction processing stage, assign a name to the required IPD.
5. If you plan on having multiple IPDs linked via workflows, name the queues that will connect the workflows contained within each IPD.
6. Determine the selection criteria for extracting interactions from queues (View property in the Interaction Queue block). For example, you may wish to extract certain interaction types earlier than other interaction types.
7. Create the required Knowledge Manager objects, such as categories and standard responses. For more information, see the *eService/Multimedia 8.0 Knowledge Manager Help*.
8. Create the required Configuration Manager/Genesys Administrator objects, such as media server Applications, Skills, Persons, Agent Groups, Places, Place Groups, and Business Attributes, just to mention a few. For more information, see the *Framework 8.1 Configuration Manager Help*.
9. Optionally, map Context Services attributes to Configuration Server Business Attributes. Once this is done, you can select a Business Attribute DB ID for a value in many Context Services block fields.
10. Create the routing workflow(s) that will perform the specialized processing tasks, which will be referenced by Workflow block(s).

Starting SCXML Page

The starting SCXML page for any Composer routing project is the SCXML page generated from an interaction process diagram (IPD) and not the SCXML page generated from a workflow. IPD SCXML pages internally include workflow SCXML pages. An IPD is therefore required for both voice and multimedia workflows. When an IPD diagram is Published, Composer sets the URL of the starting SCXML page in the resultant Script object of type Enhanced Routing.

When executing an IPD, the Orchestration Platform "merges" the IPD SCXML page and any included workflow SCXML pages to create a single page and then executes that page.

Note: One Composer Project may contain multiple IPDs. Both Java Composer Projects and .NET Composer Projects support IPDs.

Creating a New Routing Project

When building any application in Composer, you first need to create a Project. A Project for a routing application contains all the workflows and other files for your application. It also automatically creates the required interaction process diagram. By associating a workflow or other routing application with a Project, you enable Composer to manage all the associated files and resources in the Project Explorer.

Creating a Project for a Workflow

Before creating a new Project, you may wish to review Your First Application: DNS Routing.

To create a new Project for a workflow:

1. Click the down arrow next to the button above the Project Explorer.
2. Select **Java Composer Project**.
3. In the Project dialog box, type a name for your Project.
4. 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.
5. Select the Project type:
    - **Integrated Voice and Route**. Select to create a Project that contains both callflows and workflows that interact with each other; for example a routing strategy that invokes a GVP voice application.   For more information on both voice and routing applications, see What is GVP and How Do Voice Apps Work and What Is a Routing Strategy, respectively. This selection grayed out if you have not enabled Composer's routing functionality as described in the Hiding File Types topic.
    - **Voice**. Select to create a Project associated with the GVP 8.x. This type of Project may include callflows, and related server-side files. For more information on this type of Project, see topic, What is GVP and How Do Voice Apps Work.
    - **Route**. Select to create a Project associated with the URS 8.x SCXML Engine/Interpreter. For more information on this type of Project, see topic, What Is a Routing Workflow. This selection grayed out if you have not enabled Composer's routing functionality as described in the Hiding File Types topic.
7. Click **Next**.
8. If you want to use templates, expand the appropriate Project type category and select a template for your application. Templates are sample applications for different purposes.

Note: The Route After Sending Autoresponse Project template demonstrates the use of an interaction processing diagram.   If you want to start from scratch, choose the Blank Project template and click Next.

9. Select the default locale and click **Next**.
10. Optional. If using the GVP ICM Adapter in a VoiceXML application, select the **Enable ICM** checkbox to enable integration. When checked, ICM variables will be visible in the Entry block. See the ICM Interaction Data block for more information.
11. Click **Finish**.

Composer now creates your new Project. Your new Project folder and its subfolders appear in the Project Explorer. The canvas area shows default.ixprocess tab for the interaction process diagram.

Continue with the Block Palette Reference for interaction process diagrams.

Creating the IPD

Note: Composer automatically creates a default IPD as part of new Project creation. You can also create an IPD on demand using the main toolbar button. This topic presents typical high-level steps for creating:

- IPDs for Voice Workflows
- IPDs for Multimedia Workflows

Notes:

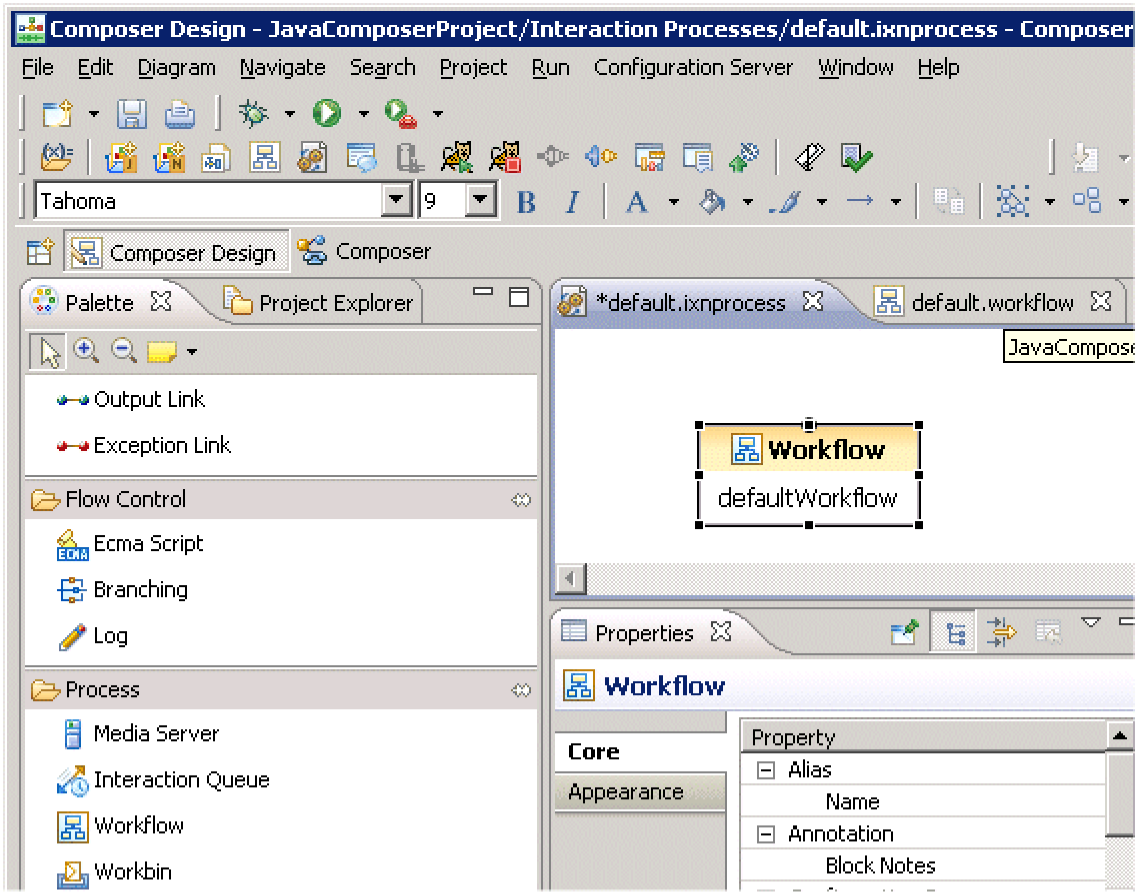
- IPDs are not stored in Configuration Server. Your Workspace (Interaction Processes folder in Project Explorer) is the only location where these diagrams exist (and, of course, in source control if enabled).
- If a Composer Project contains a folder at `include/user`, then any files with extension .js will be included in the generated SCXML. This allows you to write custom ECMAScript and include it in the application.

Creating IPDs for Voice Workflows

An IPD for a voice workflow contains a single Workflow block. After creating a new Project (which automatically creates an IPD with a Workflow block), the typical high level steps are as follows:

1. Click the default.ixprocess tab on the canvas that will contain the IPD.

Note: You can change the name of an interaction process diagram file by right-clicking the file name in the Project Explorer and selecting **Rename**. This brings up a preview dialog. After accepting the change, the default.ixprocess tab reflects the name change.



2. Double-click the Workflow block to display its properties in the Properties tab below.

3. Opposite Resource, click under Value to display the button.

4. Click the button to open the Select Resource dialog box and select the empty workflow diagram. If you have not changed the name, this will be default.workflow.

Note: There is no need for a Media Server block or an Interaction Queue block when creating an IPD for a voice workflow.

5. Click the default.workflow tab on the canvas as shown above and use the designer to develop your workflow diagram. This includes saving and validating. For more information on developing a workflow, see the Example Diagram section in the Creating a New Workflow topic.

6. Validate the code and Publish your IPD diagram.

7. Generate the code by selecting**Diagram** > **Generate Code**, or by clicking the Generate Code icon on the upper-right of the Composer main window when the canvas is selected. Check the **Problems** tab for errors and fix any problems. If code generation succeeds, click **OK** at the confirmation dialog box. The SCXML code is generated in the src-gen folder. The generated file name in the src-gen folder will follow the format:

<ipd_diagram_name>_<workflow_block_name>.scxml Note: One file per voice/interaction-less (see Interaction ID) workflow block will be generated.

8. Debug the workflows.
9. Deploy the Project.

Note: Composer Diagram Preferences includes the following:

- **Prompt to Save before Publishing Interaction Process Diagram**.
- **Prompt to delete Published objects when Interaction Process Diagram is deleted**.

Creating IPDs for Multimedia Workflows

After creating a new Project (which automatically creates an IPD with a Workflow block), the typical high level steps for creating an IPD for multimedia interactions are as follows:

1. Click the default.ixprocess tab on the canvas that will contain the IPD.
2. Add a Media Server block that has at least one Endpoint defined to the canvas by double-clicking or dragging and dropping. Configure the Media Server block properties in the Properties view.
3. Add an Interaction Queue block to the canvas by double-clicking or dragging and dropping. Configure the Interaction Queue block properties in the Properties view. This includes defining at least one interaction queue view.
4. Connect the Media Server Endpoint to the Interaction Queue block.
5. Click the default.workflow tab and use the designer to develop your workflow diagram.
6. Click the default.ixprocess tab and open the Workflow block. Use the Resource property to point the workflow diagram you just developed.
7. Connect the view from the Interaction Queue block to the Workflow block.

The subsequent steps of adding additional blocks to complete the IPD or linking IPDs with workflows are user-defined and reflect your company's business logic. This includes saving and validating. For example, you may choose to continue processing interactions in this IPD by repeating the process of sending interactions from workflows back to queues, using views to extract the interactions, and then sending the interactions for further specialized processing by other workflows.

8. Validate the code and Publish your IPD diagram.
9. Generate the code for the IPD and workflow diagram All generated files go into the src-gen folder of the Project.
10. Click the tab on the canvas that contains the IPD and generate the code. The generated file name in the src-gen folder will follow the format: <ipd_diagram_name>_<interactionqueue_block_name>.scxml

Note: One file will be generated for each interaction queue. Functionally these files will be equivalent. Generating one file per queue makes it easier to configure the application URL manually. As the developer, you will know the interaction queue you have submitted the interaction to and therefore can easily identify the correct SCXML page.

11. Debug the workflows.
12. Deploy the Project.

Manually execute any other configuration steps in Configuration Server using Genesys Administrator or Configuration Manager. For example, you must hook up your mailbox to send e-mails to the new end point defined in your e-mail server.

Generated Files

Multiple top level SCXML files are generated, one for each Interaction Queue connected to the Media Server block Endpoints. This allows different start states and different sessions for different interactions pulled from different Interaction Queues connected to the media servers. File names for the generated files use the following format:

- For IPDs that do not contain Interaction Queue blocks: IPD_<ipdname>_<Workflow block name>.scxml
- For IPDs that use interaction queues: IPD_<ipdname>_<InteractionQueue block name>.scxml

Each top level SCXML file for an IPD includes:

- All workflows (generated SCXML files) referred to by the Workflow blocks in the IPD.
- All workflows (generated SCXML files) referred to indirectly via Interaction Queues in other IPD’s.

The SCXML file inclusion in the top-level document is done using <xi:include>. The workflow file name without the .scxml extension is used for the _prefix attribute.

Creating a New Workflow Diagram

Cheat Sheets

Composer provides a cheat sheet to walk you through the steps for building a routing strategy.

- In the Welcome Screen (**Help** > **Welcome**), click the Composer link and select the Create a Routing Strategy tutorial. It will also describe the steps for how to simulate calls.
- If you are already inside the workbench, access the same cheat sheet from the Menu bar at the top by selecting **Help** > **Cheat Sheets** > **Composer** > **Routing Strategy**.

Adding a Workflow Diagram to a Project

To add a new workflow diagram to an existing Composer Project:

1. Click the button on the main toolbar to create a new workflow .
    - Or use the keyboard shortcut: Ctrl+Alt+R.
    - Or select **File** > **New** > **Workflow Diagram**.
    - Or right-click the Workflows folder in a Project and select **New** > **Other** > **Workflow Diagram**.
2. In the wizard, select the tab for the type of the workflow. There are two main types of workflows in Composer represented by wizard tabs:
    - **Main Workflow**: Used for the main application where the call will land or be transferred to from another application.
    - **Subworkflow**: Used for modularizing your applications. It is useful for structuring large applications into manageable components.
3. Select either **Main Workflow** or **Subworkflow**.
4. Select the type of diagram.
5. Click **Next**.
6. Select the Project.
7. Click **Finish**.
8. Create the workflow. See Your First Application "DNIS Routing" on how to get started.

Note: The condition expression for event-related properties in interaction process (IPD) and workflow diagrams are not XML-escaped when generating the SCXML code. For more information, see Troubleshooting ORS Compile Errors and Non Escaped Special Charcters.

Using the SCXML Editor

The Composer SCXML editor is embedded/integrated within the user interface and are made available to you whenever an .scxml file is created or accessed within Composer.

Creating a New Project

Follow the steps below after you have created a Project.

1. Switch to Composer perspective.
2. Click the File menu and select **New** > **SCXML File**. The Create New SCXML File dialog box opens.
3. Select the Project.
4. Name the file. You now have two choices:

    - If you do not wish to use a template, click **Finish**.
    - To use a template, click Next, Use SCXML Template, select the template, and click Finish.

The Workflows folder in the Project Explorer shows the name of the file under your Project. Composer displays the SCXML Editor.

5. Create the code.

Open an Existing or Imported File

The SCXML editor also opens whenever you open an existing .scxml ile, whether previously created as described above, or previously imported into Composer. Open an existing file as follows:

- Select **File** > **Open File** and navigate to the file to open, OR

Open a Project's `src` or `src-gen` folder in the Project Explorer, then double-click the file to open it in the editor.

Using SCXML Templates

To create a new SCXML file for an existing Project:

1. Select from the menu: **File** > **New** > **Other**
2. Select from Wizard: **Composer** > **Other** > **SCXML file**.
3. Select the Project name for the new file.
4. Type a file name for your new file and click **Next.** The Select SCXML Template dialog box opens. The **Use SCXML Template** check box is selected.
5. Select a template to preview.
6. Click **Finish**. The SCXML editor opens with your new file.

Also see Project Templates.

Your First Application: Routing Based on DNIS or ANI

The steps below lead you through creating a simple routing strategy workflow for voice interactions. This workflow routes incoming calls based on the number dialed by the customer (DNIS) English-speakers dial one number; Spanish-speakers dial another number. Assume this number is attached to the interaction when it arrives at the contact center.

Note: The same type of Composer configuration could also be used to route incoming calls based on the originating phone number (ANI).

Starting the Workflow

After creating a new Project in Composer Design or Composer perspective:

1. Click the button on the main toolbar to create a new workflow and continue with step 2. Alternatives:
    - Select **File** > **New** > **Workflow Diagram** or select **File** > **New** > **Other**. In the New dialog box, expand **Composer** > **Diagrams**. Select **Workflow Diagram** and click **Next**. Continue with step 2.
    - Or use the keyboard shortcut: Ctrl+Alt+R and continue with step 2.
2. In the Main workflow tab, select **Empty Diagram** and click **Next**.
3. Select the parent Project.
4. Name the diagram (must have an extension of `.workflow`) and click **Finish**. The `Workflows` folder in the Project Explorer shows the name of your diagram under your Project.
5. Select the `Workflows` folder in the Project you just created.
6. Build the diagram as described below.

Creating the Workflow Diagram

For general guidelines on placing, configuring, and connecting blocks, see the Using the Designer topic.

1. Connect to Configuration Server. You can also use the keyboard shortcut: Alt+I+C.
2. Create a new project called "DNIS_Routing."
3. Add the following blocks from the Palette to the canvas area: Entry, Branching, Target (add two), and

Exit.

4. Use the Link tool to connect the Entry block to the Branching block.

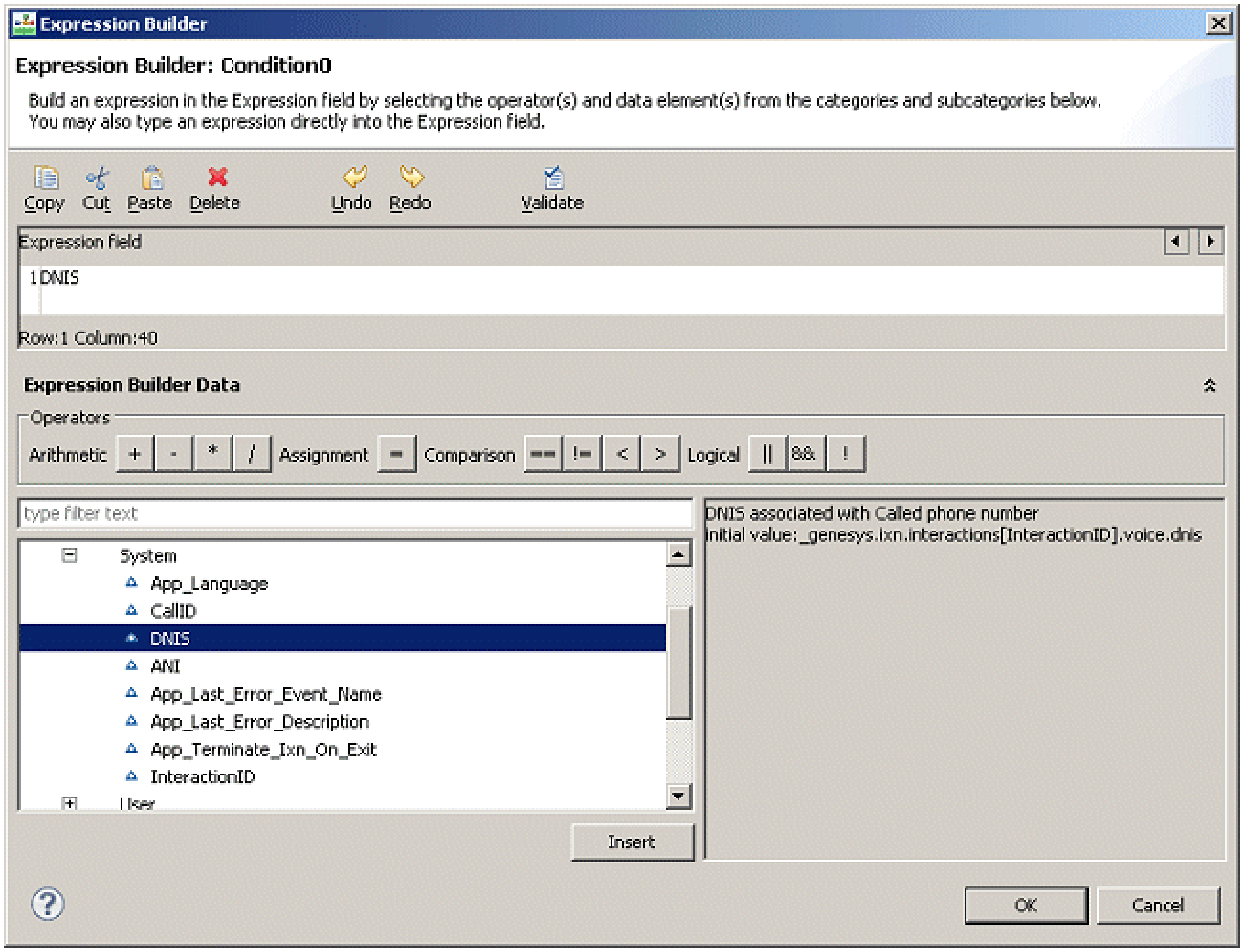
Typically, you start by segmenting incoming interactions to take different paths in the workflow. For example, you could segment by date, time of day, day of week, number dialed (DNIS), or originating number (ANI), just to mention a few examples. You could also segment based on a logical expression that you create in Expression Builder. You can use the Branching block for this purpose as described below.

1. Select the Branching block to cause the lower Properties tab to show the fields associated with the block. An alternative method is to right-click the Branching block and select **Show Properties View** from the shortcut menu.
2. In the Properties tab, opposite the**Conditions** field, click under the **Value** column. This brings up the button.
3. Click the button. This brings up the Branching Conditions dialog box.
4. In the Branching Conditions dialog box, select **Add**. Condition0 appears under **Node Name**.
5. Change `Condition0` to `8004662809`.
6. Click opposite `8004662809` under **Expression**. This brings up the button.
7. Click the button to bring up Expression Builder.

Using Expression Builder

We will now define two sample expressions in Expression Builder. In the case of the Branching block, the expressions will define the branching conditions that will cause interactions to take different paths in the strategy.

1. Click the button on the right to expand Expression Builder.
2. Expand **Workflow variables** followed by **System**.
3. Double-click **DNIS**. Expression Builder appears as shown below.



Expression Builder: Condition0

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below. You may also type an expression directly into the Expression field.

Expression field

1 DNIS

Row:1 Column:40

Expression Builder Data

Operators

Arithmetic + - * / Assignment = Comparison == != < > Logical || && !

type filter text

System
App_Language
CallID
DNIS
ANI
App_Last_Error_Event_Name
App_Last_Error_Description
App_Terminate_Ixn_On_Exit
InteractionID
User

DNIS associated with Called phone number
initial value: _genesys.ixn.interactions[InteractionID].voice.dnis

4. Continuing with this example, note that "DNIS" appears to appear under Expression field opposite 1. ANI and DNIS correspond to URS functions getDNIS() and getANI(). These functions can be used in a call or workflow to read the DNIS and ANI of the call. ANI is the originating phone number (user name of the calling SIP phone). DNIS is the number that the user dialed (provisioned number for the application, or "dialog" if you're making the call through the debugger).

5. Opposite **Operators**, click the button for the equal sign (=). data.DNIS= now appears under Expression field.

6. Type a single quote after the equal sign.

7. Type the 800 number. For this example, use `8004662809`.

8. Type a single quote after the number. Expression field now shows `data.DNIS='8004662809'`.

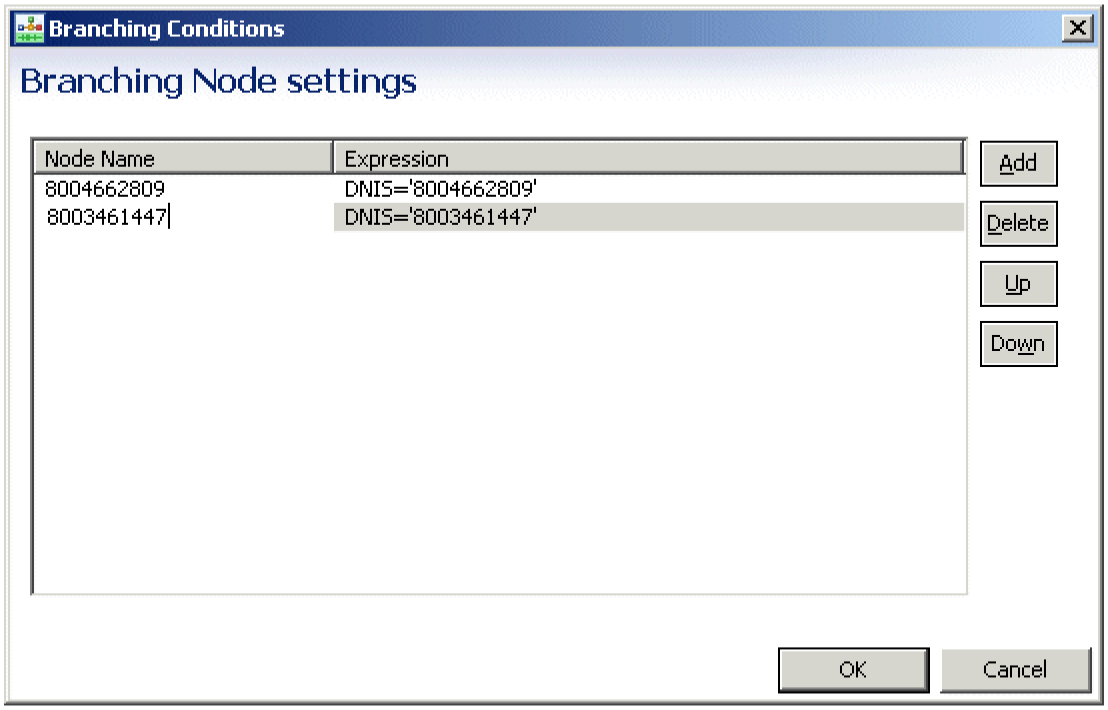
9. Click the button to validate the expression. No syntax error found appears above.

10. Click **OK** to close the Expression Builder dialog box and return to the Branching Conditions dialog box.

11. In the Branching Conditions dialog box, select **Add**. This time `Condition1` appears under **Node Name**.

12. Change `Condition1` to `8008361447`.

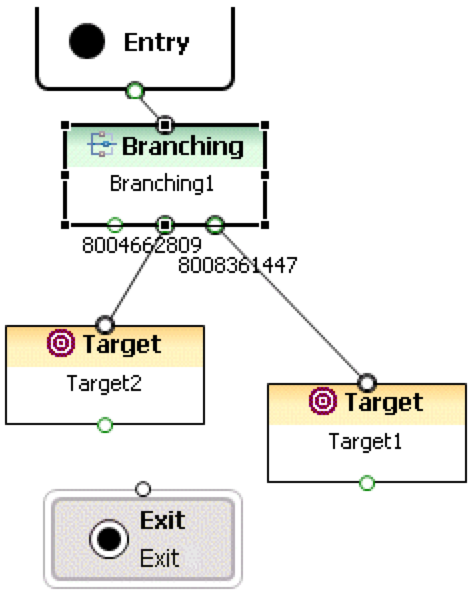
13. Click opposite 8008361447 under **Expression**. This brings up the ⋯ button.
14. Click the ⋯ button to bring up Expression Builder.
15. Repeat the Expression Builder steps to add the second expression: `DNIS='8008361447'` and click **OK**. The Branching Conditions dialog box now appears as shown below.


Branching Conditions

Branching Node settings

Node Name	Expression
8004662809	DNIS='8004662809'
8003461447	DNIS='8003461447'

16. Click **OK** in the Branching Conditions dialog box. The Branching block now shows three ports.
    - The second and third ports correspond to the conditions defined in the Expression Builder.
    - As described ahead, the first port could be used for default routing or another purpose.
17. Connect the second port to the Target block below it.
18. Connect the third port to the Target block below it. The routing strategy diagram (8.0.3) now appears as shown below.

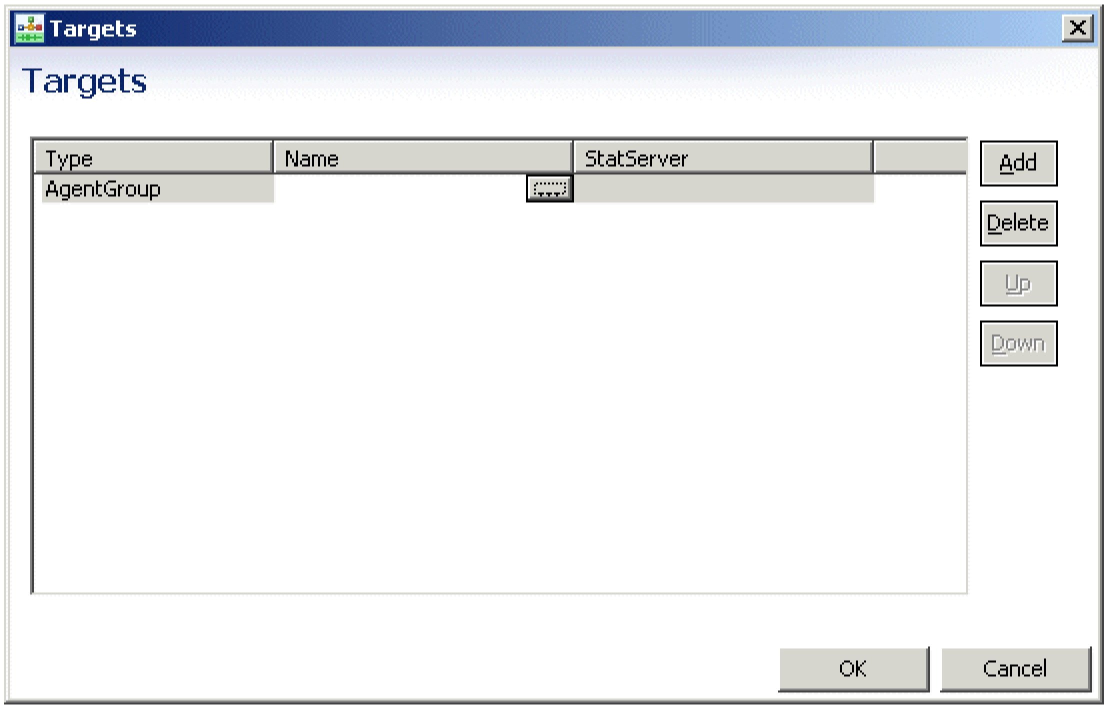


19. Save the diagram as it exists so far by selecting **File** > **Save**.

Target Selection

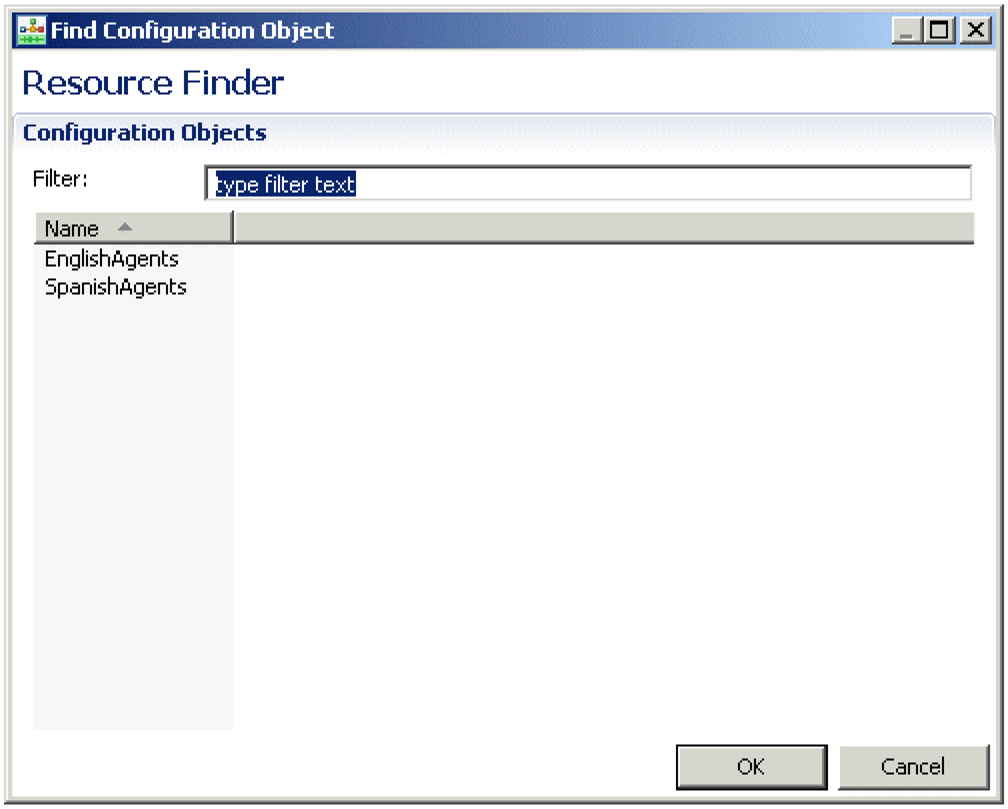
This section describes how to configure the Target blocks in our example strategy diagram. Targets refers to routing target objects that exist in your Configuration Database. For example: **Agent**, **Agent Group**, **ACD Queue**, **Place**, **Place Group**, **Route Point**, **Skill**, or **Variable**. As described below, you select the target type in the Targets property.

1. Click a Target block to cause the lower Properties tab to show fields. An alternative method is to right-click the block and select **Show Properties View** from the shortcut menu.
2. In the Properties tab, opposite the **Name** property, type `EnglishAgents`. The name must start with an underscore or a letter.
3. Click under **Value** opposite the Targets property to display the button.
4. Click the button. The Targets dialog box opens.
5. Click **Add** in the Targets dialog box.
6. Click under Type to display a down arrow.
7. Click the down arrow and select the target type. Available selections are: **Agent, AgentGroup, ACDQueue, Place, PlaceGroup, RoutePoint, Skill,** or **Variable**.
8. Select **AgentGroup**. **AgentGroup** appears under **Type**.
9. Click under the **Name** field to display the button.



Type	Name	StatServer
AgentGroup		

10. Click the button to bring up the Resource Finder. Targets of type **AgentGroup** appear for selection. An example is shown below.

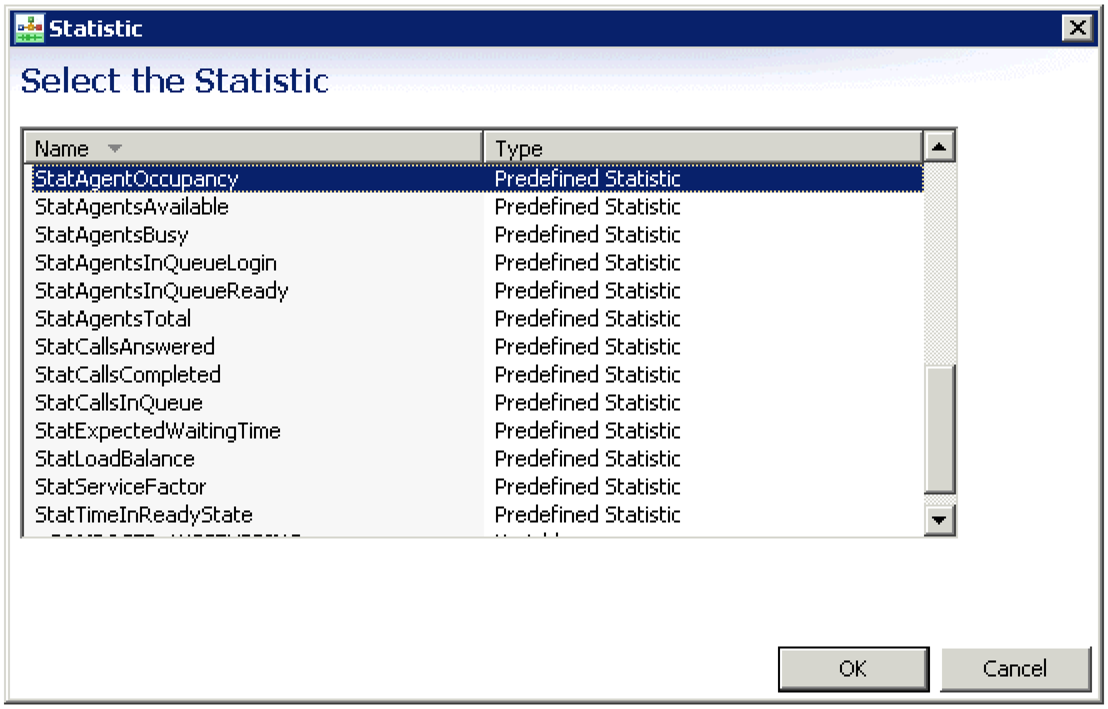


11. Select a routing target. In this case, select **EnglishAgents** and click **OK**.
12. Click **OK** in the Targets dialog box.

Routing Based on the Value of a Statistic

You have the option of instructing Universal Routing Server to use the value of a statistic during target selection. For example, you may wish to route to an agent who has been in a "Ready" state for the longest period of time.

1. If you have not already done so, make sure the Configuration Server preference is set to control whether or not to create Router predefined statistics when connecting to the Configuration Server.
2. In the Properties tab, opposite the **Statistic** property, click under the **Value** column to display the button.
3. Click the button to open the Statistics dialog box.



Name	Type
StatAgentOccupancy	Predefined Statistic
StatAgentsAvailable	Predefined Statistic
StatAgentsBusy	Predefined Statistic
StatAgentsInQueueLogin	Predefined Statistic
StatAgentsInQueueReady	Predefined Statistic
StatAgentsTotal	Predefined Statistic
StatCallsAnswered	Predefined Statistic
StatCallsCompleted	Predefined Statistic
StatCallsInQueue	Predefined Statistic
StatExpectedWaitingTime	Predefined Statistic
StatLoadBalance	Predefined Statistic
StatServiceFactor	Predefined Statistic
StatTimeInReadyState	Predefined Statistic

You can select from the following statistics:

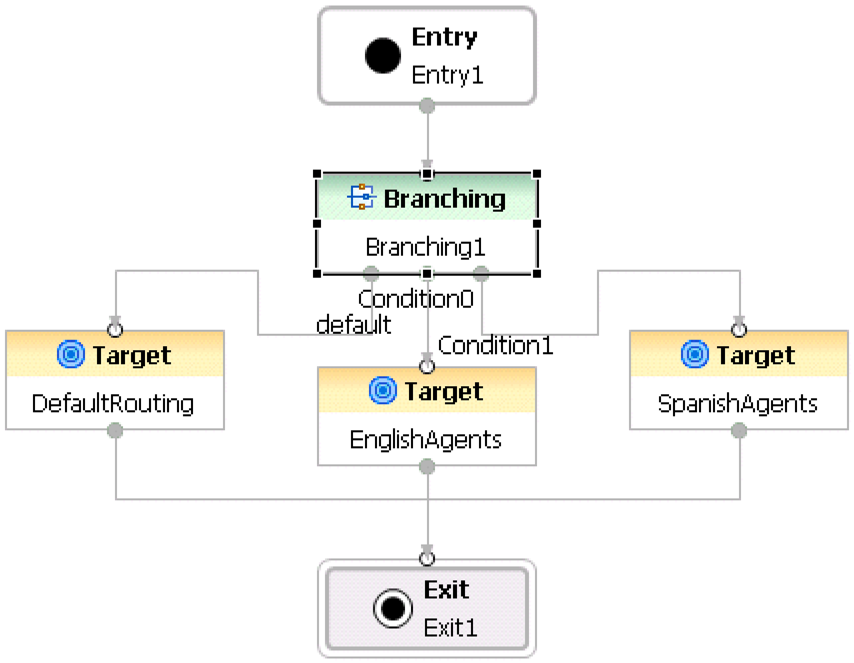
CallsWaiting	RStatLoadBalance	StatTimeInReadyState
InVQWaitTime	StatAgentsTotal	StatAgentsAvailable
PositionInQueue	StatCallsAnswered	StatAgentsBusy
RStatLBEWTLAA	StatCallsCompleted	StatAgentsInQueueLogin
RStatCallsInQueue	StatCallsInQueue	StatAgentsInQueueReady
RStatCallsInTransition	StatEstimatedWaitingTime	StatAgentLoading
RStatCost	StatExpectedWaitingTime	StatAgentLoadingMedia
RStatExpectedLBEWTLAA	StatLoadBalance	StatAgentOccupancy
RStatExpectedLoadBalance	StatServiceFactor	

For a definition of each statistic, refer to the chapter on routing statistics in the *Universal Routing 8.1 Reference Manual*.

4. For the sample strategy, select `StatTimeInReadyState`.
5. In the Properties tab, opposite the Statistics Order property, click under the **Value** column to display a drop-down menu.
6. Select **Max** from the drop-down menu since we want the agent who has spent the maximum amount of time in a Ready state. You have no finished configuring the first Target block.

Adding Additional Blocks and Connecting

1. Repeat the steps in the Target Selection section to define properties for the second Target block for Spanish-speaking Agents.
2. Repeat the steps in the Target Selection section to add a third Target block named DefaultRouting for default routing where you route to an Agent target type.
3. Connect the three Target blocks to the Exit object. The routing strategy diagram now appears as shown below.



YourFirstApp8.gif

Saving

1. Save the diagram as it exists so far by selecting **File** > **Save**. You will not be able to generate code if you do not save the file.
2. If you have set your Configuration Server preferences to do so, validate the code by selecting **Diagram** > **Validate**.

You can also click the Validate icon ✓ on the upper-right of the Composer main window when the workflow canvas is selected. The Problems tab shows the results of validation for this particular Resource. Fix any problems before continuing.

Generating Code

1. Generate the code by selecting **Diagram** > **Generate Code**, or by clicking the Generate Code icon on the upper-right of the Composer main window when the canvas is selected. Check the Problems tab for errors and fix any problems. If code generation succeeds, click **OK** at the confirmation dialog box. The SCXML code is generated in the `src-gen` folder.
2. Test the workflow.
3. Deploy the workflow.

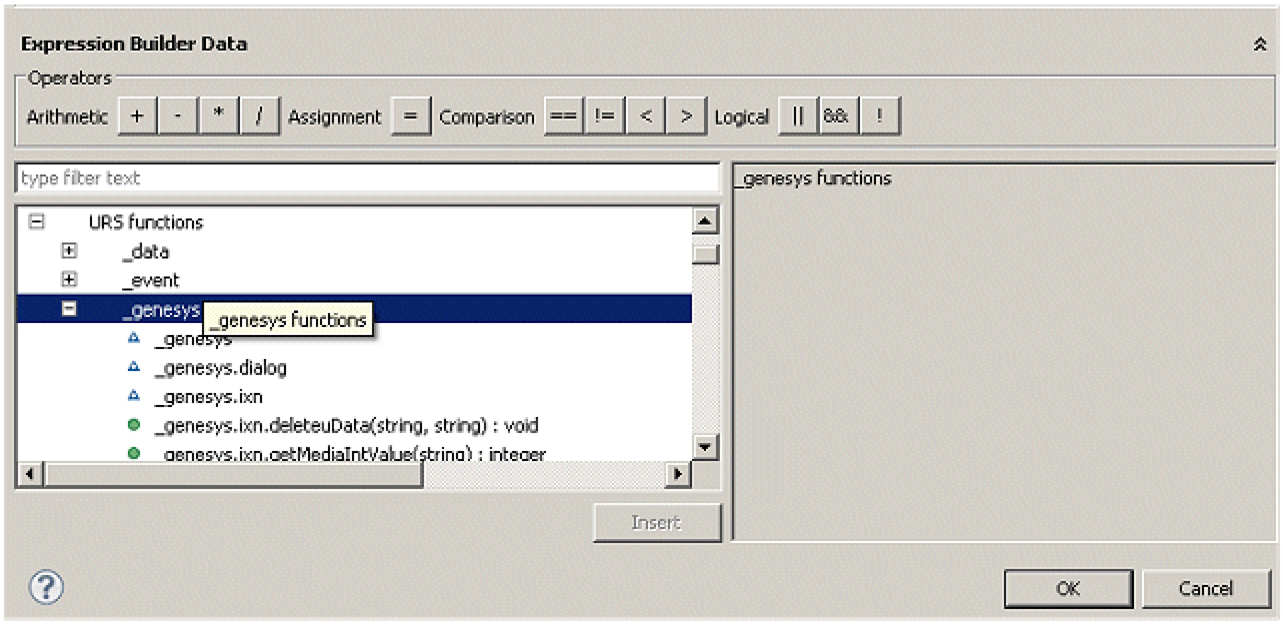
Using URS and ORS Functions

Composer's Expression Builder allows you to use SCXML elements and extensions as described the Orchestration Server Developer's Guide as well as Universal Routing Server (Workflow) functions.

Expression Builder opens from many blocks; to name a few:

- Assign--Assign Data Property
- Branching--Conditions Property
- ECMAScript (for workflows)--Script Property
- Entry--Variables Property
- Log--Logging Details Property
- Looping--Exit Expression Property

Using one or more of the above blocks in a workflow gives access to functions for SCXML-based strategies. The _genesys data subcategory is where you will find modules that access Genesys-platform related objects, properties, and functions.



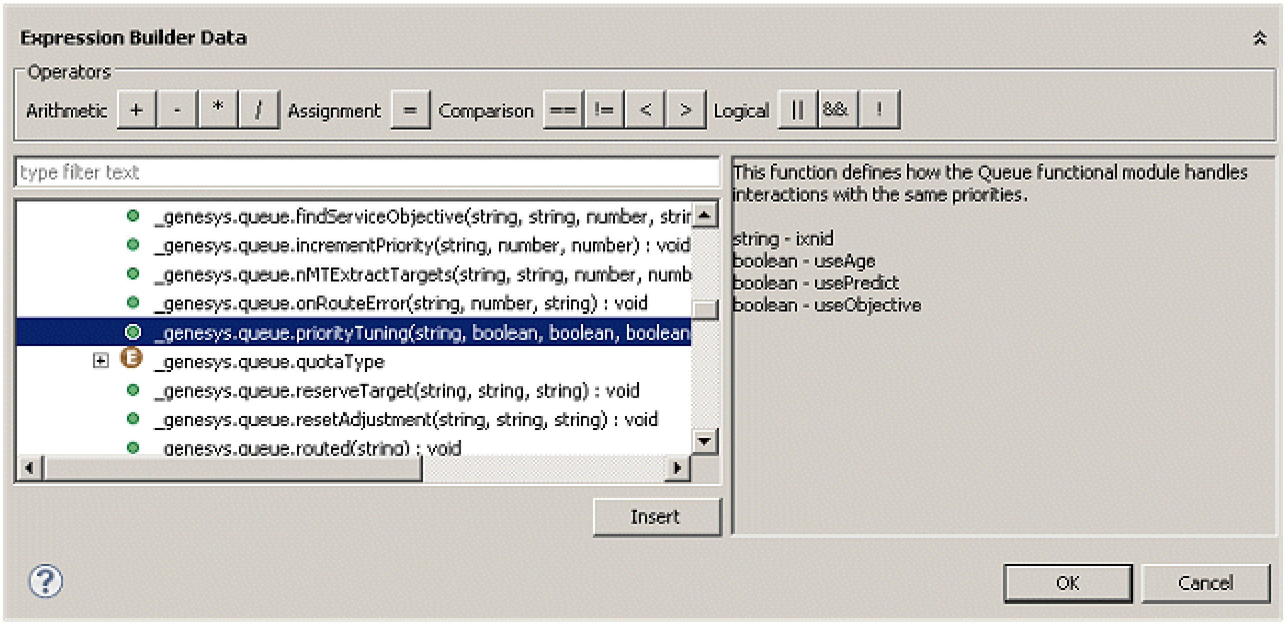
Below is summary information on certain functions accessible via Expression Builder.

Interaction Priority

URS always puts interactions into waiting queues according to their priorities. The `priorityTuning` function defines how URS handles interactions with the same priorities. By default, interactions with the same priority are ordered according to the time the interaction began to wait for some target.

- `useAge` parameter(true or false). If true, URS uses the time the interaction was created instead of the time interaction is placed into the waiting queue. Age of interaction is usually the time that URS starts the strategy for the interaction.
- `usePredict` parameter (true or false). If true, then URS calculates the estimated time for the interaction to be answered and will use this time instead of the time that the interaction has already waited.
- `useObjective` parameter (true or false). This parameter works with the URS use_service_objective option (*Universal Routing 8.1 Reference Manual*), which allows you to scale the time interval for processing different interactions. If this option set to true, URS scales the time interval that the interaction waits (or is going to wait) according to the Service Objective of the interaction

If two interactions wait the same time, such as 60 seconds, but interaction #1 has a Service Objective of 30 seconds and interaction #2 has a Service Objective of 120 seconds, then URS puts interaction #1 ahead of interaction #2 in the queue (60/30 > 60/120). The figure below shows how to access the `priorityTuning` function in Expression Builder.



_genesys.queue.findServiceObjective(string, string, number, strir
_genesys.queue.incrementPriority(string, number, number) : void
_genesys.queue.nMTExtractTargets(string, string, number, numb
_genesys.queue.onRouteError(string, number, string) : void
_genesys.queue.priorityTuning(string, boolean, boolean, boolean
_genesys.queue.quotaType
_genesys.queue.reserveTarget(string, string, string) : void
_genesys.queue.resetAdjustment(string, string, string) : void
_genesys.queue.routed(string) : void

This function defines how the Queue functional module handles interactions with the same priorities.

string - ixnid
boolean - useAge
boolean - usePredict
boolean - useObjective

- For more detailed information on using this function, see the *Universal Routing 8.1 Reference Manual* and the *Universal Routing 8.1.Routing Application Configuration Guide* located on the Universal Routing Wiki.
- For information on Orchestration Server functions, see Orchestration Extensions and Core Extensions topics in the *Orchestration Developers Guide*.

Routing Block Palette Reference

When you create an application, Composer's palette contains the diagram building blocks. Use the workflow diagram blocks, located on the Element Descriptions *palette*, to develop routing applications. The palette contains the link tools, and various categories of blocks used to build routing workflow diagrams:

- **Interaction Process Diagram Blocks**. Required for routing applications. Use to provide a high level view of how multimedia interactions flow through various components like media servers, interaction queues, workbins, and workflows. In addition, an IPD functions as the starting SCXML page for both voice and multimedia interactions.
- **Context Services Blocks**. Context Services refers to an optional capability of Universal Contact Server and its Universal Contact Server (UCS) Database, a repository of customer-related service, and interaction-centric data (current and historical) from Genesys and third party sources.
- **eServices Blocks**. Use to create a routing workflow that performs specialized processing of multimedia (non-voice) interactions.
- **Flow Control Blocks**.
- **Link Tools**. Use to connect blocks in the order that the application should follow.
- **Routing Blocks**. Routing blocks specify a routing action to be performed with the current interaction, such as sending an interaction to a specific agent group.
- **Server Side Blocks**. 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.
- **Voice Treatment Blocks**. Voice Treatment blocks specify an action to be performed with the current interaction, such as playing music for the caller.

The **Common Blocks** section describes blocks that can be used by both routing and voice applications.

Interaction Process Diagram Blocks

The following blocks can comprise an interaction process diagram (IPD):

- **Interaction Queue Block** to define or select a multimedia (non-voice) interaction queue in an interaction process diagram and to create Views, which defines conditions for pulling interactions out of the queue for submittal to workflows.
- **Media Server Block** to get interactions of a particular media type (other than voice) into an interaction process diagram.
- **Workflow Block** to point to a workflow resource (workflow diagram or SCXML file) to which interactions should be sent for processing.
- **Workbin Block** to define a temporary storage area called a workbin, which is accessible from the agent desktop.
- **Flow Control blocks** to support multiple views per interaction queue, the following Flow Control blocks are available when creating an IPD: Branching, ECMAScript, Log.

IPD Overview

There is a difference between interaction process diagrams (IPDs) used for voice-only interactions and IPDs used for multimedia interactions.

- An IPD for multimedia interactions will contain Interaction Queue, Media Server, and Workflow blocks and possibly multiple instances of each block depending on the design.
- An IPD for voice interactions will only contain a Workflow block that points to a single workflow resource.
- An IPD for multimedia interactions may contain workflow-generated blocks whereas an IPD for voice interactions will not show these nodes.

Starting a New IPD

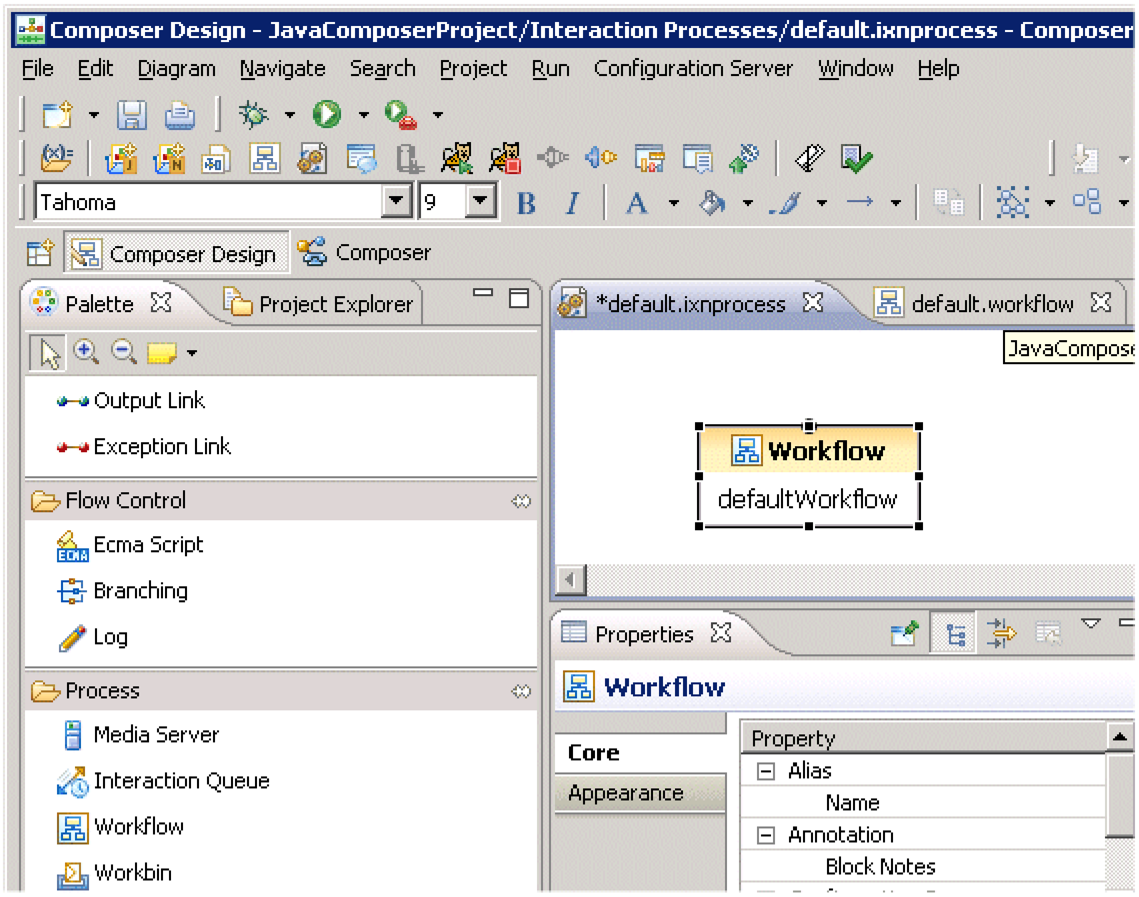
Use these instructions after reviewing the IPD Planning and Preparation topic.   Before starting a new IPD, you may wish to Connect to Configuration Server (optional). You can do this now or during the validation phase. When not connected, Configuration Database objects do not appear for selection in Composer dialog boxes; you must know the names in advance.

Method #1: Create a New Java Composer Project

1. Create a new Java Composer Project. In Composer Design perspective, this automatically creates an Interaction Processes folder and default.ixprocess file in the Project Explorer. It also automatically creates the following tabs on the canvas:

   - `default.ixprocess`
   - `default.workflow`

The `default.ixprocess` tab contains a starting Workflow block. The Palette tab shows the Media Server, Interaction Queue, and Workflow blocks.



You can rename the IPD at any time by right-clicking the `default.ixnprocess` file in the `Interaction Processes` folder in the Project Explorer and selecting **Rename**. The renaming operation does not result in any changes being written to Configuration Server.

Method #2: New IPD Diagram Wizard

1. Bring up the wizard. There are various ways:
   - Click the Create New Interaction Process button on the toolbar.
   - Or select **File** > **New** > **Other** > **Composer** > **Diagrams** > **Interaction Process Diagram**.
   - Or in the Project Explorer > Right-click > **New** > **Other** > **Composer** > **Diagrams** > *Interaction Process Diagram*.
2. Name the IPD
3. Select or create the associated Project.
4. Click **Finish**.

Composer creates an Interaction Processes folder and default.ixprocess file in the Project Explorer. It also automatically creates default.ixprocess and default.workflow tabs on the canvas. The `default.ixprocess` tab contains a starting Workflow block. The palette shows the Media Server, Interaction Queue, and Workflow blocks. Note: An IPD does not use Entry or Exit blocks.

IPD Properties

Double-clicking an IPD in the Workflows folder shows the following properties:

Name Property

This property shows the name of the diagram. An IPD diagram can be renamed at any time. The renaming operation will not result in any changes being written to Configuration Server.

Events Property

With the `.ixnprocess` tab selected, click the empty space in the IPD to see Events in the Properties view.

The Events property (which replaces the 8.1.2 Wait for Event property) works with the Interaction ID property in Routing and certain eServices blocks. You select/enter the event(s) that the generated code will wait for before the workflow code is invoked. If unset, the IPD code will not wait for any event before invoking the workflow code. ORS will transition on the first event received. The Application does not need to receive all declared events to transition to the next block.

Composer supplies default handlers for most the common events (interaction.added, interaction.deleted, and so on). You can enable/disable handlers, add/remove handlers, or customize them. The `catch all errors` handler, which was previously hard-coded in the IPD SCXML, is now exposed as an event handler and can be enabled/disabled/customized.

All system handlers run into the system thread of the application while the workflow generated code runs into the user thread of the application.

Event handlers can control the lifecycle of the user thread by raising the `application.start` event (to start the application, see the interaction added event handler) or the `application.exit` event (to terminate the application, see the `interaction deleted` event handler).

To define this property in case of "interaction-less" processing (defined below), you may:

- **Wait for a user-defined event**:
  - Add a new Event item in the Events property list.
  - Specify the appropriate Event name.
  - In the body of that event, add the code: `<raise event="application.start">` to start the user thread (to run the workflow SCXML).

- **Start the application without waiting for any event**:
  - Remove all Events items
  - Or disable all Events items
  - Or in the Events dialog box, add the predefined "interaction-less processing" event (or any similar event having no Event AND no Condition defined).

In addition to catch all errors, Composer defines the following events:

- `interaction.present` or `interaction.added`. This property will be used by default as a value of the IPD/Events property.
- `interaction.onmerge`. This event will be caught by the generated IPD SCXML. The value of the variable system.InteractionID will be updated when the transfer is completed. The parent or primary interaction ID will be referred to instead of the consult interaction ID.
- `interaction.deleted` (active interaction id). This event will be caught by the generated IPD SCXML. Default behavior will be to exit the session. Default behavior can be overrriden if interaction.deleted is handled in the workflow diagram.
- `interaction.deleted` (consult interaction id). This event will be caught by the generated IPD SCXML. Default behavior will be to exit the session if the parent interaction is gone or to do nothing otherwise. Default behavior can be overridden if interaction.deleted is handled in the workflow diagram.

Note: The condition expression for event-related properties in interaction process (IPD) and workflow diagrams are not XML-escaped when generating the SCXML code. For more information, see Troubleshooting ORS Compile Errors and Non Escaped Special Charcters.

Interaction-less Processing

In the case of "interaction-less" processing (for example, see the Force Route Block Interaction ID Property), you can start an ORS session using the ORS REST API and then decide either not to wait (no Wait For Event defined) or to wait for a user-defined event. The ORS REST API allows you to send an event to a particular ORS session. To define this property:

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Wait For Event dialog box.
3. Do one of the following:
   - Leave interaction.present to keep the default value, the system variable InteractionId, which will be initialized automatically in this case.
   - Click Add and select from the list of SCXML events or enter the event name. After selecting an event, the dialog box displays a description of the event.

A system variable, AppStartEvent, will be generated in the IPD <datamodel>, which will be initialized to the contents of the specified start event. If unset, the variable will be set to undefined (not the string undefined).

   - If necessary, click Remove to clear a selected event.

4. Click OK.

Created By Property

To be filled in by the user/author of the document.

Created On Property

Auto-populated by Composer to indicate the timestamp when the diagram was created.

Designed Using Property

Auto-populated by Composer to indicate version of Composer used to create this diagram.

Last Modified By Property

Provided by the user to indicate who updated the diagram last.

Last Modified On Property

Filled in by Composer when the diagram is modified.

Version Property

Provided by the user for versioning purposes during development.

Namespaces Property

Use to refer to custom namespaces in the generated code. To define this property:

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Namespaces dialog box.
3. Click Add.
4. Enter the namespace prefix (see example below)
5. Enter the namespace URL (see example below)

6. Click OK.

When an event is sent to an ORS session via http, a response can be sent back from the session via http by using the `ws:response` tag as shown below. `<ws:response requestid="_data.reserveSendId" resultcode="JSON.stringify( resultReserve )" />` Namespace: xmlns:ws=http://www.genesyslab.com/modules/ws

Extensions Property

This attribute allows arbitrary attributes to be added to the root <scxml> element and is used by Orchestration Server to control persistence, session recovery, and other functionality.

Note: Composer-generated SCXML applications do not support the `w3c` value for the `_transitionStyle` extension attribute.

Refer to `<scxml>` element, `_transitionStyle` extension attribute in the SCXML Language Reference section of the *Orchestration Server Developer's Guide* for details.

Read Context Property

This is the counterpart of the Pass Context property in the Routing Blocks. The value of this property is true or false. When true, the application will try to read the URL of the originating session from the interaction's User Data. If that URL is defined, it will then attempt to fetch the context from the originating Orchestration Session.

Session Persistence Property

With the `.ixnprocess` tab selected, click the empty space in the IPD to see Session Persistence in the Properties view.

Select true or false. Use this property to set the _persist attribute of the <scxml> tag when generating the SCXML code. For more information, see the <scxml> element Attribute Details, SCXML Language Reference section, in the *Orchestration Server Developer's Guide*.

Deleting Blocks

If you need to delete all blocks in an IPD, then select the non-linked blocks and click **Edit** >> **Delete.** If any linked blocks are selected, the delete operation will not work.

Queue Interaction Block

Use this block to place a multimedia (non-voice) interaction into an existing queue created with the Interaction Queue block. The generated SCXML code instructs Universal Routing Server to request Interaction Server to move the current interaction into the specified queue. You can select an existing interaction queue from within this block or specify a variable that contains the name of an interaction queue. You can also send an interaction to a queue in another IPD within the same Project. The selected interaction queue appears as a Queue Reference block in the interaction process diagram. Important Note! Each interaction path in a workflow for multimedia interactions should end with one of these blocks: Stop Interaction, Queue Interaction, or Route Interaction. Also see information on the App Terminate Ixn On Exit variable.

Use Case

The logic of a routing workflow may determine some attributes of an interaction, such as by looking at the Subject line of an inbound e-mail, and then use different interaction queues as a method of segmenting these different types of interactions. You could use the Branching block for this purpose. The Queue Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Interaction Queue Name Property

Use this property to specify the queue where the interaction is to be placed.

1. Click under Value to display the ⋯ button.
2. Click the button to open the Interaction Queue dialog box. Your dialog box may include existing IRD business processes with associated interaction queues underneath.
3. Select an interaction queue to which the incoming interaction has to be sent.
4. Click OK.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction. Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Diagram Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID

2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before queueing it. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Pass Context Property

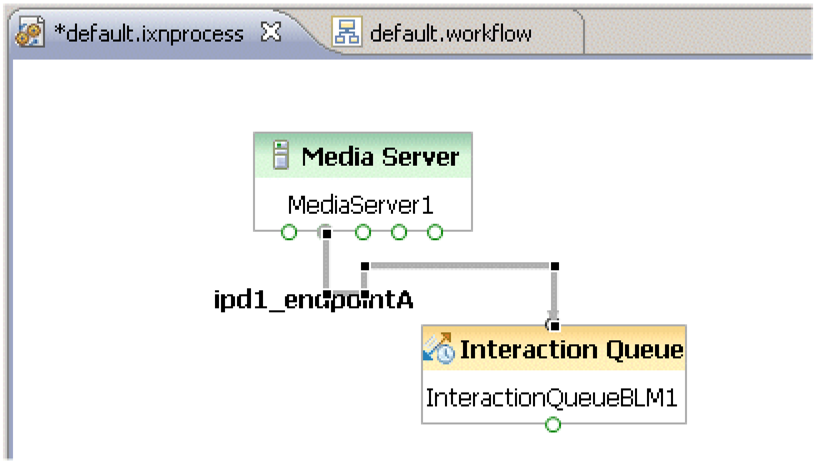
This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Interaction Queue Block

This block is used only for multimedia workflows. Use this block to define or select a multimedia (non-voice) interaction queue in an interaction process diagram and to create Views, which defines conditions for pulling interactions out of the queue for submittal to workflows. You can create multiple views per queue, with each view managing submission of an interaction to a separate routing workflow. The Publishing operation pushes the interaction queue into the Configuration Database. Its definition is stored as a `CfgScript` object of type Interaction Queue. After defining the queue, you can then place interactions it as follows:

- The interactions can be sent to the queue from a Media Server block.



- Interactions can also be sent to the queue by using the Queue Interaction block in a Workflow. Queues that you defined with the Interaction Queue block appear for selection in the Queue Interaction block.

Notes

- Composer shows one output port per defined view. This allows the user to route interactions coming through this view to a specific workflow.
- For all properties below, no updates to Configuration Server are created until you invoke the Publish operation.
- You cannot reuse an existing interaction queue in the same IPD, but you can use the same interaction queue in different IPDs. For more information, see Linking IPDs with Workflows.
- Composer points Interaction Queue objects directly to `EnhancedRoutingScript` (ERS) objects. It does not point Interaction QueueView objects to `EnhancedRoutingScript` objects.

The Interaction Queue block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Click under Value to display the button. Enter text to describe the block.

Object Name

A unique Configuration Server Object Name will be created once a Publish operation is executed. No updates to the database are created until you invoke the Publish operation. If no name is specified for the interaction queue, it will default to the currently implemented naming convention of <project name>.<diagram name>.<block name>. Note: If you rename the block after its corresponding `CfgScript` object is created in Configuration Server, the original published object name in Configuration Server remains unchanged. For more information, see Publishing Updates.

Enabled Property

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later.

Existing Queue Property

Use this property to select an existing interaction queue. Any changes made to the queue in the IPD are updated in Configuration Server during the Publish operation.

Queue Description Property

Enter a description of the interaction queue. This property will map to the Description key in the annex section Queue of the `CfgScript` object for the interaction queue.

Views Property

Use the Views property to define one or more Views for an interaction queue. Composer creates one outport per View. It is mandatory to define at least one view for an interaction queue in an IPD. Background: An interaction queue can have one or more associated Views. Each view represents an exit channel from the interaction queue into a workflow. In Configuration Server, each view will be created as a separate `CfgScript` object of type `InteractionQueueView`.

Defining a View

To define a view:

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the View Properties dialog box.
3. Click **Add** to display Main, Parameterized Conditions, and Segmentation tabs. For information on these tabs, see the Interaction Queue Views topic.
4. After you complete the applicable fields in these tabs, click **OK** to close the View Properties dialog box.

Note: Multiple views are supported on a single queue. This enables certain interactions to be processed earlier than others.

Interaction Queue Definition

Genesys software maintains both **interaction** and **virtual queues**.

- A multimedia queue in an interaction process diagram is called an *interaction queue*. You use Composer's Interaction Queue block to create/configure an interaction queue, which can also be thought of as a *persistent* queue. Once an interaction queue is configured and activated, multimedia interactions arriving from media servers (Media Server block) can be selected using the Views block property.
- In contrast, all voice interactions arriving from T-Servers are submitted to Universal Routing Server (URS), which places them in virtual queues. A virtual queue is not a physical queue but rather a logical queue to which all voice interactions are queued if the specified targets within a routing workflow are not available.

Deleting Interaction Queues

If you delete an Interaction Queue block from an IPD, Composer does not allow deletion of the Interaction Queue block itself. In this case, you must manually clean up the objects using Genesys Administrator.

Interaction Queue Views

This topic applies to the Views property for the Interaction Queue Block and Workbin Block topics. Views define parameters for pulling interactions from queues and workbins for submittal to workflows (routing strategies) for further processing.

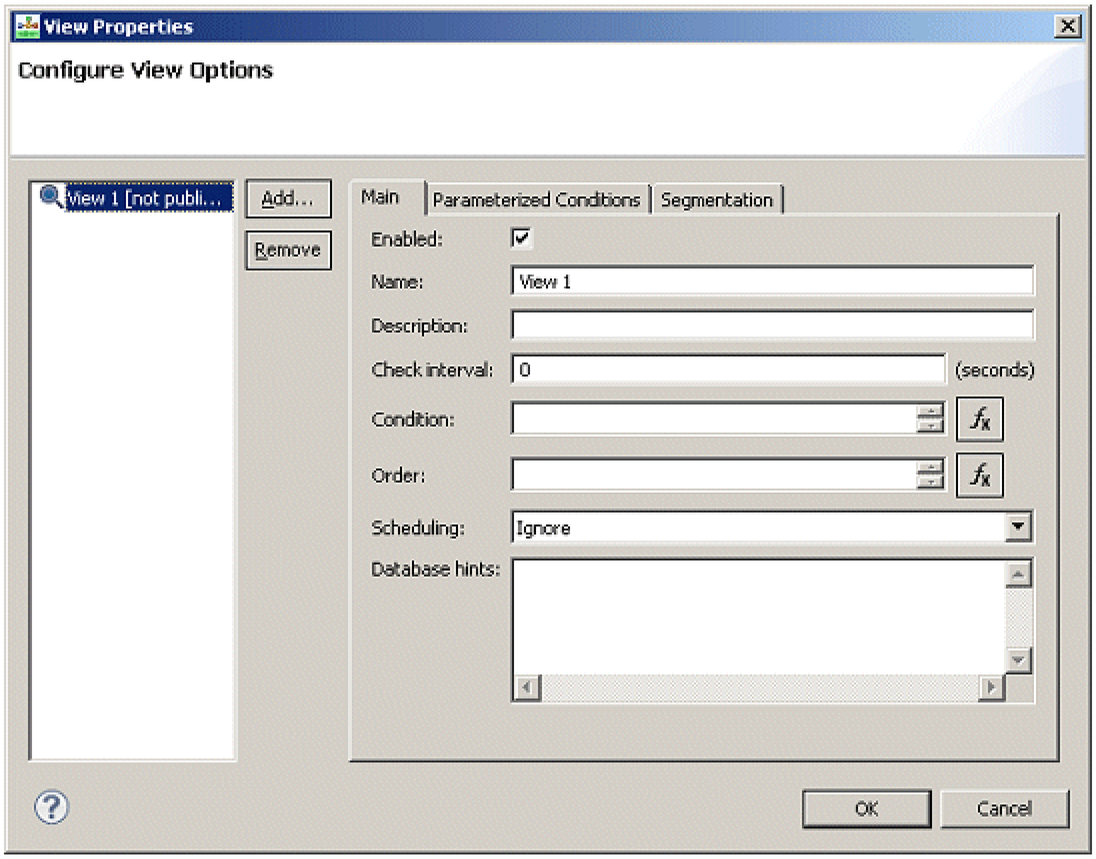
Multiple Views Per Queue

You can create multiple Views per queue. Composer creates a dedicated output port on the Interaction Queue block for each View defined in the block.

Defining Views

You are required to create at least one View. To define a View: From the Views property in the Interaction Queue or Workbin block:

1. Click under Value to display the ... button.
2. Click the ... button to open the View Properties dialog box.



Enabled: ✓
Name: View 1
Description:
Check interval: 0 (seconds)
Condition:
Order:
Scheduling: Ignore
Database hints:

1. Click Add to display fields in the Main tab.
2. Complete the fields as described below.

Main Tab

Fields in the Main tab are described table below.

Field	Description
Enabled	Check the box to make the view ready to extract interactions.
Name	Enter a name for the view to be used when saving as Configuration Database `Script` object.
Description	Enter text describing the view.
Check Interval	Enter the number of seconds to specify the frequency (time interval) that Interaction Server will use to check the queue and, if necessary, adjust the number of interactions that can be submitted to the workflow based on the Scheduling field.

Condition	You have the option of creating an expression to be used as the basis for extract interactions from the queue. Examples: `customer_segment='gold' AND service_type='sales'` `_time_in_queue[] > 4320` You can specify one or more expressions, which can be comprised of: • An interaction attribute name from the interactions table. The *eServices/Multimedia User's Guide* lists and describes the interaction attributes that you can use when building an expression. • A relational operator, such as an equal sign or a greater than sign. • The attribute value in single quotes. • The expression is used for interaction selection as if you were constructing a SQL SELECT statement and specifying a WHERE clause.
Order	You have the option of defining the order for extracting interactions from the queue: `order:= [property_order[,order]]property_order:=` `property_name [asc|desc]` Example using an attribute found in the eService/Multimedia interactions table: `priority DESC`
Scheduling	Use to specify the scheduling condition that Interaction Server should use, based upon the scheduled time contained in interactions. The interaction scheduling functionality uses a database field called `scheduled_at`, which is mapped to an interaction property called `ScheduledAt`. For information on this field, see the chapter on interaction properties in the *eService/ Multimedia User's Guide*. Select one of the following: • Ignore Scheduling. Default. Select if there is no scheduled processing. Even if the value of `ScheduledAt` is set for some interactions, Interaction Server ignores it. • Scheduled Only. Select to process only interactions that are scheduled (`ScheduledAt` is set) as per the value of the scheduled time. If selected, Interaction Server uses the following condition: (`_current_time() >= scheduled_at`) and the following order: `scheduled_at, received_at, id`. This condition and the conditions below are stored in the Scripts folder of Configuration Manager, `InteractionQueueView` object, Annex tab. • Scheduled and Unscheduled. Select to process

	scheduled interactions at scheduled times (`ScheduledAt` is set) and after that, process unscheduled interactions. In this case, scheduled interactions are delayed until the scheduled time, and all others are processed immediately afterwards. If selected, Interaction Server uses the following condition: ((scheduled_at is NULL) OR (`_current_time() >= scheduled_at`)) and the following order: `scheduled_at, received_at, id`. • Unscheduled Only. Select to process only interactions that are unscheduled (`ScheduledAt` is not set). Interaction Server uses the following condition: (`scheduled_at` is NULL)
Database Hints	This field is only applicable to an Oracle database. Background: Oracle allows special tags in SQL queries that cause queries to execute in a way that optimizes performance. These tags are called Hints. For example, you may wish Oracle to use a certain index to reorder data during query execution. You can apply a Hint, which will cause Oracle to use a specific index. One Hint that Oracle provides is: `/*+ index (interactions interactions_default_view_idx) */`. You could enter this Hint in the Database Hints field.

Parameterized Conditions Tab

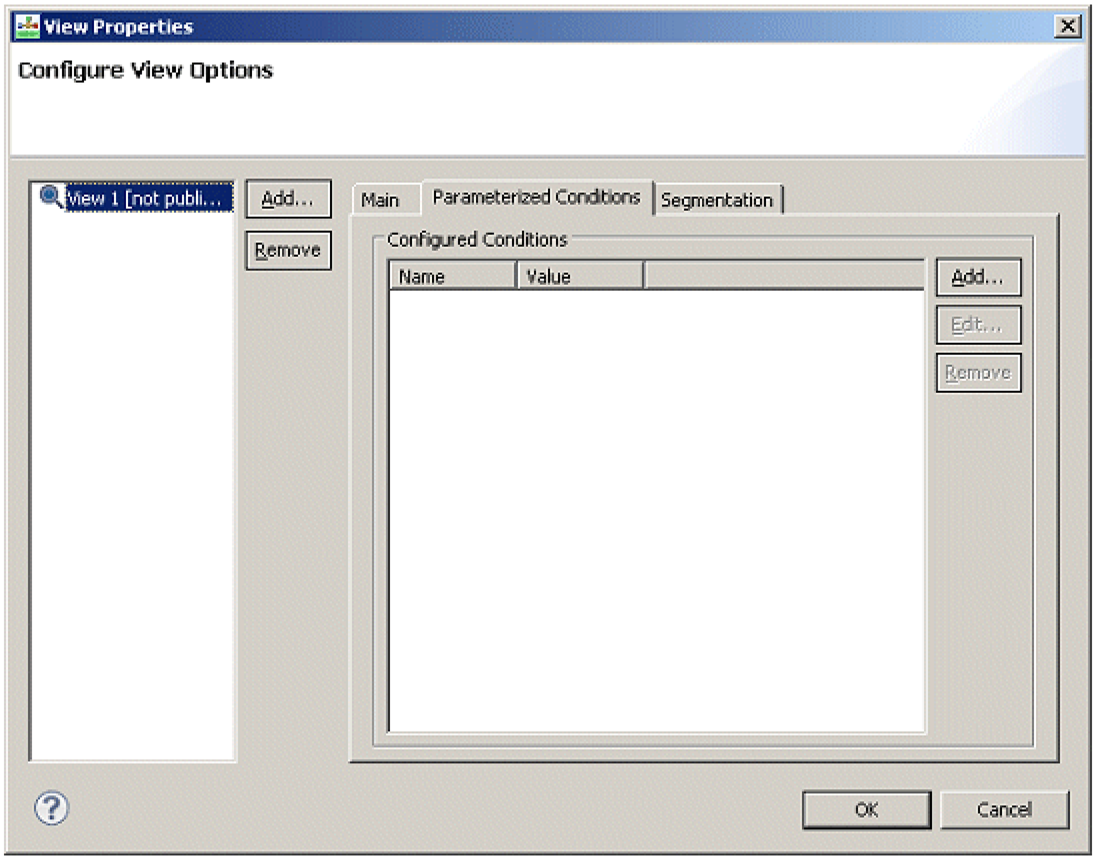
Use the Parameterized Conditions tab to specify interaction attributes that can be used in pull requests from clients of Interaction Server (for example, from an agent desktop). Each pull request can use any listed attribute, a combination of listed attributes, or none. If an attribute is not listed on this tab, then client applications cannot use it. For details on pull requests, see the RequestPull section in the chapter on Interaction Management Protocol in the *eService/Multimedia Open Media Interaction Models Reference Manual.* For example, if the Parameterized Conditions tab lists the from_address attribute, then a pull request from a client can include a condition such as `from_address=joe_customer@myisp.com`. This would retrieve all interactions from a particular contact. The Condition tab and the Parameterized Conditions tab both make use of interaction attributes (see the chapter on Interaction Properties in the *eService/Multimedia User's Guide*. The difference between them is:

- The Condition tab states a condition that applies to all pull requests.
- The Parameterized Conditions tab only lists attributes that can be used as parameters in a pull request, but it is up to the client whether or not to use these attributes.

You can: Select the attribute from a drop-down list of interaction attributes. This list includes most of the attributes in the `interactions` table. The exceptions are `abandoned_at, destinations, moved_to_queue_at, scheduled_at, server_id`, and `snapshot_place_id`. You can also enter the name of a custom property that you have created in Configuration Manager. Creating custom properties is described in the Interaction Properties section of the chapter on Interaction Properties in the *eService/Multimedia User's Guide*.

Using the Parameterized Conditions Tab

1. Click the Parameterized Conditions tab in the View Properties dialog box.
2. Click **Add**. The Property Configuration dialog box opens.



3. Under Name, enter an Interactions table attribute.
4. Under Value, enter a value.
5. Click **OK**. The View Properties dialog box shows your entry.

Segmentation Tab

Use the Segmentation tab on the View Properties dialog box to submit an equal number of interactions of different segments, to define a default limit for each segment pulled from a queue, and to limit the total number of interactions that can be submitted to a workflow.

Use Case

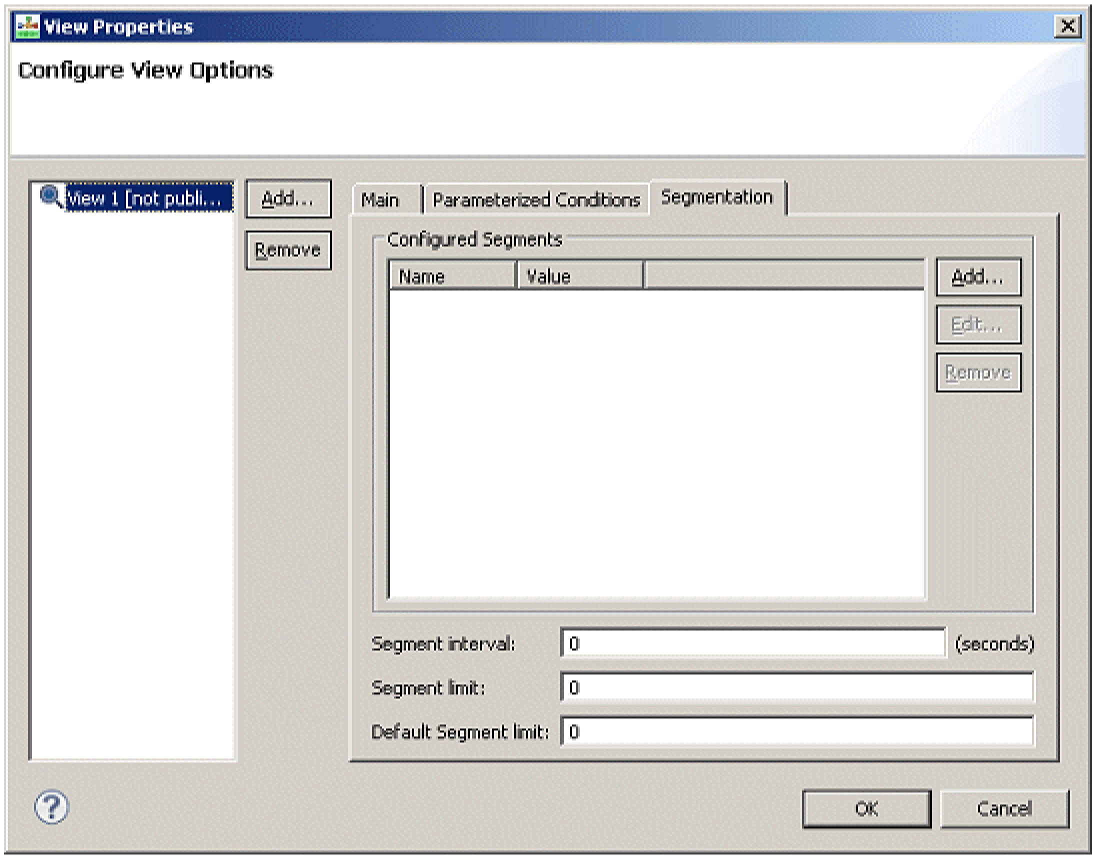
Assume the following:

- You have a simple business process: a queue, the queue's view, a strategy, and a submitter that submits interactions from the queue to the strategy through the view.
- There are two groups of agents equal in number. One group is trained to handle only customers of the gold Customer Segment and another group is trained to handle only customers of the bronze Customer Segment.
- The strategy directs interactions to the corresponding group of agents based on the value of the customer_segment property of an interaction (assume the value could be either gold or bronze).
- Next, start placing interactions into the queue, five interactions from bronze customers, then four interaction from gold customers, then again five interactions from bronze customers, three from gold, and so on.

If the strategy has a limit of five interactions that may be submitted into it, when the limit is reached, the strategy will be full of interactions from bronze customers, but will have no interactions from gold customers. As a result, interactions from gold customers will be waiting back in the queue and free agents, who are able to handle them, will also be waiting. Because the interactions are not yet in the strategy, the strategy is unable to route the interactions. To avoid such a scenario, you could add the customer_segment value to the Segmentation tab of the View Properties dialog box. After that, Interaction Server will fetch all interactions from the queue, grouping by the customer_segment property. It will find two distinct values of the property: gold and bronze. Interaction Server will then divide strategy limit by two (the number of distinct values) and limit the submission of each group of interactions to the strategy by the calculated value. As a result, Interaction Server submits an equal number of interactions from each group.

Using the Segmentation Tab

1. Click the Segmentation tab in the View Properties dialog box.
2. Click **Add**. The Property Configuration dialog box opens.
3. Opposite Name, enter an Interactions table attribute.
4. Opposite Value, enter a value.
5. Click **OK**. The View Properties dialog box shows your entry. An example is shown below.



Segment interval: 0 (seconds)

Segment limit: 0

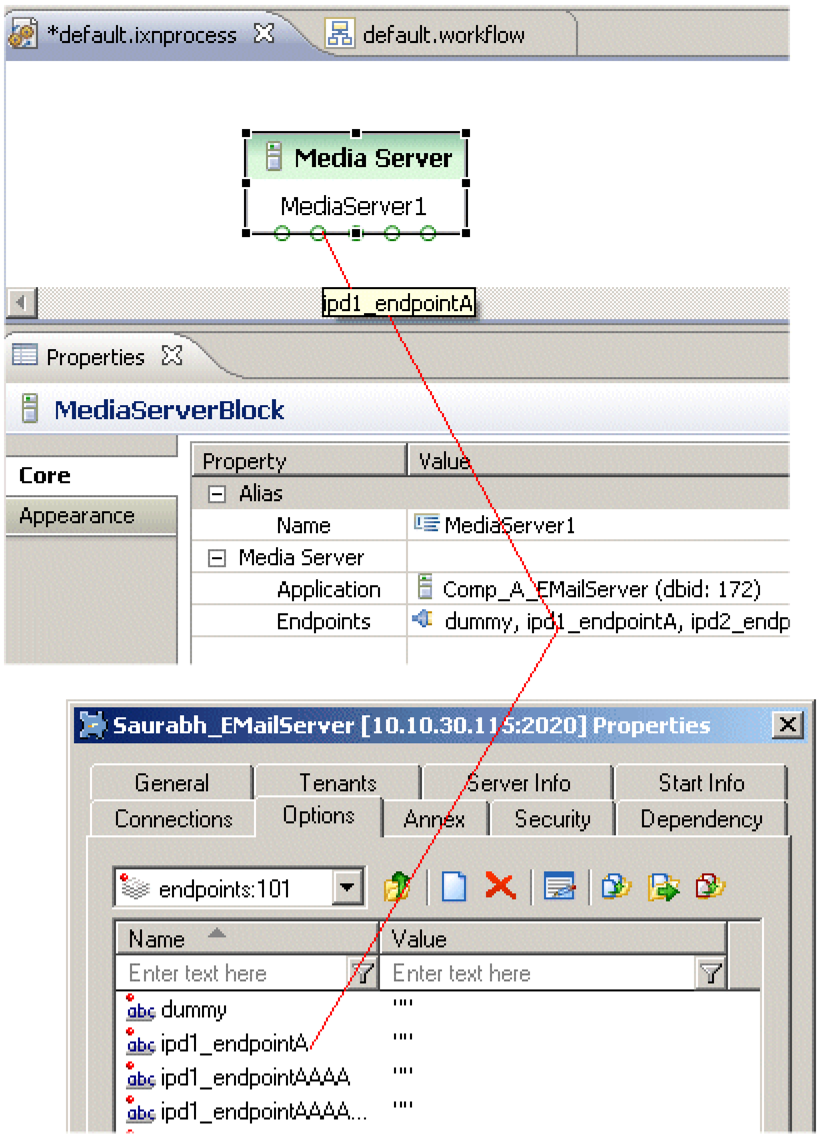
Default Segment limit: 0

Media Server Block

This block is used only for multimedia workflows. Use the Media Server block to get interactions of a particular media type (other than voice) into an interaction process diagram.

Endpoints

A Media Server is associated with one or more Endpoints, with each Endpoint connecting the Media Server to an interaction queue. For a Media Server block to show Endpoints, those Endpoints must first exist in the Configuration Database. Then, after you select the Media Server via the Composer Application Property, the Endpoint ports appear on the Media Server block as well as being listed opposite the Composer Endpoints property. The figure below illustrates this.



Property	Value
Alias	
Name	MediaServer1
Media Server	
Application	Comp_A_EMailServer (dbid: 172)
Endpoints	dummy, ipd1_endpointA, ipd2_endp

Name	Value
dummy	""
ipd1_endpointA	""
ipd1_endpointAAAA	""
ipd1_endpointAAAA...	""

Endpoints1.gif

The Publishing operation causes Composer to push the information into the Configuration Database. Its definition is stored as a CfgScript object. The Media Server block will usually be the first block in a multimedia IPD followed by an Interaction Queue block, and then a Workflow block. The process goes like this: You connect a Media Server Endpoint to an Interaction Queue block and the Interaction Queue block to a Workflow block. This arrangement directs multimedia interactions from the media server into a queue. A view defined for the queue then pulls interactions from the queue and sends them to a workflow for specialized processing. **Notes**

- A Media Server block has one or more outports; no input ports are available.
- Multiple Media Server blocks in an IPD cannot refer to the same media server application.

Media Servers Supported

Composer supports using the following Genesys eService/Multimedia media servers in IPDs:

- E-mail Server Java--interfaces with the enterprise mail server and the Genesys Web API Server, bringing in new e-mail interactions from customers and sending out replies or other outbound messages.
- SMS Server--used for the common text messaging service available on cellphones and other handheld devices.
- Chat Server--works with Web API Server to open, conduct, and close chat.
- Third Party Server, such as Capture Point applications. Capture Point applications are defined in two possible ways:

1. An application with type `CFGCapturePoint.`
2. An application with type `CFGThirdPartyServer`, containing a configuration option called capture-point-type in the settings section.

For more information on the above server types, see the *eServices 8.1 Deployment Guide.* The Media Server block has the following properties:

Name Property

Find this property's details under Common Properties.

Application Property

Select a media server to specify the `CfgApplication` object in Configuration Server that this block represents. Any media server already referenced by other Media Server blocks in the same IPD will not be listed. Once the media server Application object is selected, each Endpoint defined in its `CfgApplication` object will be shown as an outport on the block. **Notes:**

- Different Media Server blocks in the same IPD cannot refer to the same Media Server Application instance in Configuration Server.
- The same Media Server instance in Configuration Server may be referenced by multiple IPDs.
- One IPD can have multiple Media Server blocks.

Endpoints Property

Click under Value to display the Media Server Endpoints dialog box and select one or more Endpoints. An Endpoint connects a media server to a queue (Interaction Queue block) within an IPD. The dialog box lists all the Endpoints associated with the media server specified in the Media Server Application property. When you connect an Endpoint to an Interaction Queue block in the IPD diagram, this will cause interactions coming out of this Endpoint to go into the named interaction queue. **Notes:**

- For endpoints already existing on the selected Media Server, only those that were previously unused are displayed as outports on the block. All endpoints are displayed in the Endpoints dialog, with information about which diagram is using each endpoint.
- You may define Media Server Endpoints in offline mode. The changes will take effect immediately in the local IPD diagram. However, other IPD diagrams that use this Media Server will not see the changes as that will require that Configuration Server be updated with Endpoint details.
- In offline mode, you may input multiple Endpoints and have those definitions stored. Updates to Configuration Server occur when Composer is connected to Configuration Server and you invoke the Publish action.
- If an Endpoint of a Media Server is already assigned in Configuration Server, then the Endpoint for the Media Server block is not available for connection to an Interaction Queue block.
- If a Media Server block is deleted, its definition will be removed from the IPD. However, the Media Server object in Configuration Server will not be modified. Any Endpoints created in the current Media Server block will remain and may be deleted using Genesys Administrator. You can, however, delete the Endpoints first and then delete the block.

Workflow Block

This block can be used for voice workflows, multimedia workflows, or interaction-less processing when the block is not connected to an Interaction Queue block. Use this block in an interaction process diagram to point to a workflow resource (workflow diagram or SCXML file) to which interactions should be sent for processing. Outgoing connections automatically appearing from a Workflow block that represent objects specified inside the workflow are referred to in this help as "workflow-generated blocks." Important Note! When a workflow is part of an interaction process diagram used for multimedia interactions, always finish the workflow (and each workflow branch) with one of these blocks: Stop Interaction, Queue Interaction, or Route Interaction. The Workflow block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Click under Value to display the ... button. Enter text to describe the block.

Object Name Property

A unique Configuration Server Object Name will be created once a Publish operation is executed. No updates to the database are created until you invoke the Publish operation. Note: If you rename the block after its corresponding `CfgScript` object is created in Configuration Server, the original published object name in Configuration Server remains unchanged. For more information, see Publishing Updates.

Workflow Blocks and Publishing an IPD

When publishing an interaction process diagram (IPD) to Configuration Server, Workflow blocks are handled in two different ways:

Use Case #1: The Workflow block is dedicated to voice or interaction-less processing. In that case, you must use a stand-alone block (the block is not connected to any other block).

- When generating the code: Composer generates one SCXML file per such Workflow block (Name= IPD_<ipd file name>_<workflow block name>.scxml).
- When publishing: Composer creates one Enhanced Routing Script object (Name=<Project Name>.<IPD name>.<Workflow block name>) per such Workflow block in the IPD being published. The Application/

url property of the ERS refers to the SCXML url (if deployed). The Workflow block Object Name property (read only property) is updated to the name of the Enhanced Routing Script object.

Use Case #2: The Workflow block is dedicated to multimedia processing. In this case, the block is connected (directly or indirectly) after a Workbin block or an Interaction Queue block.

- When generating the code: Composer generates one SCXML file per Interaction Queue/Workbin block (Name=IPD_<ipd file name>_<interaction queue block name>.scxml). If an IPD has an Interaction Queue block connected to multiple Workflow blocks (multiple views are defined on the Interaction Queue block), only one SCXML file is generated when generating the code for that IPD. This unique IPD SCXML is used to initiate the execution for all Workflow blocks. At runtime, the Workflow SCXML to execute is selected depending on the view the interaction is pulled from.

- When publishing: Composer creates one Interaction Queue Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>) per Interaction Queue block. Composer creates one Interaction Queue View Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>.<View name>) per Interaction Queue block defined view.

Composer creates one Enhanced Routing Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>.Routing) per Interaction Queue block in the IPD being published. The Application/url property of this Enhanced Routing Script object refers to the Queue IPD SCXML url (if deployed). Composer does NOT create an Enhanced Routing Script object for the workflow blocks. The Workflow block Object Name property (read only property) is NOT updated.

Resource Property

To define a resource:

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Select Resource dialog box.
3. Select a workflow resource, which can be:
    - A workflow diagram that exists in any of the Projects in the Composer workspace.
    - An SCXML file created in Composer's SCXML Editor.
4. Select the workflow resource.
5. Click OK.

Shortcut Menu

Right-clicking a Workflow block opens a menu with the following options:

- **Add Note**
- **File**
- **Edit**

- **Delete From Model**
- **Format**
- **Open Workflow**
- **Show Properties View**

Selecting **Open Workflow** opens the workflow resource file in Composer.

- If the resource is a workflow, the diagram will be opened in the workflow diagram editor.
- If the resource is a SCXML file, it will be opened in the SCXML editor in Composer.

Workbin Block

This block is used only for multimedia workflows. Use this block in an interaction process diagram to define a temporary storage area called a workbin, which is accessible from the agent desktop. You can then use the Workbin property in the Route Interaction block when redirecting interactions for continued processing. Internally, a workbin has a queue and can support views. The workbin owner (agent, agent group, place, or place group) can view contents of the workbin and pull interactions out in any order. An agent associated with a workbin may get a notification when interactions are put into the workbin.

Input and Output Ports

A Workbin block:

- Does not have any input ports. You add Interactions to a workbin via the Route Interaction block in a workflow diagram.
- Has only one outport, which can be connected to a Workflow block in the IPD. Floating workbins are also allowed that are not required to be connected to Workflow blocks. This allows interactions routed to workbins to remain in workbins until they are pulled out by agents

The Interaction Workbin block has the following properties:

Name Property

Use this property to define the name of the workbin, which will appear on the block in the IPD (this is not the Configuration Database name). Find this property's details under Common Properties.

Block Notes Property

Click under Value to display the … button. Enter text to describe the block.

Object Name Property

A unique Configuration Server Object Name will be created once a Publish operation is executed. No updates to the database are created until you invoke the Publish operation. Note: If you rename the block after its corresponding CfgScript object is created in Configuration Server, the original published object name in Configuration Server remains unchanged. For more information, see Publishing Updates.

Interaction Queue Property

Specifies the queue to be used with this workbin. This may be an existing queue in the IPD or it can be a private queue exclusive to this workbin. Multiple workbins can use the same queue.

1. Click under Value to display the button.
2. Click the button to open the Workbin Interaction Queue dialog box.
3. Select one of the following:
    - Use **Private Queue** for a queue exclusive to this workbin.
    - Use **Queue** to select an interaction queue already defined in this IPD.
4. Click OK.

Views Property

A workbin can have one or more views defined in it. Each view represents an exit channel from the workbin (or workbin’s queue). The criteria defined in the view must be satisfied before an interaction can exit out from the view. You can also use a view to enable certain types of interactions to be processed earlier than others. Note: Irrespective of the number of views defined in the Workbin block, the block will have only one outport and will feed interactions to only one workflow. To define a view:

1. Click under Value to display the button.
2. Click the button to open the View Properties dialog box.
3. Click Add to display Main, Parameterized Conditions, and Segmentation tabs. For information on these tabs, see Interaction Queue Views.
4. After you complete the applicable fields in these tabs, click OK to close the View Properties dialog box. Each view will be created as a separate CfgScript object of type Interaction Queue View.

Description Property

Enter text that describes the workbin.

Display Name Property

Enter a display name for the workbin to appear in the interaction process diagram.

Enabled Property

Select **true** or **false** to enabled or disabled this queue as ready to accept interactions. A queue may be enabled during design time. Note: Setting this property is equivalent to enabling or disabling the Workbin object from Genesys Administrator or Configuration Manager.

Order Property

Use this property to specify the WHERE clause of a SQL statement, which will determine the order in which interactions will be sorted in the workbin.

Owner Property

A workbin may be owned by (associated with) one or more agents. Click the down arrow and select one of the following:

- **Agent**
- **Agent Group**(default)
- **Place**
- **Place Group**

Agents associated with a workbin may get a notification if an item is placed in the workbin.

Enabled

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later.

Deleting Workbins

If you delete a Workbin block from an IPD in online mode, Composer displays a prompt asking if you wish to delete its Configuration objects. These will include:

- Interaction Workbin (CfgScript:: `InteractionWorkBin`)
- Interaction Queues (CfgScript::`InteractionQueue`)
- Interaction Queue Views (CfgScript:: `InteractionQueueView`)

If you indicate Yes," then the Interaction Queue object and view objects will be deleted from Configuration Server. If you indicate No," Composer will not delete these objects from Configuration Server. You must manually clean up these objects outside of Composer. In offline mode, if you delete a Workbin block from an IPD, the Configuration Server connection dialog is shown. If a connection cannot be established, object deletion will fail and you must manually delete the objects outside of Composer.

Flow Control Blocks

In an IPD, when an interaction is submitted from an interaction queue to a routing workflow, you can create multiple views per queue, with each view having its own set of conditions and managing submission of an interaction to a separate routing workflow. To support multiple views per interaction queue, the following Flow Control blocks are available when creating an IPD:

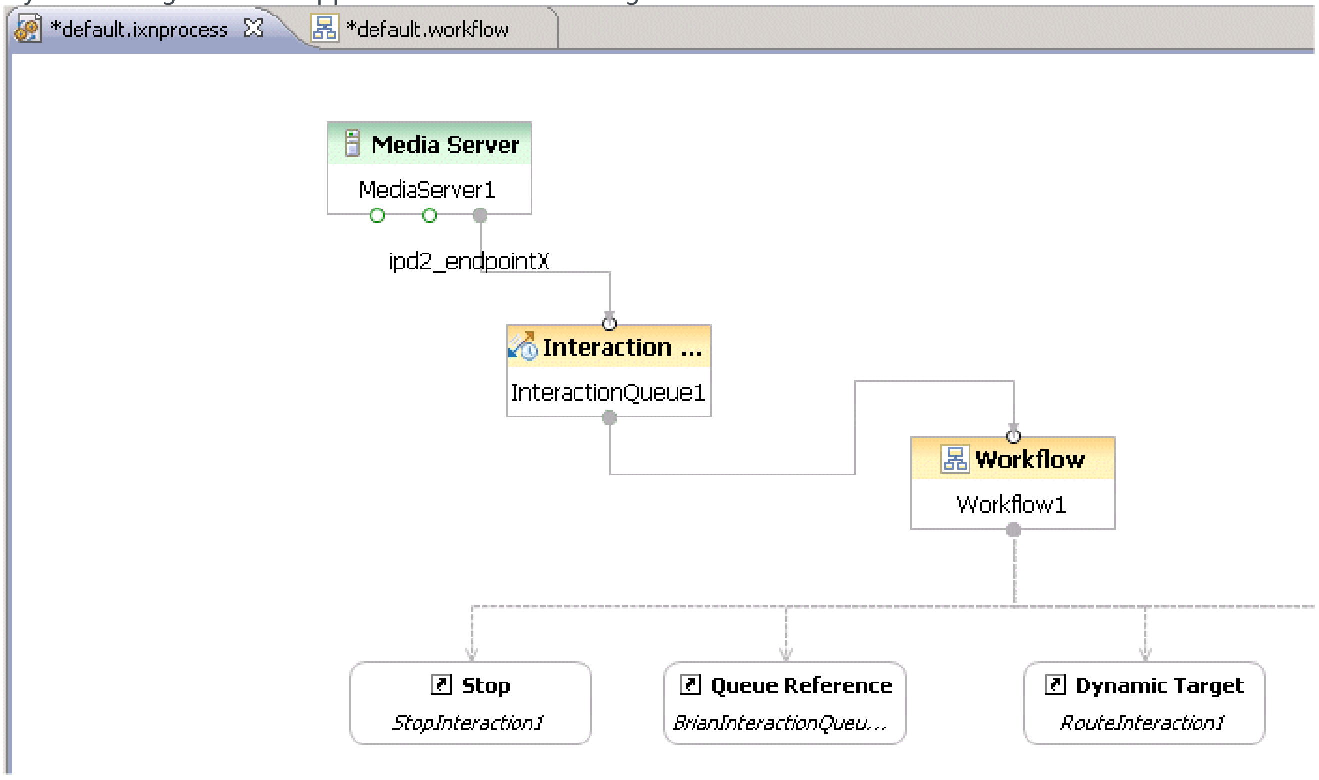
- **Branching**
- **ECMAScript**
- **Log**

Workflow Generated Blocks

Outgoing connections automatically appearing from a Workflow block that represent objects specified inside the workflow are called *workflow-generated blocks*. Composer generates the following types of workflow-generated blocks:

- Forward to External (Email Forward Block)
- Autoresponse
- Chat Transcript
- Classify Interaction
- Dynamic Target
- Queue Reference
- Stop
- Workbin Reference

For example, assume a Workflow block in an IPD references a workflow that contains a Queue Interaction block that references an interaction queue, a Route Interaction block that specifies a routing target, and a Stop Interaction block. The workflow-generated Stop, Queue Reference, and Dynamic Target blocks appear as shown in the figure below.



Queue Reference Block

If a workflow referenced in any Workflow blocks puts an interaction into a queue using the Chat Transcript, Email Response, E-mail Forward, or Queue Interaction block, the IPD shows this information. It display an additional block of type Queue Reference to represent the interaction queue and shows an interaction going from the Workflow block to this Queue Reference block. This block is automatically generated by Composer, therefore is not available on the IPD palette. Composer uses a different color, appearance, and structure for the Queue Reference block allowing you to easily differentiate it from other blocks selectable from the palette. If a workflow referenced in any Workflow block creates a new interaction, the IPD shows this information using the Queue Reference block. The connector for the new interaction from the Workflow block to the 'Queue Reference' block will be shown with a distinctive line pattern in a different color from normal connectors. UI team will be consulted. If the queue used in the Queue Interaction block was already defined in the IPD, the workflow will connect back to the Interaction Queue block already be present in the diagram. **Blocks Linking to Queue Reference Blocks**

- IPDs will show a linked Chat Transcript workflow-generated block if a referenced workflow uses a Chat Transcript block. The linked Chat Transcript block will link to a Queue reference block representing the queue used in the relevant block in the workflow.
- IPDs will show a linked E-mail Response workflow-generated block if a referenced workflow uses an E-mail Response block. The linked E-mail Response block will link to a Queue reference block representing the queue used in the relevant block in the workflow.
- IPDs will show a linked Forward workflow-generated block if a referenced workflow creates a reply e-mail using the E-mail Forward block with Forward to External as the Forward Type property. The linked Forward block will link to a Queue reference block representing the queue used in the relevant block in the workflow.
- IPDs will show a linked Classify Interaction workflow-generated block if a referenced workflow uses a Classify Interaction block to classify an interaction.

Dynamic Target Block

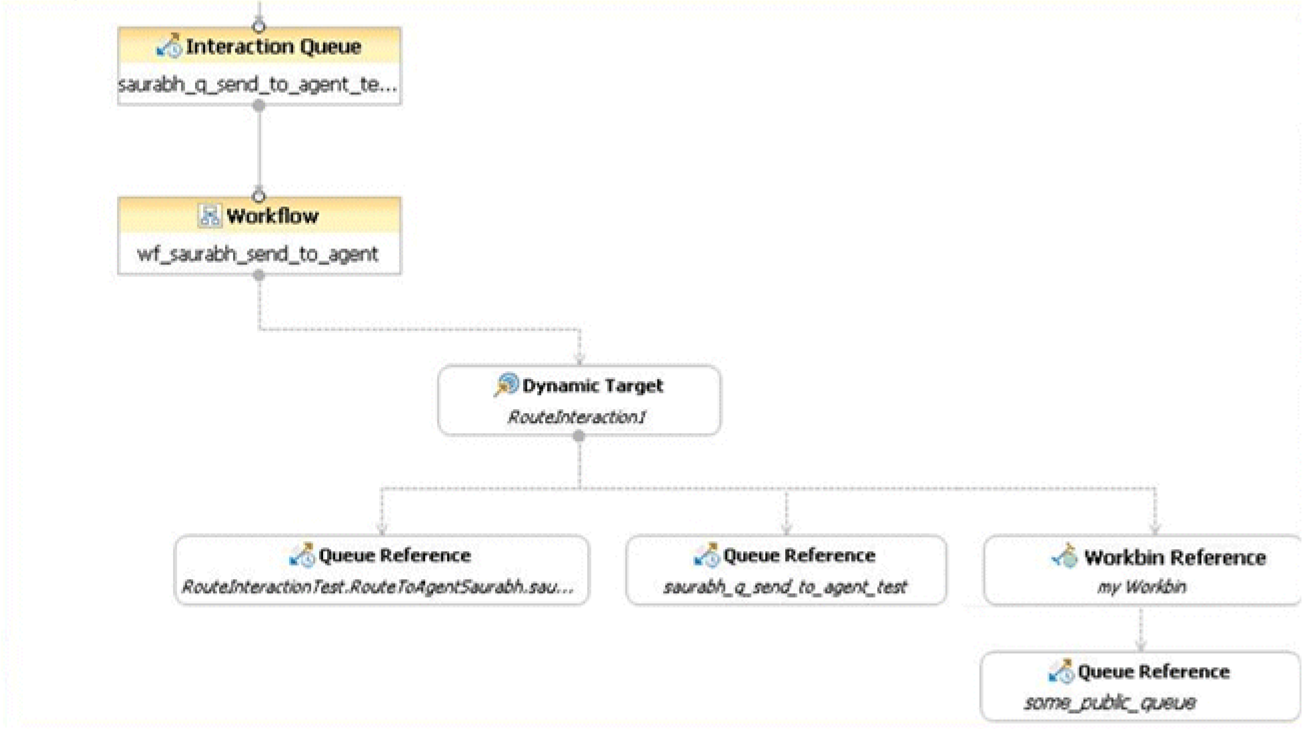
If a workflow referenced in any Workflow blocks routes the interaction to a dynamic target via the Route Interaction block, the IPD diagram will show this information using an additional block of type "Dynamic Target." This block is not available in the IPD palette since it is automatically generated by Composer. A Dynamic Target block gives a visual indication of the type of target the interaction is being routed to. A note attachment is added to the block giving details of the targets.

Stop Block

If a workflow referenced in any Workflow block stops an interaction using the Stop Interaction block, the IPD shows this information using an additional block of type "Stop." This block is not available in the IPD palette since it is automatically generated by Composer.

Workbin Reference

If a workflow referenced in any Workflow blocks uses the Route Interaction block to put an interaction into a workbin, the IPD diagram shows this information using a workflow-generated blocks of type "Workbin Reference." If a referenced Workbin block uses a non-private queue, a Queue Reference block is shown in the IPD originating from the Workbin Reference block. This is helpful if the Workbin block uses a queue that is defined in another IPD in the same Project. These blocks are not available in the IPD palette since they are automatically generated by Composer. The figure below shows example Queue Reference and Workbin Reference blocks.



Linking IPDs with Workflows

You can't explicitly link IPDs together as there is no higher-level diagram in Composer that shows IPDs as blocks and then allows you to interconnect them. To clarify:

- IPDs are linked via the workflows they are connected to. For example, assume IPD1 references Workflow1 and this workflow uses an Queue Interaction block. The block can be set to route the interaction to a Queue2 and Queue2 could exist in IPD2. Therefore IPD1 gets connected to IPD2.
- Interaction Queue blocks in IPDs don’t reference queues; they are queues.
- An interaction queue can be defined in only one IPD. Once defined and Published, a corresponding object is created in Configuration Server. This interaction queue cannot exist in any other IPD. It can, however, be referenced by blocks in any number of workflows in the same Project.

Note: IRD allows moving a queue from one business process to another. Once moved, the queue can be used in the new business process. To do the same in Composer, you can copy an Interaction Queue block from one IPD and paste into another. Then delete the block from the original IPD. The same Interaction Queue block cannot exist in two IPDs.

Publishing Updates

Publishing an interaction process diagram validates Project configuration information and pushes the information out to Configuration Server. When you configure an Interaction Queue, Workflow, or Workbin block, Composer does not send the information to Configuration Server until you invoke the Publish operation. This gives you complete control of the update process. When publishing an interaction process diagram, Composer also updates the Configuration Server/Object Name property of the IPD blocks for which a configuration object was created (applies to Workflow, Workbin, Interaction Queue blocks). You can set preferences to:

- Automatically publish upon saving.
- Display a prompt to save before publishing.
- Delete published objects when an interaction process diagram is deleted.
- Delete published objects when a project is closed or deleted.

When to Publish

- Any time properties for any of the blocks in the IPD are changed or blocks are added/removed.
- Any time a workflow diagram is renamed. In such cases, you must go back to the Workflow block in the IPD diagram and point the block to the renamed workflow.
- If you rename your workflow Project -- this will change the deployment URL for the Project. Publishing again will point the enhanced routing object to the new URL.
- If you delete Published objects in Configuration Server, you can re-publish the diagram to create new objects.

How to Publish

1. Before publishing, you must connect to Configuration Server.
2. Right-click an interaction process diagram in the Project Explorer.
3. Select Publish to Configuration Server. The following message appears: The selected IPD diagram's data was successfully published to Configuration Server.

Once these objects are created successfully in Configuration Server, some manual configuration is still required before interactions can work. For example, to redirect e-mails to an Endpoint, you must set endpoint key in the pop-clientX section of the e-mail server Application to the correct end-point. For more details, see Deploying a Routing Application.

Composer Marker Section

The Publish operation puts a marker section in each Configuration Database object it creates. The section name is __COMPOSER__. The section has the following keys:

Key Name	Value	Description
source	composer	Hard-coded value to indicate object was created by Composer
last_updated	<timestamp>	Indicates the timestamp when the publish operation was initiated that modified this object. Example: Thu Jan 07 11:26:36 PST 2010. I
owner_diagram	<string>	Name of the Composer diagram that published this object.
owner_project	<string>	Name of the Composer project that owns the IPD diagram in owner_diagram.
owner_uuid	<string>	Identifier of the Composer diagram that published this object.

IRD Objects Not Modified

The Publish operation does not modify queue and view objects that were not created by Composer. This avoids accidentally modifying objects previously created by Universal Routing's Interaction Routing Designer (IRD), which may cause issues in a legacy IRD/URS system. Using the marker section described above to indicate where the object was created, the following types of objects are not modified if they were not created by Composer:

- CfgScript of type Interaction Queue
- CfgScript of type Interaction Queue View
- CfgScript of type Workflow

Media Server Block Update Detail

The following updates are written to Configuration Server for all Media Server blocks in all IPD diagrams in the Project:

- Updates endpoints:<tenant DBID> section of specified CfgApplication.
- Each endpoint name attribute's value is set to the name of the interaction queue's CfgScript object that it is connected to. Only endpoints that were owned by this diagram, or were previously unconnected, are updated.

Interaction Queue Block Update Detail

The following updates are written to Configuration Server for all Interaction Queueblocks in all IPD diagrams in the Project:

- A CfgScript object (type = Interaction Queue) is created under the current tenant/Scripts folder. If the object already exists, its properties are updated.

Block Property	Configuration Server Object/Option
Description	CfgScript object/Annex Section Queue/Description property
Queue Enabled	State property of the CfgScript object. If TRUE, set to CFGEnabled. If FALSE, set to CFGDisabled.

- In the Annex of the CfgScript object, the property application is created in section Orchestration. Composer writes the value in this format: script:<name of Enhanced Routing CfgScript object where <name of EnhancedRouting CfgScript object> will be replaced with the workflow name.

This will essentially cause all views of the interaction queue to submit interactions to the SCXML application that the Enhanced Routing object points to. In the IPD, this will be a Workflow block pointing to an existing workflow diagram or an SCXML file. Note: If you rename an Interaction Queue block after its corresponding CfgScript object has been created, the object name in Configuration Server remains unchanged. Instead, the key Name in the Annex section Namespace and its value are set to the new name. Composer displays the changed name.

Interaction Queue View Property Update Detail

The following updates are written to Configuration Server for all Views defined for all Interaction Queue blocks and Workbin blocks in all IPD diagrams in the Project:

- A CfgScript object (type = Interaction Queue View) is created under the current tenant/Scripts folder. If the object already exists, its properties are updated. The name of the object follows this format: <container queue CfgScript object name>/<view name>

Block Property	Config. Server Object Type	Annex/Option Section	Key/Property
View Name	CfgScript	Namespace	Name
Description	CfgScript	View	Description
Enabled	CfgScript	-	State property
Check Interval	CfgScript	View	Freeze Interval
Condition	CfgScript	View	Condition
Order	CfgScript	View	Order

Scheduling	CfgScript	View	scheduling-mode
Parameterized Conditions (multi-valued)	CfgScript	View	Each value creates a key like "Condition.<value>"
Database Hints (Oracle)	CfgScript	View	sql-hint
Segmentation (multi-valued)	CfgScript	View	segment-by Value will be "value1,value2,...valueN"
Segment Check Interval	CfgScript	View	segment-check-interval
Segment Limit	CfgScript	View	segment-total-limit

Note: If you rename a View after its corresponding CfgScript object has been created, the object name in Configuration Server remains unchanged. Instead, the key Name in the Annex section Namespace and its value are set to the new name. Composer displays the changed name.

Workflow Block Update Detail

The following updates are written to Configuration Server for all Workflow blocks in the Project.

- A CfgScript object of the Enhanced Routing type is created under the current tenant/Scripts folder.
- In its annex, property url is created in section Application. The value is the URL of the generated SCXML document on the Composer web server (bundled Tomcat or local IIS). You can change this property using Configuration Manager or Genesys Administrator to set the correct value for the deployment environment.

In its annex, in section ApplicationParams, the key CustomerView_URL is added. Its value is in this format: [http:// http://]<configured CV host>:<configured CV port>. Context Services port and host values are picked up from Composer preferences.

Workflow Blocks and Publishing an IPD

When publishing an interaction process diagram (IPD) to Configuration Server, Workflow blocks are handled in two different ways:

Use Case #1: The Workflow block is dedicated to voice or interaction-less processing. In that case, you must use a stand-alone block (the block is not connected to any other block).

- When generating the code: Composer generates one SCXML file per such Workflow block (Name= IPD_<ipd file name>_<workflow block name>.scxml).

- When publishing: Composer creates one Enhanced Routing Script object (Name=<Project Name>.<IPD name>.<Workflow block name>) per such Workflow block in the IPD being published. The Application/url property of the ERS refers to the SCXML url (if deployed). The Workflow block Object Name property (read only property) is updated to the name of the Enhanced Routing Script object.

Use Case #2: The Workflow block is dedicated to multimedia processing. In this case, the block is connected (directly or indirectly) after a Workbin block or an Interaction Queue block.

- When generating the code: Composer generates one SCXML file per Interaction Queue/Workbin block (Name=IPD_<ipd file name>_<interaction queue block name>.scxml). If an IPD has an Interaction Queue block connected to multiple Workflow blocks (multiple views are defined on the Interaction Queue block), only one SCXML file is generated when generating the code for that IPD. This unique IPD SCXML is used to initiate the execution for all Workflow blocks. At runtime, the Workflow SCXML to execute is selected depending on the view the interaction is pulled from.

- When publishing: Composer creates one Interaction Queue Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>) per Interaction Queue block. Composer creates one Interaction Queue View Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>.<View name>) per Interaction Queue block defined view.

Composer creates one Enhanced Routing Script object (Name=<Project Name>.<IPD name>.<Interaction Queue block name>.Routing) per Interaction Queue block in the IPD being published. The Application/url property of this Enhanced Routing Script object refers to the Queue IPD SCXML url (if deployed). Composer does NOT create an Enhanced Routing Script object for the workflow blocks. The Workflow block Object Name property (read only property) is NOT updated.

Workbin Block Update Detail

The following updates are written to Configuration Server for all Workbin blocks in the Project.

- A CfgScript object of the Interaction Workbin type is created under the current tenant/Scripts folder. If the object already exists, its properties are updated.
- A CfgScript object of the Interaction Queue type is created under the current tenant/Scripts folder. The name of the object follows this format: <Workbin CfgScript object name>.PrivateQueue.
- A CfgScript object of the Interaction Queue View type is created under the current tenant/Scripts folder. The name of the object follows this format: <Workbin CfgScript object name>.PrivateView.
- A CfgScript object of the Enhanced Routing type is created under the current tenant/Scripts folder. The name of the object follows this format: <Workbin CfgScript object name>.PrivateQueue.Routing.
- A CfgScript object of the Interaction Queue View type is created under the current tenant/Scripts folder for each of this workbin user defined view. The name of the object follows this format: <Workbin CfgScript object name>.<view name>.

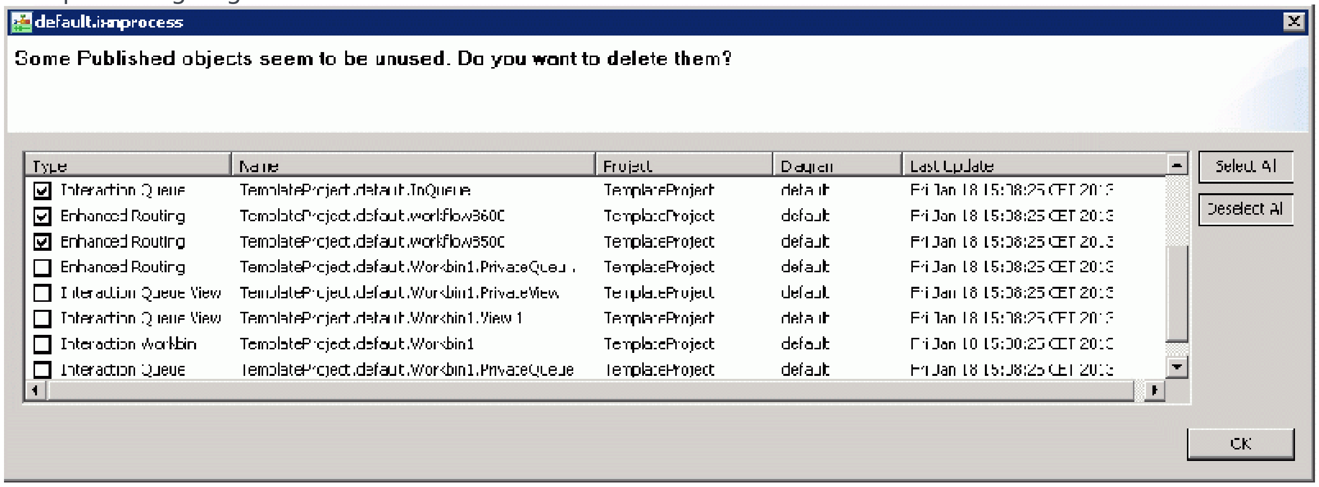
Deleting Items from Configuration Server

Configuration Server objects (Enhanced Routing, Interaction Queue, Interaction Queue View, Workbin) might be deleted from the Configuration Server in following situations:

- An interaction process diagram is deleted and that IPD contained blocks for which configuration objects were created. See also the Delete published objects when Interaction Process Diagram is deleted option. Note that Composer must be connected to the Configuration Server in order to be able to effectively delete the Configuration objects.
- A project containing interaction process diagrams is closed or deleted and those IPD contained block(s) for which configuration objects were created. See also the Delete published objects when Project is closed or deleted option. Note that Composer must be connected to the Configuration Server in order to be able to effectively delete the Configuration objects.

- An interaction process diagram is published and
    - some blocks of that IPD, for which configuration objects were created, have been deleted.
    - some blocks of that IPD, for which configuration objects were created, have been updated (for example some views were removed from an Interaction Queue block).

In all cases, the user is prompted for deletion confirmation for each Configuration Server object that Composer is going to delete.



Some Published objects seem to be unused. Do you want to delete them?

Type	Name	Project	Diagram	Last Update
☑ Interaction Queue	TemplateProject.default.InQueue	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☑ Enhanced Routing	TemplateProject.default.workflow3600	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☑ Enhanced Routing	TemplateProject.default.workflow3500	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☐ Enhanced Routing	TemplateProject.default.Workbin1.PrivateQueu	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☐ Interaction Queue View	TemplateProject.default.Workbin1.PrivateView	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☐ Interaction Queue View	TemplateProject.default.Workbin1.View 1	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☐ Interaction Workbin	TemplateProject.default.Workbin1	TemplateProject	default	Fri Jan 18 15:08:25 CET 201
☐ Interaction Queue	TemplateProject.default.Workbin1.PrivateQueue	TemplateProject	default	Fri Jan 18 15:08:25 CET 201

Route Flow Control Blocks

The table below summarizes the routing blocks used for flow control.

Assign Block	Use to assign a computed value/expression or a literal value to a variable
Attach Block	Use for attaching a specific interaction to the current Orchestration Server session.
Begin Parallel Block	Use to enable the design of multiple threads, such as running busy treatments in parallel files.
Branching Block	Use as a decision point in a callflow or workflow. It enables you to specify multiple application routes based on a branching condition. Depending on which condition is satisfied, the call follows the corresponding application route.
Detach Block	Use for detaching a specific interaction from the current Orchestration Server session.
Disconnect Block	Use to invoke the Cancel Call treatment, which ends the strategy and deletes the interaction from URS memory.
ECMAScript Block	Use to build an ECMA Script expression for routing decisions.
End Parallel Block	Use to mark the end of the threads that were started by a matching Begin Parallel block.
Entry Block	All routing strategy diagrams must start with an Entry block. Defines variables that can be shared across different blocks in the same workflow. The Entry block cannot have any incoming connections.
Exit Block	Use to terminate the workflow or to return control back to calling workflow in case of subworkflow (subroutine).
Log Block	Use to log information about the application; for example, caller-recorded input that is collected while the application is running, or error messages.
Looping Block	In cases where multiple records are returned, the Looping block can loop through all the records.
SCXML State Block	Use to write custom SCXML code for Composer to include in the SCXML document that it generates based on the workflow diagram.
Subroutine Block	Use to invoke external SCXML documents or a sub-workflow created using Composer.
User Data	Use to update an interaction's User Data.
Wait Event Block	Use to have ORS transition out when one of the defined events is received and the associated condition is true.

Assign Block Common

Use the Assign common block to assign a computed value/expression or an entered value to a variable.

See the Query Services block Service Data property for an example of using the Assign block and Expression Builder to parse a JSON string and assign the service data to a variable.

Note: Function getSIPHeaderValue(headername) returns the SIP header value associated with the given SIP headername. You may wish to use this function with the Assign block.

The Assign block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Assign Data Property

This property assigns a value (expression) to a variable. You select the variable and then enter an expression, either a literal or one created in Expression Builder.

To select a variable and assign a value:

1. Click the **Assign Data** row in the block's property table.
2. Click the ⋯ button to open the Assign Data to Variables dialog box.
3. Click in the **Variable** field to display a down arrow.
4. Click the down arrow and select a variable whose value will be evaluated to determine the branching condition. Default application variables are described in the Property Entry block for voice applications and the Property Entry block for routing applications. You can also use a custom variable.
5. Click under Expression to display the ⋯ button.
6. Click the ⋯ button to open Expression Builder. For examples of how to use Expression Builder, see the Expression Builder topic.

7. Select an operator for the branching condition.Your variable's value will be equal to (==), less than (<), greater than (>). less than or equal to (<=), greater than or equal to (>=) or not equal to (!=) to value you enter in the Expression field.
8. In the Expression field, create a value to compare to the variable's value. Enclose the value in single quotes (' ').
9. Click the button to validate the expression. Syntax messages appear under the Expression Builder title.
10. Click **OK** to close Expression Builder and return to the Assign Data to Variables dialog box.
11. You can make multiple variable/value assignments. Click the **Add** button if you wish to add more assignments and repeat the steps above.

Editing Expressions

To edit an expression:

1. Click its row under Expression in the Assign Data to Variables dialog box. This causes the button to appear.
2. Click the button to re-open Expression Builder where you can edit the expression.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

- For callflows, invalid ECMAScript expressions may raise the following exception event: `error.semantic`.
- For workflows, invalid ECMAScript expressions may raise the following exception events: `error.script.SyntaxError`, and `error.script.ReferenceError`.

You can use custom events to define the ECMAScript exception event handling.

Condition Property

Find this property's details under Common Properties for Callflow Blocksor Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Attach Block

Use the Attach block for attaching a specific interaction to the current Orchestration Server session. For more information see the *Orchestration Server Developer's Guide*, <attach> interface action element in Interaction Interface Action Elements. The Attach block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

This property provides the ability to define a set of exceptions handled by this block. Any exception not caught by a block in a thread might be caught by the enclosing Begin Parallel block. Find more detail under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Interaction ID Property

This property specifies the ID of the Interaction to attach to this Orchestration Server session. Set to a meaningful value or keep the default value, which is the system variable InteractionId. Find more details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Begin Parallel Block

Use this block to enable the design of multiple threads, such as running busy treatments in parallel files. A thread is a list of blocks that run one after another. Use the End Parallel block to mark the end of the threads that were started by a matching Begin Parallel block. Starting with 8.1.2, Composer removes the restriction on the type of blocks that can be used in a busy treatments chain in 8.1.0 and earlier. Blocks such as the ECMAScript block, Database blocks, and so on, are now usable in busy treatment chains. Blocks that work on an interaction or create new interactions may not be used as busy treatments.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

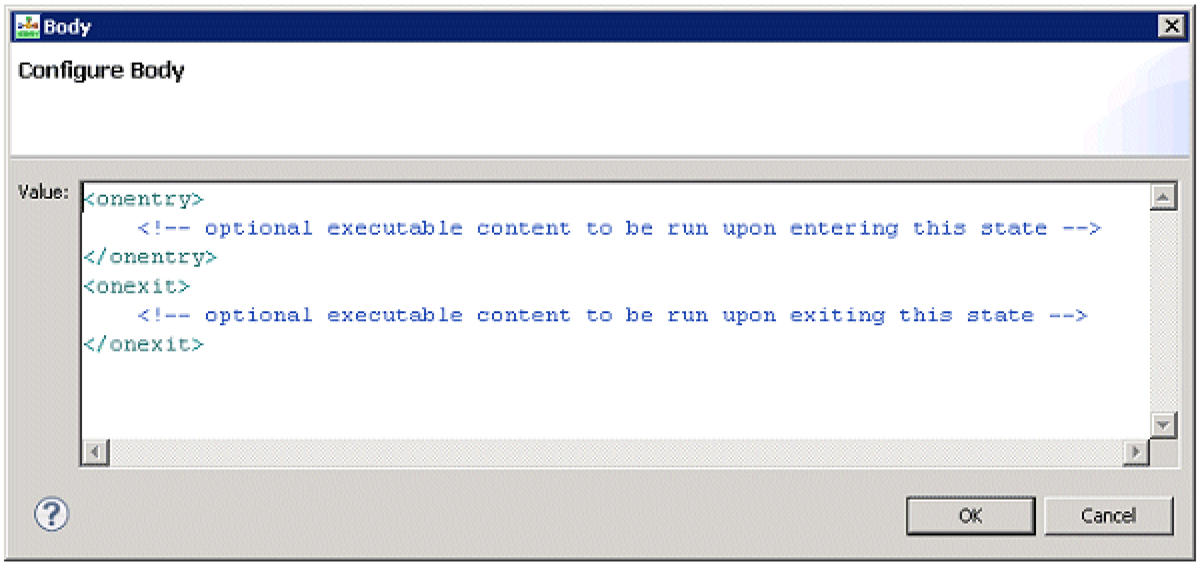
Log Level Property

Find this property's details under Common Properties.

Body Property

This property represents the SCXML that is mandatorily executed in the Parallel block before the parallel regions or legs begin to execute. This is useful for initialization of variables or other logic that should be completed before parallel regions begin to execute.

1. Click opposite **Body** under **Value**. This brings up the ⋯ button.
2. Click the ⋯ button to bring up the Configure Body dialog box.



```
<onentry>
    <!-- optional executable content to be run upon entering this state -->
</onentry>
<onexit>
    <!-- optional executable content to be run upon exiting this state -->
</onexit>
```

3. Enter the executable content of the <state> element. . All content (children) of the state are editable. You also have the option of adding code to <onentry> and <onexit>.
4. When through, click **OK**. Note: The editor does not validate against the SCXML schema.

Complete All Threads Property

This property controls when Orchestration Server is to transition out of the <parallel> structure.

Threads Property

block will have a Threads property to specify the number of threads to run in parallel. The number of outport will be automatically adjusted accordingly.

Enable Status Property

Find this property's details under Common Properties.

Branching Common Block

The Branching block is used for both routing and voice applications. For an example of using the branching block in a strategy, see the Example Workflow Diagram section in the Creating a New Workflow tutorial. Use the Branching block as a decision point in a callflow or workflow. It enables you to specify multiple application routes based on a branching condition. Depending on which condition is satisfied, the call follows the corresponding application route. A default path is automatically created once the conditions have been defined. If the application cannot find a matching condition, it routes the call to the default flow. **Note:** To support creating multiple views per interaction queue, the Branching block is available when creating an IPD.

Date/Time Functions

You can open Expression Builder from the Condition property and access the following date/time URS functions (**Data Category=URS Functions** > **Data Subcategory=genesys**):

- `_genesys.session.timeInZone(tzone)`
- `_genesys.session.dayInZone(tzone)`
- `_genesys.session.dateInZone(tzone)`
- `_genesys.session.day.Wednesday`
- `_genesys.session.day.Tuesday`
- `_genesys.session.day.Thursday`
- `_genesys.session.day.Sunday`
- `_genesys.session.day.Saturday`
- `_genesys.session.day.Monday`
- `_genesys.session.day.Friday`

The Branching block has the following properties:

Exceptions Property

The Branching block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties for Callflow Blocks Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Cancel Event Block

Use cancel a custom event. You specify the event name and a message, which is selection of a dynamic variable. It is a terminating block (can end an application instead of an Exit block). Orchestration Server 8.1.2+ versions are required for Raise and Cancel Event blocks.

The Cancel Event block has the following properties:

The Cancel Event block has no page exceptions.

Name Property

Find this property's details under Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks.

Condition Property

Find this property's details under Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Workflow Blocks.

Request Id Property

Select the variable containing the request ID.

Enable Status

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later.

Detach Block

Use the Detach block for detaching a specific interaction from the Orchestration Server session. For more information see the *Orchestration Server Developer's Guide*, <detach> interface action element. The Detach block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

This provides the ability to define a set of exceptions handled by this block. Any exception not caught by a block in a thread might be caught by the enclosing Begin Parallel block. Find more details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Interaction ID Property

This property specifies the ID of the Interaction to detach from this ORS session. Set to a meaningful value or keep the default value, which is the system variable InteractionId. Find more details under Common Properties.

Pass Context

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Pass Context Timeout

This property can be passed a positive integer value or a variable. This is the maximum time to wait (in seconds) for the destination session to read the originating session's context.

Enable Status Property

Find this property's details under Common Properties.

Disconnect Block Routing

Use to disconnect the caller and end the call. The Disconnect block invokes the CancelCall treatment, which ends the workflow and deletes the interaction from URS memory. The Disconnect block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Find more details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Single Session Treatments

When using the Play Application, Play Sound (Music and ARM Types) Exit, and Disconnect blocks, voice applications can now optionally use a single VXML session on Media Control Platform to play/run multiple treatments instead of using one session per treatment. This enables DTMF buffering between multiple MSML treatments. For more information, see Single Session Treatments.

ECMAScript Block

Orchestration Server (ORS) 8.0+ supports SCXML plus ECMAScript as a routing language. While the core SCXML provides State Chart functionality, you can specify ORS-specific instructions, such as conditions that can be used for routing decisions, in the form of ECMAScript. The Script property brings up Composer's Expression Builder for creating those conditions in the form of expressions. Use the ECMAScript block to build an ECMAScript expression. Notes:

- The ECMAScript block supports general ECMAScript in addition to ORS-specific ECMAScript functionality.
- If the Composer Project contains a folder at include/user, then any files with extension .js will be included in the generated SCXML. This allows you to write custom ECMAScript and include it in the application.
- To support creating multiple views per interaction queue, the ECMAScript block is available when creating an IPD.

The ECMA Script block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

- For callflows, invalid ECMAScript expressions may raise the following exception event: `error.semantic`
- For workflows, invalid ECMAScript expressions may raise the following exception events: `error.script.SyntaxError` and `error.script.ReferenceError`

You can use custom events to define the ECMAScript exception event handling.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Script Property

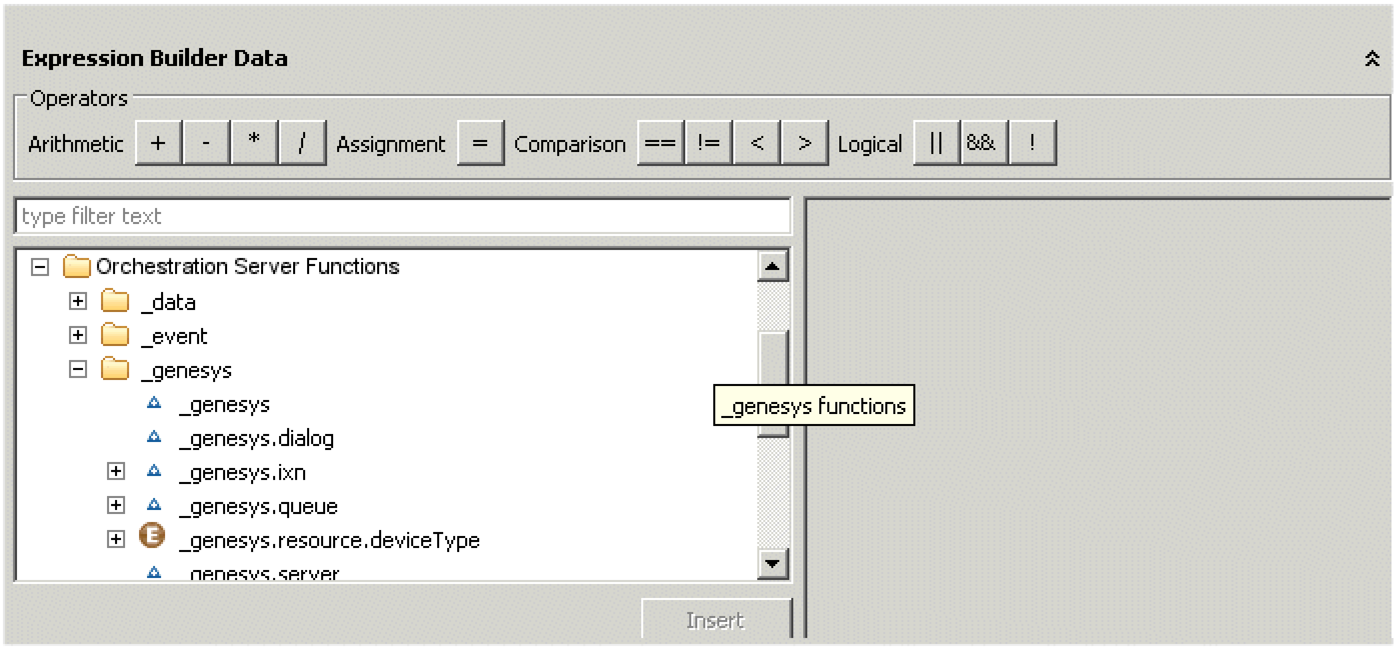
To create an ECMAScript expression in Expression Builder:

1. Click opposite Script under Value. This brings up the button.
2. Click the button to bring up Expression Builder.

Expression Builder gives access to various categories of data, which can be used in expressions. To create an expression, follow the instructions in the Creating Expressions topic.

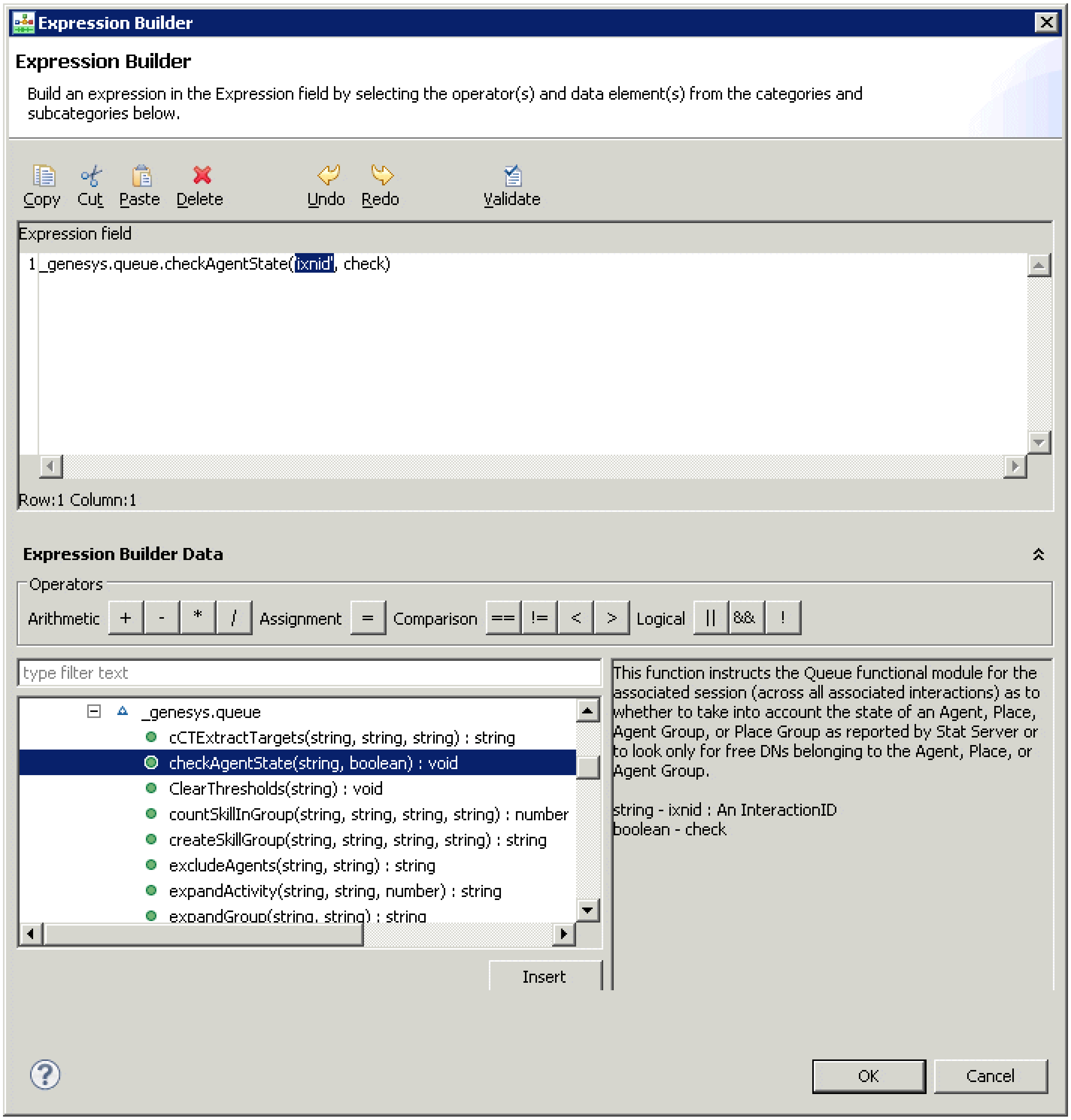
Using Genesys Functional Modules

Assume you expand Orchestration Server Functions in step 2 above. The lower Expression Builder Data area appears as shown below.



ECMAStart.gif

Orchestration Server Functions shows different categories of Genesys-supplied Functional Modules. For more information, see the Orchestration Server information on the Genesys Documentation Wiki. For example, assume you double-click genesys.queue.checkAgentState(check). Expression Builder now appears as shown below.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.

Expression field

```
_genesys.queue.checkAgentState(ixnid, check)
```

Row:1 Column:1

Expression Builder Data

Operators

Arithmetic + - * / Assignment = Comparison == != < > Logical || && !

type filter text

_genesys.queue
- cCTExtractTargets(string, string, string) : string
- checkAgentState(string, boolean) : void
- ClearThresholds(string) : void
- countSkillInGroup(string, string, string, string) : number
- createSkillGroup(string, string, string, string) : string
- excludeAgents(string, string) : string
- expandActivity(string, string, number) : string
- expandGroup(string, string) : string

This function instructs the Queue functional module for the associated session (across all associated interactions) as to whether to take into account the state of an Agent, Place, Agent Group, or Place Group as reported by Stat Server or to look only for free DNs belonging to the Agent, Place, or Agent Group.

string - ixnid : An InteractionID
boolean - check

ECMA1.gif

In this case, the genesys.queue module implements the target selection functionality of URS (finding resources for interactions and delivering interactions to the resource). The Interaction Routing Designer Function object precursor (not necessarily equivalent) is described in the Universal Routing

8.1 Reference Manual, CheckAgentState function. When URS executes Functional Modules, it returns events back to the instance of logic running the SCXML document that requested the action.

End Parallel Block

This block works with the Begin Parallel Block to enable the design of multiple threads. Use the End Parallel block to mark the end of the threads that were started by a matching Begin Parallel block.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Entry Block Routing

Use the Entry block to:

- Set global error (exception) handlers.
- Define global variables for the workflow.

All workflow diagrams must start with an Entry block, which cannot have any incoming connections. The Entry block is used as the entry point for a main workflow or subworkflow (subroutine). Composer throws a validation message if the Entry block is missing or more than one is added. The Entry block defines the initial entry state for interactions, all global state variables, and the datamodel. In the SCXML code for the workflow, all subsequent blocks are added as child states inside the Entry block's state. This allows the event handlers in the Entry block to act as global event handlers for the entire workflow. Any events not caught at the local level (for individual blocks) are caught at the global level by the Event handlers in the Entry block. The Entry block global variables define the State for the application and are maintained as the <datamodel> in the SCXML engine. The Back End block provides a Pass State property to pass state information to the Server side. An Entry block user-defined variable can be used to access the results of a Stored Procedure call specified in a DB Data block for both voice and routing applications. Note: Outlinks starting from the Entry block cannot be renamed or assigned a name through the Properties view. The Entry block has the following properties:

Name Property

Find this property's details under Common Properties.

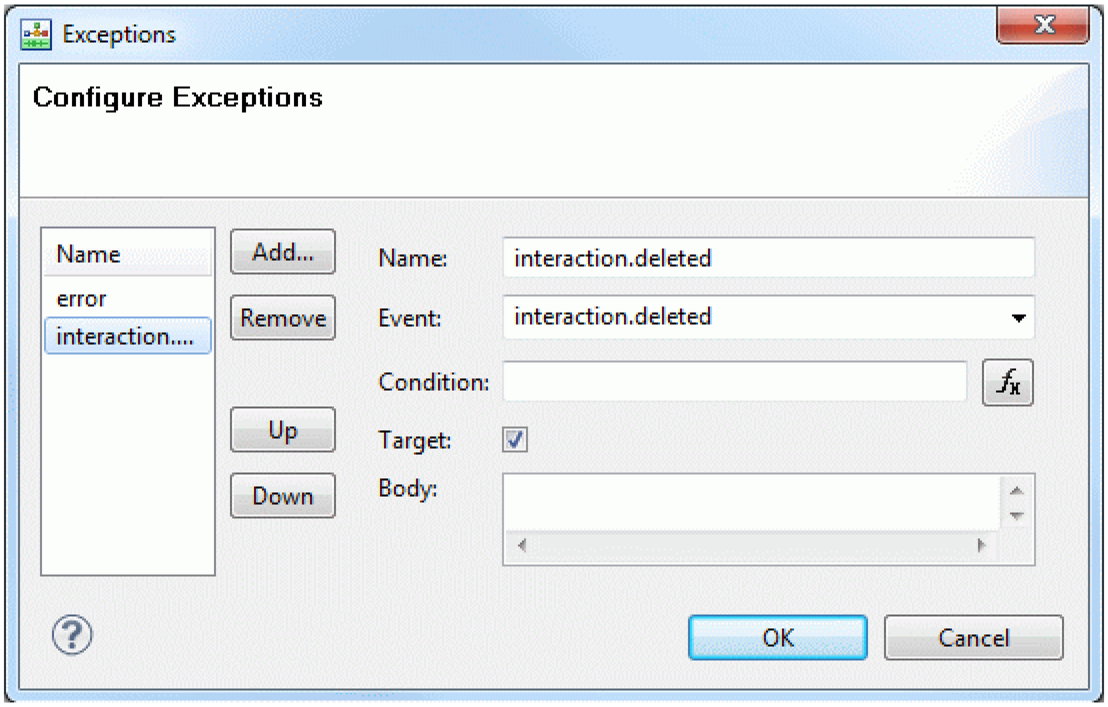
Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Use this property to define which exceptions or events to handle at the Entry block. This block contributes a state in the generated SCXML code that is a parent state for all blocks in a Workflow. This allows an event received for any of the workflow blocks to be handled at the Entry block.

1. Click opposite **Exceptions** under **Value**. This brings up the button.
2. Click the button to bring up the Exceptions dialog box.



Configure Exceptions

Name
error
interaction....

Name: interaction.deleted
Event: interaction.deleted
Condition:
Target: ☑
Body:

The Entry block `interaction.deleted` event is updated to have guard conditions:

- Current interaction deletion.
- The `interaction.deleted` event is from an interaction deletion and not from a detach operation.

```
_event.data.interactionid == system.InteractionID && (!_event.data.resultof ||
_event.data.resultof == 'deletion')
```

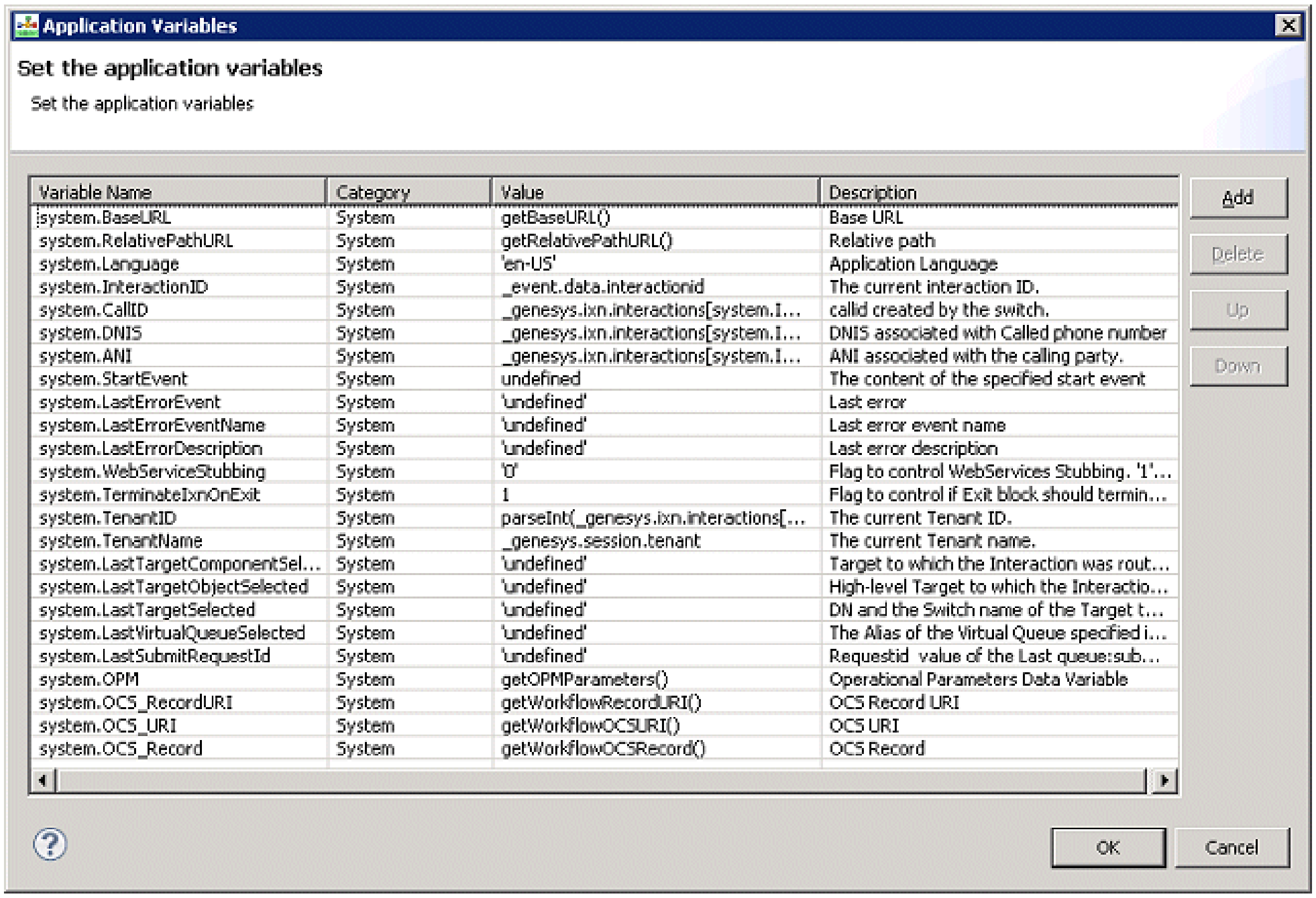
3. Click **Add** to add new exceptions to the list of handled exceptions.
4. For each exception, specify a unique name and an exception event.
   - **Name**--Composer uses the name of the exception to label the outport.
   - **Event**--Use to select the specific exception event.
   - **Condition**--The guard condition for this exception, which you define in Expression Builder. The exception is selected only if the condition evaluates to true.
   - **Target**--If true, an exception port is created and the user can connect it to the block this exception will transition to when it is executed. If false, the exception will not cause a change in the state configuration when it is executed. The executable content contained in the exception will still be executed, so the exception will function as a simple exception event handler.
   - **Body**--(optional) Executable scxml code that will be executed when this event is received and any specified condition evaluates to true. This code is executed before any other blocks that are connected as this exception's event handlers.
5. When done with the dialog box, click **OK**.

You can also find general information on the Exceptions property under Common Properties.

Variables Property

You have the option of defining workflow variables in the Entry block (Project variables are defined outside the Entry block via the toolbar). Variable descriptions entered in the Entry block are visible when selecting the variable for use in other blocks, such as the Assign block. Composer supports passing variables between a workflow and sub-workflow. To view variables:

1. In the Properties tab, click opposite Variables under Value to display the button.
2. Click the button to open a dialog box for defining and initializing global variables for the entire SCXML document.



Variable Name	Category	Value	Description
system.BaseURL	System	getBaseURL()	Base URL
system.RelativePathURL	System	getRelativePathURL()	Relative path
system.Language	System	'en-US'	Application Language
system.InteractionID	System	_event.data.interactionid	The current interaction ID.
system.CallID	System	_genesys.ixn.interactions[system.I...	callid created by the switch.
system.DNIS	System	_genesys.ixn.interactions[system.I...	DNIS associated with Called phone number
system.ANI	System	_genesys.ixn.interactions[system.I...	ANI associated with the calling party.
system.StartEvent	System	undefined	The content of the specified start event
system.LastErrorEvent	System	'undefined'	Last error
system.LastErrorEventName	System	'undefined'	Last error event name
system.LastErrorDescription	System	'undefined'	Last error description
system.WebServiceStubbing	System	'0'	Flag to control WebServices Stubbing. '1'...
system.TerminateIxnOnExit	System	1	Flag to control if Exit block should termin...
system.TenantID	System	parseInt(_genesys.ixn.interactions[...	The current Tenant ID.
system.TenantName	System	_genesys.session.tenant	The current Tenant name.
system.LastTargetComponentSel...	System	'undefined'	Target to which the Interaction was rout...
system.LastTargetObjectSelected	System	'undefined'	High-level Target to which the Interactio...
system.LastTargetSelected	System	'undefined'	DN and the Switch name of the Target t...
system.LastVirtualQueueSelected	System	'undefined'	The Alias of the Virtual Queue specified i...
system.LastSubmitRequestId	System	'undefined'	Requestid value of the Last queue:sub...
system.OPM	System	getOPMParameters()	Operational Parameters Data Variable
system.OCS_RecordURI	System	getWorkflowRecordURI()	OCS Record URI
system.OCS_URI	System	getWorkflowOCSURI()	OCS URI
system.OCS_Record	System	getWorkflowOCSRecord()	OCS Record

The Entry block lists all the variables associated with the workflow (referred to as global variables for the workflow). Composer supports the following types of variables for workflow diagrams in the Entry block:

- **Predefined system variables**, which cannot be deleted.
- User-defined variables local to the diagram file.
- Input variables, which are only used for Subroutines. These are user-defined and should be passed from the main diagram to the called Subroutine diagram file.

Also see:

- The Column Names and Records Variable properties used by the DB Data block.
- The Variable Naming section in the User Data block.

Important! When defining a variable name, the name:

- Must not start with a number or underscore.
- May consist of letters, numbers, or underscores.

When you define and initialize a variable that is expected to be played as a date later on in the workflow, define the value using the following format: yyyyymmdd. Example: MyDate=20090618. You must use this format; Composer does not perform any conversions in this case. When you define and initialize a variable that is expected to be played as a time later on in the workflow, define a 12 hour-based value using the following format: hhmmssa or hhmmssp. Examples: MyTime=115900a or MyTime=063700p. Define a 24 hour-based value using the following format: hhmmssh Example: MyTime=192000h. You must use this format; Composer does not perform any conversions in this case. **Default Application Variables** See the Variables: Project and Workflow topic. **Adding a New Variable** To add a new variable in the Application Variables dialog box:

1. Click **Add**. Composer add a row for variable and generates a temporary name and number; for example: var7.
2. Select the row and supply the **Name**, **Type**, **Value**, and **Description** fields.
3. Click OK.

Variable Name You can use the Variable name field for either of the following purposes:

- To enter the name of a new variable.
- To change the name of an existing variable. To do this, select an existing variable from the list of variables. The variable's name appears in the Variable box, and you can the change its value in the Value box.

Excluded Characters The Variable name field will not accept the following special characters:

- less-than sign (<)
- greater-than sign (>)
- double quotation mark ()
- apostrophe (‘)
- asterisk (*)
- ampersand (&)
- pound (#)
- percentage (%)
- semi colon (;)
- question mark (?)
- period (.)
- all characters that are considered ECMAscript operators; example: "-"

The variable Value field will not accept the following special characters:

- less-than sign (<)
- greater-than sign (>)
- double quotation mark ()
- apostrophe (‘)
- ampersand (&)
- plus sign (+)
- minus sign (-)
- asterisk (*)
- percentage (%)

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Exit Block Routing

Use to terminate the workflow or to return control back to calling workflow in case of a subworkflow (subroutine). Every strategy flow must have at least one Exit block (multiple Exit blocks are allowed, such as when using the Branching block). Composer generates a validation message if an Exit block is missing. If a workflow does not perform target selection and reaches the Exit block, the call is default routed to the default destination in Configuration Server, which is a configured DN. For more information, see option default_destination in the Universal Routing 8.1 Reference Manual. The Exit block has the following properties:

Name Property

The Name property is present in all blocks in Composer. The Name property is the first property for all blocks.

- Use the Value field beside in the Name property row of the block's property table to name the block. Block names should conform to ECMAScript and SCXML identifier naming conventions, and they are limited to a maximum of 255 characters.
- Names may consist only of numbers, letters, or initial underscores (_).
- Names should only begin with a letter or underscore.
- Names can end only with a letter or a number.
- Except for the Entry and Exit blocks, you should give all blocks a descriptive name. For example, if an Input block asks the caller to input an account number, then the name of the block could be Input_Account_Number.
- The name of the block is used as the Name of the <form> tag that gets generated for that block.

To provide a name for a block:

1. Select the Name row in the block's property table.
2. In the Value field, type a block name that conforms to the restrictions above.

Block Notes Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Return Values Property

This property is visible only for subworkflows. Use to specify the variable(s) whose value(s) will be returned once the Exit block is executed. To select return variables:

1. Click the Return Values row in the block's property table.
2. Click the ⋯ button to open the Return Values dialog box.

3. Select individual variables, or click Select all or Deselect all as needed.
4. Click OK to close the Return Values dialog box.

Single Session Treatments

When using the Play Application, Play Sound (Music and ARM Types) Exit, and Disconnect blocks, voice applications can now optionally use a single VXML session on Media Control Platform to play/run multiple treatments instead of using one session per treatment. This enables DTMF buffering between multiple MSML treatments. For more information, see Single Session Treatments.

Log Common Block

Use a Log block to record information about an application. For example, you can log caller-recorded input collected while an application is running or error messages. You can use the Log block for any of the following purposes:

1. Informational – To log the application specific data
2. Error – for logging error details
3. Warning – to flag any warnings
4. Debug – for debugging

The categories in the Log Level property correspond to the above.

When used for a callflow, the Log block writes the log to the Genesys Voice Platform logs (Media Control Platform) using the VoiceXML <log> tag.

The Log block has the following properties:

The Log block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

For callflows, invalid ECMAScript expressions may raise the following exception events: error.semantic. For workflows, invalid ECMAScript expressions may raise the following exception events:

- `error.log.ReferenceError`
- `error.illegalcond.SyntaxError`
- `error.illegalcond.ReferenceError`

You can use custom events to define the ECMAScript exception event handling.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Label Property

This property applies to workflows only. It provides meta-data for the logging details.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Looping Common Block

Use this block to iterate over a sequence of blocks multiple times in the following scenarios:

1. Iterate over a sequence of blocks based on a self-incrementing counter (FOR).
2. Iterate indefinitely until an exit condition is met (WHILE).
3. Iterate over records/data returned by the DB Data block (CURSOR/FOREACH). Also, populate variables if variables mapping is defined.
4. Iterate over data returned by Context Services blocks (FOREACH). Also, populate variables if Variables Mapping is defined.
5. Iterate over JSON Array defined in the application.

For scenarios 1 and 2 above, use the Looping block with a reference to the block retrieving the data. Scenarios 3 and/or 4 can be used in conjunction with 1 or 2, in which case the loop will exit when either of the exit conditions is met.

Prerequisite

You must perform the following steps in order for the Looping block to be used to iterate over data returned by the DB Data block:

1. Create a folder named Scripts in the Project folder.
2. In the Entry block, specify a value for the Scripts property such as ../include/DBRecordSetAccess.js

The Looping block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Counter Initial Value Property

A Counter variable controls the number of loops. Specify the initial value by entering a positive integer (including zero) or selecting the variable that contains the initial value. Composer will increment the Counter variable after each iteration. The value of the Counter variable is available after the looping has exited. This is a mandatory property if the Counter Variable property is specified.

Counter Variable Property

Enter a name for the variable used to store the Counter value or select the variable that contains the name. This is a mandatory property if the Counter Initial Value property is specified.

Current Record Variable Property

Select a variable to be used to store the current record when iterating over records. Composer will assign the current record after each iteration. This property is ignored if the Data Source Property is not set

Data Source Property

Specify a block reference to the DB Data or a Context Services block (with Variables Mapping support) that provides the data to be iterated or select the variable that contains a JSON Array. This is a mandatory property if Counter Initial Value and Counter Variable are not specified.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Counter Max Value Property

Specify the maximum value by entering a positive integer greater than the initial value or selecting the variable that contains the maximum value. When the Counter variable reaches the maximum value, then the block connected to the Exit port is executed. This is a mandatory property if the Counter Variable property is specified or if the Data Source property is not specified.

Exit Expression Property

This property is optional. If specified, prior to each iteration the exit expression is evaluated. If true, the flow goes out via the Exit port of the block. This condition is used in conjunction with max records (if Data Source is specified) or Counter Max Value (if Counter Variable is specified). To enter an exit expression

1. Opposite the **Exit Expression** property, click under **Value** to display the button.
2. Click the button to open Expression Builder. For examples of how to use Expression Builder, see the Expression Builder topic.
3. Create the exit expression and click **OK**.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

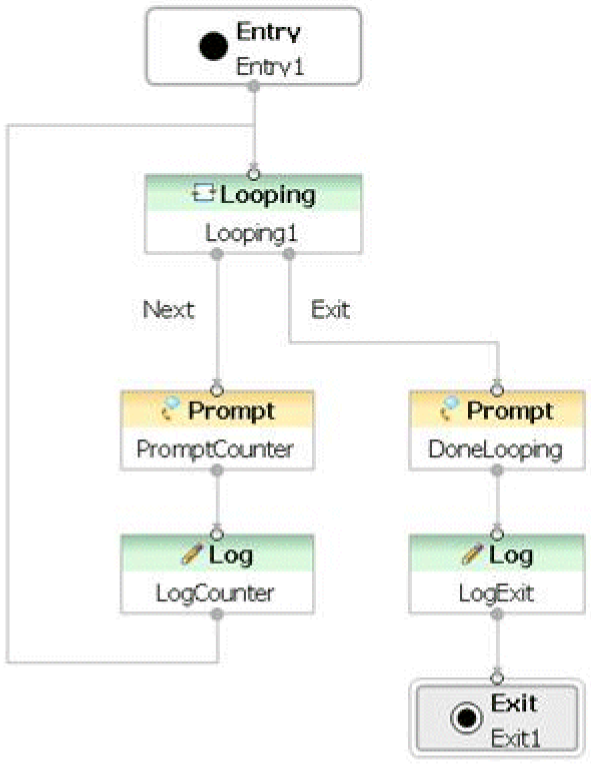
Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Using the Looping Block (Counter-based without a Data Source)

1. Add a Looping block and connect the previous block outport to the Looping block.
2. Connect the Next outport to the sequence of connected blocks.

3. Connect the outport of the last block in the sequence in step 2 back to the looping block to form a loop.
4. Connect the Exit outport of the looping block to the block(s) to continue processing after the loop has exited. The diagram when a looping block is used should appear as follows:



FOR loop: To iterate over the PromptCounter block 10 times, the following properties are set:

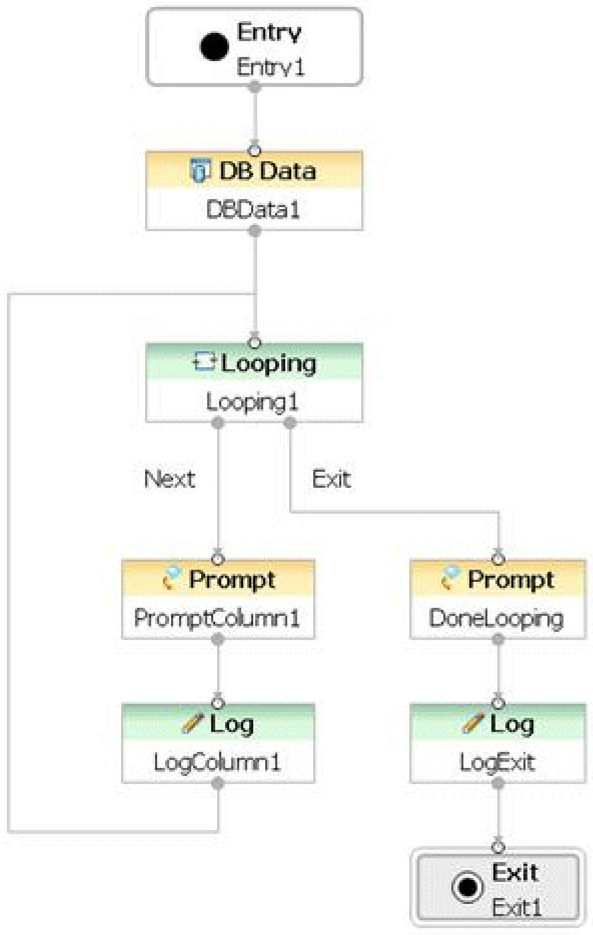
1. Counter Initial Value is 1.
2. Counter Variable Name is Variable(MyCounterVariable).
3. Counter Max Value is 10.

WHILE loop: To iterate over the PromptCounter block until a condition is satisfied, the following property is set: Exit expression is loginSuccessful != true.

Using the Looping Block with a DB Data/Context Services Block

1. Add a Looping block and connect the DB Data/Context Services block outport to the Looping block.
2. Connect the Next outport to the sequence of connected blocks.
3. Connect the outport of the last block in the sequence in step 2 back to the looping block to form a loop.
4. Connect the Exit outport of the looping block to the block(s) to continue processing after the loop has

exited. The diagram when a looping block is used should appear as follows:



CURSOR/FOREACH loop: To iterate over the PromptColumn1 block for each record returned by the DBData1 block, the following property is set: Data Source = Block Reference (DBData1) This example assumes variables were mapped for Column1 in DB Data1. If variables were not mapped, then another Assign block would be needed to store the value into a variable and the variable is then specified in the PromptColumn1 block.

Raise Event Block

Use the Raise Event block for Composer to throw custom events. You specify the event name and a message, which is selection of a dynamic variable. It is a terminating block (can end an application instead of an Exit block). Orchestration Server 8.1.2+ versions are required for Raise and Cancel Event blocks.

Also see Custom Events.

The Raise Event block has the following properties:

The Raise Event block has no page exceptions.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Delay Property

Enter the timeout or select a variable. Maps to <send delay>.

Unit Property

Select seconds or milliseconds for the delay. Maps to <send delay>.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Event Property

Select the variable or enter a value. Maps to <send event>.

Parameters Property

Add a list of key-values. Maps to <param>.

Enable Status

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later.

SCXML State block

Use to write custom SCXML code for Composer to include in the SCXML document that it generates based on the workflow diagram.

The SCXML State block has the following properties:

Name Property

Click under Value and enter the block name. Composer will use the name to identify the block in the diagram and as the state name in the SCXML code.

Block Notes Property

Find this property's details under Common Properties.

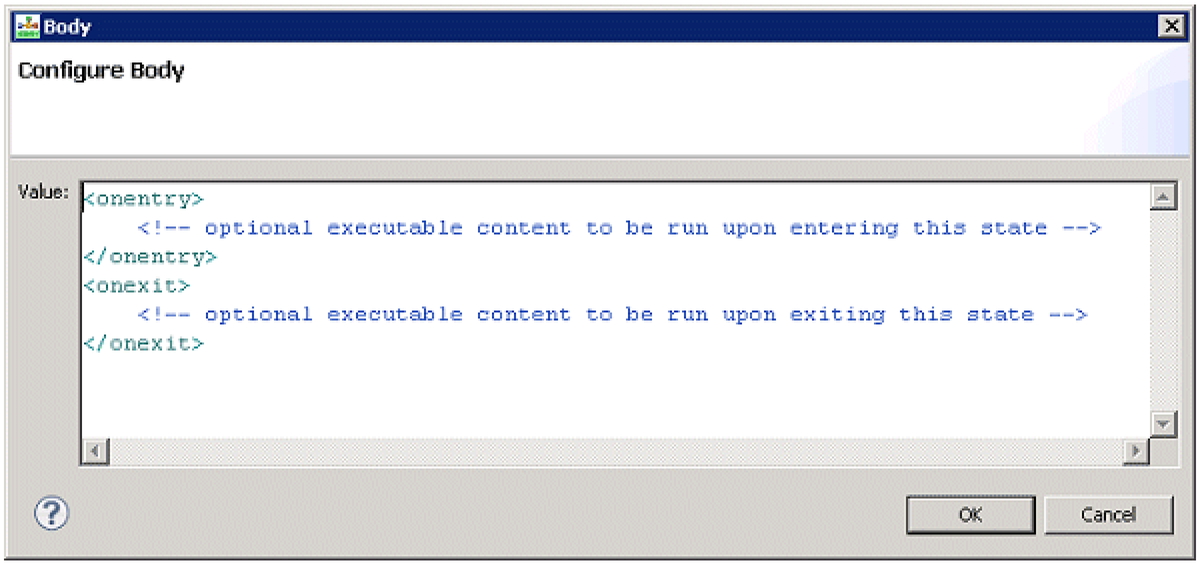
Exceptions Property

Use to configure the exception nodes, with each port being hooked up to an event configured by you and selectable using Add Custom Event. Find this property's details under Common Properties.

Body Property

This property contains all the executable content of the <state> element (<onentry>, <onexit>, <final>, ...).

1. Click opposite **Body** under Value. This brings up the ⋯ button.
2. Click the ⋯ button to bring up the Configure Body dialog box.



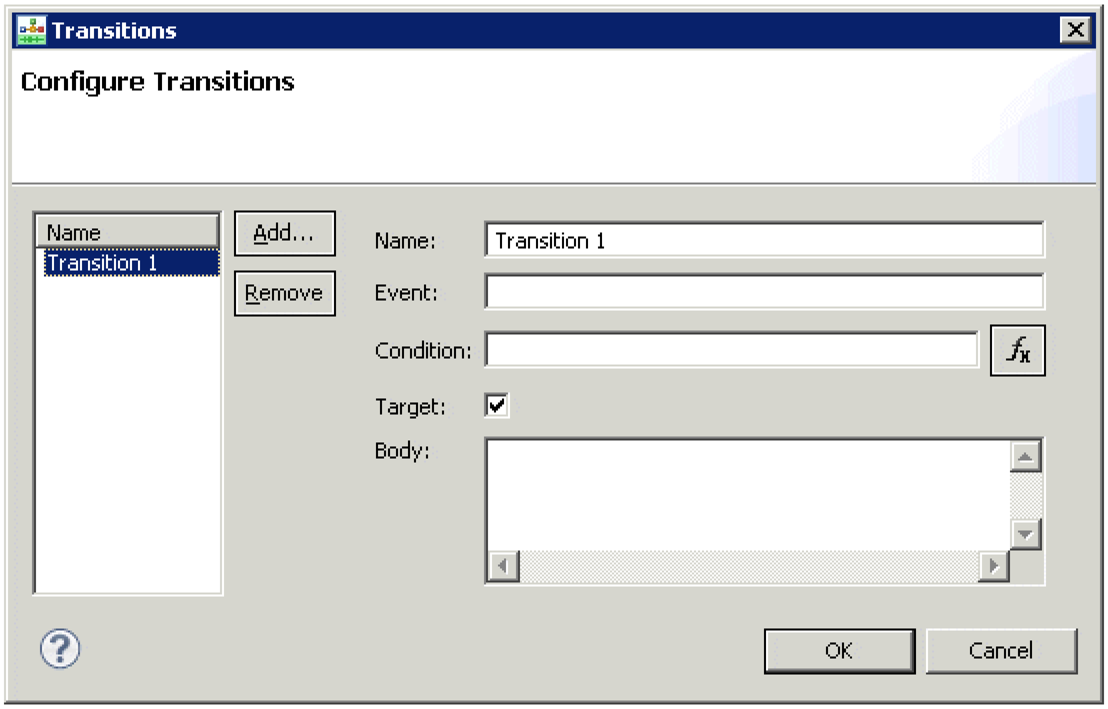
```
<onentry>
    <!-- optional executable content to be run upon entering this state -->
</onentry>
<onexit>
    <!-- optional executable content to be run upon exiting this state -->
</onexit>
```

3. Enter the executable content of the <state> element. . All content (children) of the state are editable. You also have the option of adding code to <onentry> and <onexit>.
4. When through, click **OK**. Note: The editor does not validate against the SCXML schema.

Transitions Property

Use this property to add additional outports (transitions) using the block's custom Transitions dialog.

1. Click opposite **Transitions** under **Value.** This brings up the button.
2. Click the button to bring up the Configure Transitions dialog box.
3. Click **Add**. The dialog box now appears as shown below.


Transitions

Configure Transitions

Name
Transition 1

Name: Transition 1

Event:

Condition:

Target: ✓

Body:

4. For each transition, specify at least one name, event, condition, or target (you are not required to complete all three fields). An output port is created for every transition

   - **Name**—Composer uses the name of the transition to label the outport.
   - **Event**—Use to select the event that will trigger this transition.
   - **Condition**—The guard condition for this transition. The transition is selected only if the condition evaluates to true.
   - **Target**—If true, an output port is created and the user can connect it to the block this transition will transition to when it is executed. If false, the transition will not cause a change in the state configuration when it is executed. The executable content contained in the transition will still be executed, so the transition will function as a simple event handler.

If a target is set, an outport for that transition will appear and you can connect it to other blocks. If a target is not set, an outport for that transition does not appear; in this case, you can add some SCXML code to handle the event.

When through in the dialog box, click **OK**.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

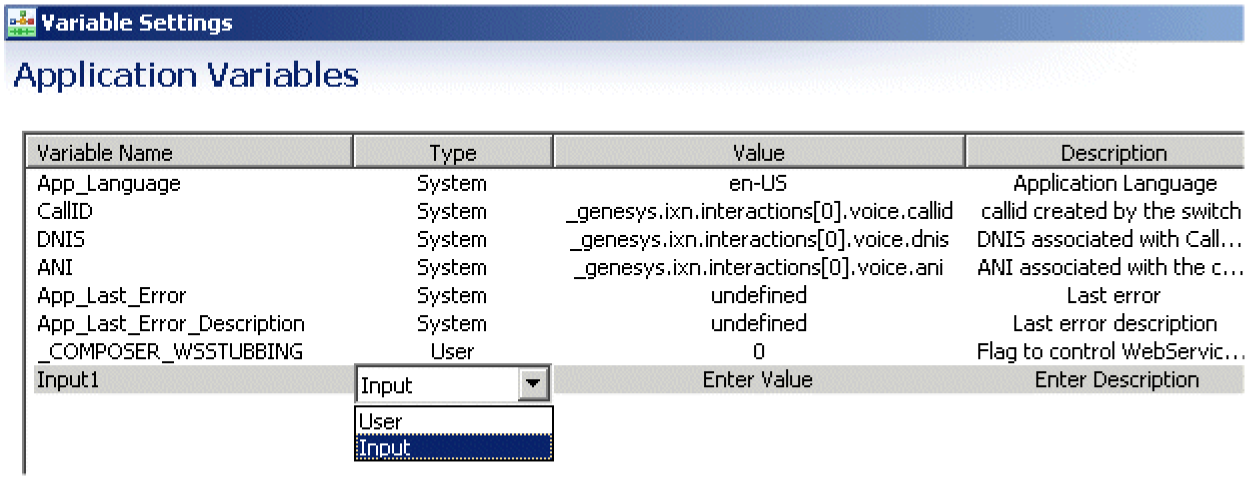
Subroutine Block

Use the Subroutine block to create reusable sub-modules (sub-workflows). You can invoke external SCXML documents or use a sub-workflow created using Composer. The input and output parameters names will be automatically picked from the sub-workflow created by Composer. Composer supports passing variables between a workflow and sub-workflow.

Also see Using Composer Shared Subroutines.

Creating a Subroutine Using A Sub-Workflow

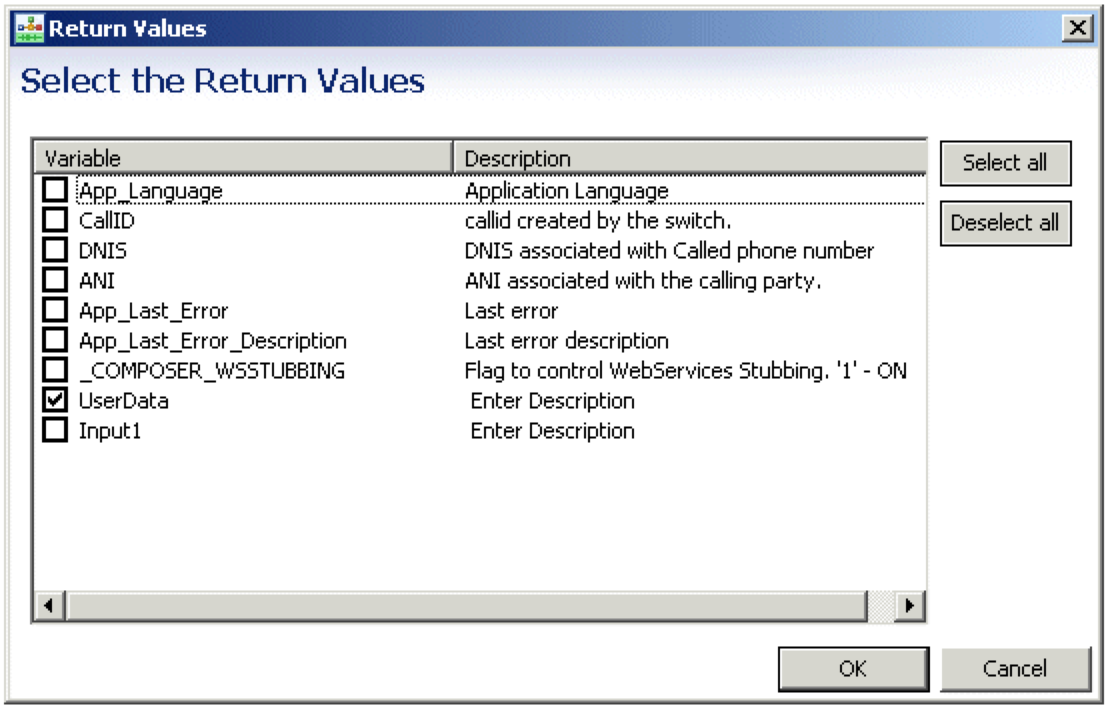
1. Create the main workflow diagram file using **New** > **Other** > **Composer** > **Workflow diagram** > **Main Workflow**.
2. After designing the main workflow diagram, create the sub-workflow diagram using **New** > **Other** > **Composer** > **Workflow diagram** > **Sub-Workflow**.
3. In the Entry Block of the sub-workflow diagram, enter the parameters, which will be passed as input from the main to the sub-workflow diagram.


Variable Settings

Application Variables

Variable Name	Type	Value	Description
App_Language	System	en-US	Application Language
CallID	System	_genesys.ixn.interactions[0].voice.callid	callid created by the switch
DNIS	System	_genesys.ixn.interactions[0].voice.dnis	DNIS associated with Call...
ANI	System	_genesys.ixn.interactions[0].voice.ani	ANI associated with the c...
App_Last_Error	System	undefined	Last error
App_Last_Error_Description	System	undefined	Last error description
_COMPOSER_WSSTUBBING	User	0	Flag to control WebServic...
Input1	Input	Enter Value	Enter Description

4. Design the sub-workflow diagram.
5. In the Exit block of the sub-workflow diagram, select the variables, which will be returned back to the called main diagram.

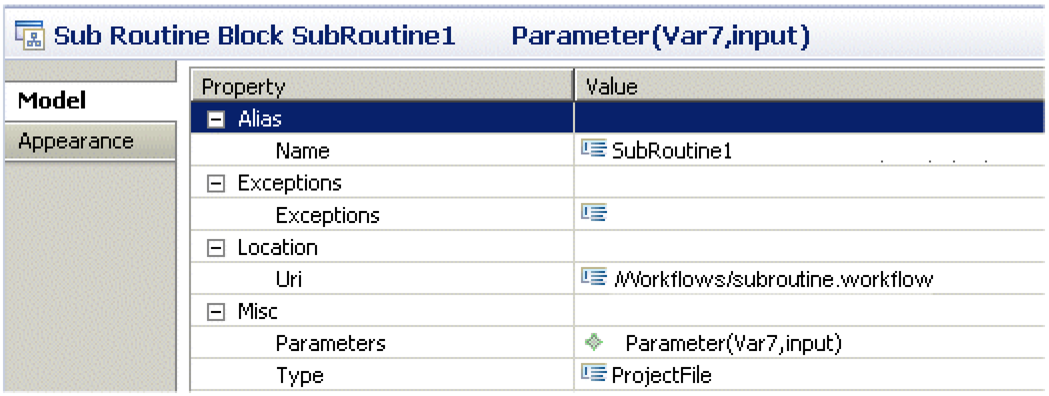


Return Values

Select the Return Values

	Variable	Description
☐	App_Language	Application Language
☐	CallID	callid created by the switch.
☐	DNIS	DNIS associated with Called phone number
☐	ANI	ANI associated with the calling party.
☐	App_Last_Error	Last error
☐	App_Last_Error_Description	Last error description
☐	_COMPOSER_WSSTUBBING	Flag to control WebServices Stubbing. '1' - ON
☑	UserData	Enter Description
☐	Input1	Enter Description

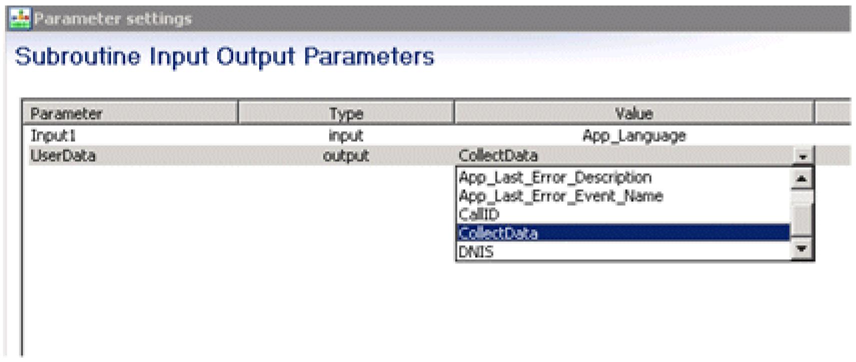
6. In the main diagram, use the Subroutine Block to call the newly created sub-workflow and the input and output parameters. For input/output synchronization, use the Uri property of the Subroutine block to select the sub-workflow diagram.



Sub Routine Block SubRoutine1 Parameter(Var7,input)

Property	Value
⊟ Alias	
Name	SubRoutine1
⊟ Exceptions	
Exceptions	
⊟ Location	
Uri	/Workflows/subroutine.workflow
⊟ Misc	
Parameters	Parameter(Var7,input)
Type	ProjectFile

Now the Parameters property can be used for the Parameter Synchronization. The main diagram implicitly parses the sub-workflow parameters and lists them in the Parameter settings dialog as shown below.


Parameter settings

Subroutine Input Output Parameters

Parameter	Type	Value
Input1	input	App_Language
UserData	output	CollectData

7. Define the value for the input type variables and collect the returning output type variables in a variable.

The Subroutine block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Uri Property

The Uri property specifies the destination (URL or Composer Project) depending on the value of the Type property. To set a URL destination for the Uri property **(Typ**e property is set to **UR**L):

1. Select the **Uri** row in the block's property table.
2. In the **Value** field, type a valid URL. Variables should not be selected as all subroutines are fetched by Orchestration Server before it starts executing the application at which time variables do not exist.

To set a Project destination for the Uri property (**Type** property is set to ProjectFile):

1. Click the **Uri** row in the block's property table.

2. Click the ⋯ button to open the Uri dialog box.
3. Select a workflow in the list.
4. Click **OK** to close the dialog box.

Type Property

The Type property sets the type of the invoked subroutine. There are two options:

- **URL**--The invoked sub-workflow can be found at the location specified in the Uri property.
- **ProjectFile**--The invoked sub-workflow is another workflow in the Project.

To select a value for the Type property:

1. Select the **Type** row in the block's property table.
2. In the **Value** field, select URL or ProjectFile from the drop-down list.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Parameters Property

Use the Parameters property to specify parameters to pass to the invoked sub-workflow. To specify parameters:

1. The URI field must contain a value.
2. Click the **Parameters** row under **Value**.
3. Click the ▭ button to open the Subroutine Input Output Parameters dialog box.
4. Click the **Add** button to enter parameter details.
5. In the Parameter field, accept the default name or change it.
6. From the Type drop-down list, select input, output, or inout:

input	Input parameters are variables submitted to the sub-workflow.
out	Output parameters are variables that the sub-workflow returns and will be reassigned back to the current workflow.
inout	Inout parameters are parameters that act as both input and output.

1. In the **Value** drop-down list, select from among the variables shown, type your own expression, or click the ▭ button to use Skill Expression Builder.
2. In the **Definition** field, type a description for this parameter.
3. Click **Add** again to enter another parameter, or click **OK** to finish.

Delete Button To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

User Data Block

You can use in a routing application to update an interaction's User Data and for attaching Business Attributes, Categories and Skills. Corresponds to the Universal Routing Server function _genesys.ixn.setuData(input,ixn) available in Expression Builder. This block generates ECMAScript inside an SCXML state and does not rely on External Service Protocol via <session:fetch>. For manually attaching categories to an interaction, the User Data block can be used and then a Branching block can be (optionally) used to segment interactions to different logical branches based on the different categories. Important! See Mandatory User Data For UCS Blocks.

Variable Naming

Do not assign the value of a variable named `data` to a key-value pair. This will not work since the generated code also declares a variable named `data`.

The User Data block has the following properties:

Name Property

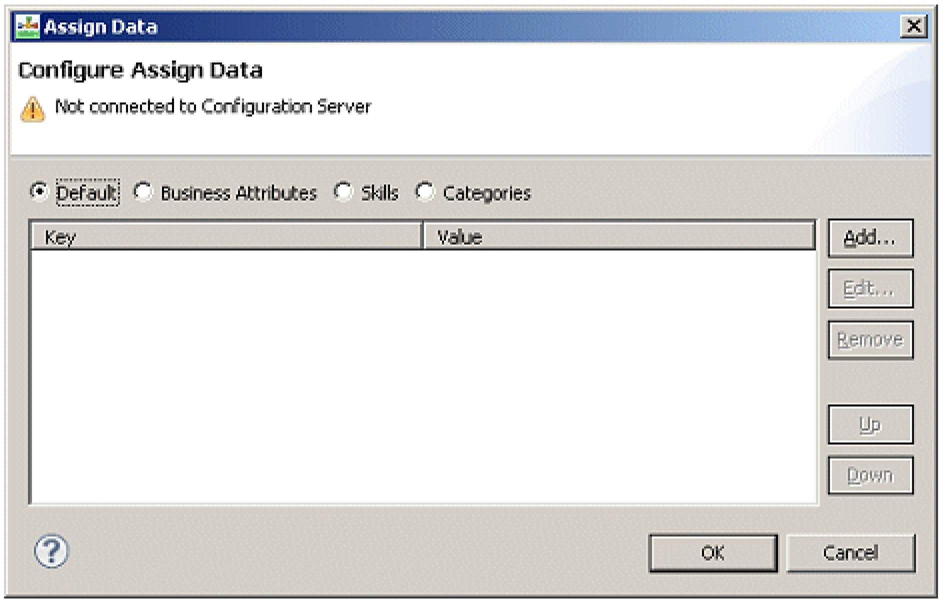
Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Assign Data Property

This property specifies one or more key-value pairs to add to the interaction's User Data. To specify hey-value pairs, click in the `Value` column to display the ⋯ button and click it to bring up the dialog. Data from various sources can be attached as well as free form key value pairs can be specified (Default). However, **note that only one category can be used in a block. Switching categories will erase previously specified values.**



UserData1.gif

1. **Default** This option can be used to specify both the key name and the key value as literals or variables.

Note: In case the key or value contains special characters that may require encoding e.g. '<', '-', or quotes, define a variable and set its value to this literal and use the variable as the value.

2. **Business Attributes** This picks up specific business attributes and if connected to Configuration Server, shows a list of values configured for these attributes.

3. **Skills** One or more skills can be specified. If connected to Configuration Server, a list of skills is shown.

4. **Categories** This option requires an active connection to Configuration Server and supports attaching categories defined in Configuration Server as well as **Relevancy** for the category. **Relevancy** is a number from 1 to 100 that reflects the minimum relevance percentage that each classification category must have in order for Classification Server to consider an interaction as belonging to that category.

Click the Add button to add a key-value pair of the selected type. When done, click Ok to dismiss the dialog and Save the diagram.

Exceptions Property

Find this property's details under Common Properties.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction. **Background:** Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/`Wait For Event=interaction.present` for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Wait For Event Property

This property allows you to choose whether to wait for the user data changed event before transitioning to the next block.

1. Click **Wait for Event** under **Property**.
2. Under **Value**, select one of the following:
    - **True**
    - **False**

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Wait Event Block

Use to have ORS transition out when one of the defined events is received and the associated condition is true. The Wait block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Event Property

Select the variable that contains the data associated with the event that makes ORS transition out.

Transitions Property

Use this property to define the events/condition pairs that makes ORS transition out. ORS does not need all the events in the list to occur in order to transition out, but only one of them. The condition is optional. If not set, it behaves as if condition was true (ORS transitions out of the block). To specify events/condition pairs:

1. Click the **Transitions** row in the block's property table.
2. Click the ⋯ button to open the Configure Transitions dialog box.
3. Click **Add** to add an entry.
4. Enter the Event.
5. Click under **Conditions**.
6. Click the ⋯ button to use Expression Builder to define the conditions.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Timeout Property

Select the variable that contains the timeout value.

Unit Property

Select the unit of measure for the timeout.

Routing Blocks

The table below summarizes the Routing blocks.

Default Route	Instructs URS to route a voice interaction to the default destination.
Queue Interaction	Places an a non-voice interaction into an existing queue.
Force Route	Force Universal Routing Server to route the interaction to the first target type without any other operations.
Route Interaction	Routes a non-voice interaction to one or more target objects.
Routing Rule	Selects routing rules that currently exist in the Configuration Database, such as those created with Interaction Routing Designer.
Stop Interaction	Requests Interaction Server to stop processing an interaction.
Target	Routes a voice interaction to a target. Can be used for percentage and/or conditional routing using threshold expressions.

Also see Statistics Manager and Builder.

Default Routing Block

Instructs URS to route the interaction to the default destination, as specified by the default DNs at the Routing Point or by the URS configuration option default_destination. When you use the Default Route block in a strategy, it sends the interaction to that destination. Once set this will be applicable for the entire duration of the strategy unless overridden by a Set Default Route block in the workflow execution.

The Default Route block has the following properties:

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Property Common Properties.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements referenced in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId.

Can be used for **interaction-less processing** for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.

Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS.

With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Diagram Wait_For_Event_Property) to initialize InteractionId, or not define an event at all.

For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case.

For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID.
2. And/or set the Interaction ID property to a meaningful value.
3. And/or assign a meaningful value to the InteractionId system variable.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing to the specified targets. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Pass Context Property

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Force Route Block

Use this block for both voice and multimedia interactions to force Universal Routing Server (URS) to route the interaction to the first target type (ACD Queue, Destination Label, or Routing Point) without any other operations. The interaction is then routed unconditionally, i.e., URS does not check the status of the destination. Warning! Force should always be thought of as a last plan of action and therefore used infrequently. The Force Route block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Destination Property

Use this property to specify the forced routing destination. The property also supports specifying a Resource type, which allows you to specify key-values. Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

From Property

A value expression, which returns the address that this interaction is to be redirected from. Set this property to the variable DNIS for voice interactions, or to the variable InteractionID for multimedia interactions. Composer will automatically set this property to DNIS or to InteractionID when the Destination property is set (respectively) to a Target Block or to a Route Interaction block. When the Destination property is not assigned a Block Reference value, you must select the appropriate From value.

1. Click under Value to display the ⋯ button.

2. Click the button to open the From dialog box.
3. Select one of the following:

    - **Literal.** For Value, you can specify:
        - An agent: <agent id>
        - A place: <place id>
        - A DN: <number>
        - An e-mail address: <username>@<host> or _origin or _origin.all or _udata
        - A customer number: <dn number>
        - A target format addresses: <Target DN>

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Variable**. If the variable contains a string, see Literal above. If the value is a JSON object, Value can refer to:
    - An agent: {agent: <agent id>, type:A}
    - An agent group: {agent: <name>, type:AG}
    - A place: {place: <place id>, type:AP}
    - A place group: {place: <name>, type:PG}
    - A DN: {dn: <number>, type:Q or RP or DN, switch:<switch name>}
    - An interaction queue: {id: <q name>, type:iq }
    - A workbin: {id: <wb name>, type:wb<owner>}
    - A customer number: {dn: <number>}
    - A target format addresses: Resource Object from the queue.submit.done event (the Target Block Resource Selected property).

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Configuration Server** to select the from Switch//DN if connected.
- **Resource** to select a resource using properties that will form a JSON object.

See the Orchestration Server Documentation Wiki.

4. Click **OK** to close the From dialog box.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for **interaction-less processing** for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction. **Background:** Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event IPD Wait_For_Event property to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing to the specified targets. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Type Property

Use to define the type of redirection processing that is to be done. For more information and individual values, see the Orchestration Server wiki.

Enable Status Property

Find this property's details under Common Properties.

Pass Context Property

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Force Routing to an External Number

Scenario: When force routing and doing a single-step consult to a routing point, where an external is dialed.

If you get these error messages:

```
expr='Error message: Cannot get link and/or device from call'. ‘invalid source”
```

Check that the `system.ThisDN` variable has the right value when it reaches the Force Route block.

Queue Interaction Block

Use this block to place a multimedia (non-voice) interaction into an existing queue created with the Interaction Queue block. The generated SCXML code instructs Universal Routing Server to request Interaction Server to move the current interaction into the specified queue. You can select an existing interaction queue from within this block or specify a variable that contains the name of an interaction queue. You can also send an interaction to a queue in another IPD within the same Project. The selected interaction queue appears as a Queue Reference block in the interaction process diagram. Important Note! Each interaction path in a workflow for multimedia interactions should end with one of these blocks: Stop Interaction, Queue Interaction, or Route Interaction. Also see information on the App Terminate Ixn On Exit variable.

Use Case

The logic of a routing workflow may determine some attributes of an interaction, such as by looking at the Subject line of an inbound e-mail, and then use different interaction queues as a method of segmenting these different types of interactions. You could use the Branching block for this purpose. The Queue Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Interaction Queue Name Property

Use this property to specify the queue where the interaction is to be placed.

1. Click under Value to display the ▭ button.
2. Click the button to open the Interaction Queue dialog box. Your dialog box may include existing IRD business processes with associated interaction queues underneath.
3. Select an interaction queue to which the incoming interaction has to be sent.
4. Click OK.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Diagram Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID

2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before queueing it. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Pass Context Property

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Route Interaction Block

Use this block to route a multimedia (non-voice) interaction to one or more target objects (use the Target block for voice interactions). You can also route to a target based on the value of a Skill expression. An interaction process diagram associates routing targets with queues, either interaction queues or virtual queues. When defining Route Interaction block properties, you have the option of selecting queues for both the existing interaction and a new outgoing interaction that may be created. You can also define a new interaction queue from within the block so you don't have to navigate away. Important Note! Each interaction path in a workflow for multimedia interactions should end with one of these blocks: Stop Interaction, Queue Interaction, or Route Interaction. Also see information on the App Terminate Ixn On Exit variable.

Use Case

- An inbound interaction hits a virtual route point, initiating a workflow routing strategy.
- The workflow strategy looks up data for the interaction in the customer database to determine the last agent who helped this customer and to determine the intent of the customer interaction.
- Next, the workflow sets priority to the interaction based on media type (for example, e-mail or chat) and customer segment (for example, Gold or Silver). Note: Setting the priority of an interaction is a Universal Routing Server function that is not directly related to target selection, but is normally done prior to target selection as a way to segment interactions.
- If the last agent exists, the workflow routes to the agent based on variable, setting a timeout of five seconds.
- If the last agent used is unavailable (timeout exceeded), the workflow routes to an Agent Group that is properly skilled to handle this type of interaction.
- There is an Escalation interaction queue configured as an outgoing interaction queue, so the agent can select this interaction queue from the desktop application in case he cannot handle the interaction himself and he needs to escalate it.

The Route Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki. Note: It can be used to specify extension data in Treatment blocks. To do this, use an ECMAScript block to create an ECMAScript object and assign properties to it. Then specify this object in the Hints property by selecting the variable whose content is a JSON object. This object is passed to ORS at runtime. For example, define variable `myExtensions` and myHints and set them as shown below in an ECMAScript block. `myExtensions={'NO_ANSWER_TIMEOUT':'10','NO_ANSWER_OVERFLOW':'recall','NO_ANSWER_ACTION':'notread` `myHints = { ‘extensions’ : myExtensions );` Then specify `myHints` as the value of the Hints property in the Route Interaction object. If necessary, see the Orchestration Server Documentation Wiki for more information on hints.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, `InteractionId`. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event IPD Wait_For_Event_property to initialize `InteractionId`, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=`interaction.present` for example), you may keep the default value for the Interaction ID property. The default value is the system variable `InteractionId`, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set, you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the `InteractionId` system variable

Target Component Selected

Select a variable containing the agent-level target to which the interaction was routed or should be routed to definitively.

- If the target specified in <submit> and selected for routing is of type Agent, Place, Queue, or Routing Point, this contains the target itself.
- If the desired target type is Agent Group, Place Group, or Queue Group, the function returns the agent, place, or queue from the corresponding group to which the interaction was sent.

The target format is `Name@StatServerName.Type`. The selected variable will be updated with the target information after receiving a `queue.submit.done`.

Target Object Selected

Select a variable containing the high-level target (one that you specify in a <submit>) to which the interaction was routed or should be routed to definitively. If a skill expression is used, the function returns: `?:SkillExpression@statserver.GA` or even `?GroupName:SkillExpression@statserver.GA`. The target format is `Name@StatServerName.Type.` The selected variable will be updated with the target information after receiving a `queue.submit.done`.

Target Queue Selected Property

Select a variable containing the DN and the switch name of the target to which the interaction was routed or should be routed to definitively. The selected variable will be updated with the target information after receiving a `queue.submit.done`.

Queue for Existing Interaction Property

Optional. To specify a queue for an existing interaction:

1. Click under **Value** to display the button.
2. Click **Add** to open the Queue for Existing Interaction dialog box. Do one of the following:
    - If you are connected to Configuration Server, select Configuration Server. Select the interaction queue from the **Value** field. This is the interaction queue to which the incoming interaction has to be sent. Queues listed here were previously defined with the Interaction Queue block.
    - Select **Literal** and enter the name of the interaction queue in the Value field.
3. Click **OK**.

Queue for Outgoing Interaction Property

Use to create a new interaction. Only one new interaction may be created. The agent must close the existing interaction with no further processing allowed.

1. Click under **Value** to display the button.
2. Click **Add** to open the Queue for Outgoing Interaction dialog box. Do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server**. Select the interaction queue from the **Value** field. This is the interaction queue to which any new interaction generated has to be sent. For example, the agent might create a new e-mail to a supervisor as a result of handling another interaction. Queues listed here were previously defined with the Interaction Queue block.
    - Select **Literal** and enter the name of the interaction queue in the **Value** field.
3. Click **OK**.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing to reserved targets. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Activity Property

Optional. Click the down arrow and select the variable that contains the Activity to be used for Workforce Management-based routing.

Clear Targets Property

Optional. Select **true** if targets listed in the object should be retained after the interaction moves on through the workflow and encounters other Route Interaction object. For more information, see the Clear Targets description for the voice interaction Target block.

Cut Off Time Property

Optional. Click the down arrow and select the variable that contains the cut-off time (in seconds) that defines the time window in which a potential target must be assigned to the activity defined in the Activity property above.

Include Requests from Previous Blocks Property

This property controls whether the block should transition if a target from previous Target block is selected even though it may not be specified in the current block. Set it to true for cascaded target lookups. If set to false, the block will wait until a target specified in the current block is selected for routing.

Priority Property

To use this optional property, click the down arrow under Value and select the variable, which contains a value expression returning the priority that the interaction will be given in the queue. For more information, see http://www.w3.org/TR/scxml/#ValueExpressions.

Pass Context Property

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Route Property

Use this property to set the SCXML <queue:submit> route attribute.

1. Click under Value to display the button.
2. Click the button to open the Route dialog box.
3. Select one of the following:
    - **Variable**. Then for **Value**, select the variable that contains either True (default) or False.
    - **Literal**. Then for **Value**, enter either **True** or **False**.

When set to false, the functional module will not attempt to route the associated interaction.

Statistic Property

Optional. If you wish to route based on the value of a statistic:

1. Click under **Value** to display the button.
2. Click the button to open the Statistic dialog box.
3. You can select a URS-predefined statistic, a custom statistic created with Statistics Builder, or a variable that contain the statistic.
4. Click **OK** when through in the dialog box.

In order to select a statistic, you must be connected to Configuration Server and have set the option to use URS Predefined statistics. For a listing of the URS predefined statistics, see the Statistics list for the Target block. For information on statistics applicable to multimedia interactions, start with the Routing Statistics chapter in the *Universal Routing 8.1 Reference Manual*.

Statistics Order Property

Optional. This property can work with the Statistics property. Select **Max** or **Min** to specify whether the interaction should be routed to the target with the maximum or the minimum value of the statistic.

Targets Property

Use this property for target selection.

1. If you have not already done so, connect to Configuration Server.
2. Opposite the **Targets** property, click under **Valu**e to display the ⋯ button.
3. Click the ⋯ button. The Targets dialog box opens.
4. Click **Add** in the Targets dialog box.
5. Click under **Type** to display a down arrow.
6. Click the down arrow and select the target type: Agent, AgentGroup, Place, PlaceGroup, Skill, or Variable.The selected target appears as a Dynamic Target block in the interaction process diagram.
7. Click under the Name field to display the ⋯ button.
8. Click the ⋯ button to bring up the Find Configuration Object. Targets of the selected type appear for selection. These target objects exist in the Configuration Database that you are connected to.
9. Select a routing target and click **OK**.
10. You have the option to add another target or click **OK** in the Targets dialog box.

Timeout Property

Optional. Enter the timeout number of seconds to specify the time an interaction waits for an available target. You can also specify a variable previously defined in the Entry object. If the timeout expires, the interaction is routed to the error.queue.submit.exception port.

Virtual Queue Property

Optional. A virtual queue a logical queue, not a physical queue. Interactions can be queued to a virtual queue if the specified targets are unavailable. To select a virtual queue.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the Virtual Queue dialog box.
3. Select an Alias, Switch, and Number. For more information on these fields, start with the Framework Configuration Manager Help.
4. Click OK.

Logged In Only Property

Select **true** or **false** to indicate whether only logged in agents can pull interactions out of this Workbin. Use to prevent logged out agents from pulling interactions. Selecting true = logged in agents only.

Workbin Property

Use this property if you wish to route this interaction to an agent workbin. To select a workbin:

1. Click under Value to display the button.
2. Click the button to open the Workbin Properties dialog box.
3. Select a workbin previously defined with the Workbin block.
4. Optional. Click the **Show** Unpublished **Workbins** box.
5. Click **OK**.

Routing Rule Block

Note: The Routing Rule block does not create routing rules. Instead, you select routing rules that currently exist in the Configuration Database, such as those created with Interaction Routing Designer as described in the *Universal Routing 8.1.x Reference Manual*.

Routing rules specify the method of target selection for voice interactions. Composer supports using the following types of routing rules:

- Load Balancing
- Percentage
- Statistics

The Routing Rule block has the following properties:

Name Property

Find this property's details under Property Common Properties. Enter the name of the Routing Rule block for the workflow. The name of the routing rule in the Configuration Database is entered in the Rule Name property below.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Property Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Pass Context Property

his property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Rule Name Property

Use this property to specify the routing rule name.

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Rule Name dialog box.
3. From the Type dropdown menu, select **Literal**, **Variable**, or **Configuration Server**
    - If you select **Literal**, enter the name of an existing routing rule exactly how it appears in the Configuration Database. In Configuration Manager, Routing Rules are stored in the Transactions folder.
    - If you select **Variable**, select the name of the variable that contains routing rule.
    - If you select **Configuration Server** and are Server connected, existing routing rules appear for selection based on the Rule Type property entry. Select a routing rule.
4. Click **OK** to close the dialog box.

Rule Type Property

Click the down arrow under Value and select one of the following:

- **Load Balancing**—Use to distribute voice interactions to Universal Routing Server (URS) queues according to statistic StatEstimatedWaitTime as described in the Universal Routing 8.0 Reference Manual.
- **Percentage**—Use to distribute voice interactions to targets based on a set percentage for each target. When this criterion is used, every target must be supplied with a non-negative integer percentage.
- **Statistics**—Use to route voice interactions. It uses a routing rule so that URS can obtain the values of defined statistics for targets from Stat Server.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId.

Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.

Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS.

With the introduction of support for Interaction-less processing, you can now define a specific event (for Event Property IPD Wait_For_Event_Property) to initialize InteractionId, or not define an event at all.

For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case.

For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing the interaction. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Single Step Transfer Block

Use this block for both voice and multimedia interactions to force Universal Routing Server (URS) to route the interaction to the first target type (ACD Queue, Destination Label, or Routing Point) without any other operations. The interaction is then routed unconditionally, i.e., URS does not check the status of the destination. Warning! Force should always be thought of as a last plan of action and therefore used infrequently. The Force Route block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Destination Property

Use this property to specify the routing transfer destination. Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

From Property

A value expression, which returns the address that this interaction is to be redirected from. Set this property to the variable DNIS for voice interactions, or to the variable InteractionID for multimedia interactions. Composer will automatically set this property to DNIS or to InteractionID when the Destination property is set (respectively) to a Target Block or to a Route Interaction block. This property also supports a Resource type,which allows you to specify key-values. For additional information, see the Force Route Block.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Interaction ID Property

This property specifies the ID of the Interaction to detach from this ORS session. Set to a meaningful value or keep the default value, which is the system variable InteractionId. Find more details under Common Properties.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing to the specified targets. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Pass Context

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (that is the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Pass Context Timeout

This property can be passed a positive integer value or a variable. This is the maximum time to wait (in seconds) for the destination session to read the originating session's context.

Enable Status Property

Find this property's details under Common Properties.

Stop Interaction Block

Use this block to send a request to Interaction Server to stop processing an interaction. This results in an notification to Universal Contact Server, or to a third party server, that processing for this interaction is finished. You can also:

- Select a reason code (configurable through Configuration Manager or Genesys Administrator), which is attached to the interaction.
- Indicate that the interaction should be deleted from the Universal Contact Server database.

Important Note! Each interaction path in a workflow for multimedia interactions should end with one of these blocks: Stop Interaction, Queue Interaction, or Route Interaction.

Also see information on the Variables Dialog Box App_Terminate_Ixn_On_Exit variable.

Use Case

- A customer support engineer is using Siebel to track customer support tickets. After a specific ticket is updated by the engineer, an action that he takes in Siebel triggers a Siebel workflow.
- That workflow initiates an event that generates, using the Genesys Open Media API, an interaction process diagram, which in turn, triggers a routing workflow.
- The workflow specifies that an e-mail should be created.
- The workflow specifies that the e-mail should be sent to the customer, from a Customer Support e-mail address, copying specific interested parties.
- If the process of sending the e-mail fails for any reason, the e-mail can be queued to a different queue (for example, so it can be manually reviewed and perhaps the customer will be notified by phone instead).
- If the process of sending out the e-mail is successful, the workflow would specify that the processing of the interaction should be stopped, and will provide the appropriate reason. This information will be part of a notification that is provided to Universal Contact Server or to a third party server.

The Stop Interaction block has the following properties:

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Property Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Notify Property

Use this property to select a Universal Contact Server or third party media server to notify about the stop processing action.

1. Click under Value to display the ▭ button.
2. Click the button to open the Notify dialog box.
3. Under **Notify**, you have the option of checking the box to delete the interaction from Universal Contact Server Database.
4. Click one of the following buttons: **Notify UCS** or **Notify Third Party Server**.

5. Under **Details**:

If you selected **Notify Third Party Server**:

- Opposite **Application Type**, click the down arrow to select the type in the Configuration Database. See the Application Type note below.
- Opposite **Application Name**, click the button. In the resulting Notify dialog box, select **Literal** if you wish to manually enter the name. Select Configuration Server if you wish to select it from the **Value** field.
- Opposite **Service,** click the button to enter the type of notification service for the third party server.
- Opposite **Name**, click the button to enter the name of the notification service for the third party server.

If you selected **Notify UCS**, select the Application Name of the Universal Contact Server. The remaining fields are disabled.

Note on Application Type

The supported Application Type names are:

- **ChatServer**
- **ClassificationServer**
- **ContactServer,**
- **EmailServer**
- **GenericClient**
- **GenericServer**
- **ThirdParyApp**
- **ThirdPartyServer**

Interaction Server treats any connection in its Connections list as an ESP server, except some well known types, like DBServer, Stat Server, T-Server, and URS. Any new Application Type will also be treated as an ESP server. No special configuration is required.

With Application Name, if there is a connection to it from Interaction Server, then the specific server will be called (or its backup). If only type is specified, Interaction Server will select any one of the applications in its Connections list having the specified type. This works fine with specific types, like E-mail server or Chat Server, but for generic types, there can be multiple different applications with the same generic type. So to call a generic server, the name must be specified.

Load balancing can still be used. Starting from 7.6.1 ESP server clusters are supported. The cluster application name (with type ApplicationCluster) can be specified in ESP request. In this case, Interaction Server will load balance between applications the cluster has connections to. Interaction Server will also always check that the ESP server supports the required tenant (has it in tenants list) and will only load balance between ESP servers that support required tenant. </li>

6. Click **OK** to close the Notify dialog box.

Use Notify Property

Select **true** to notify Universal Contact Server or the third party server about the stopping the processing of the interaction including the reason.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId.

Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.

Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS.

With the introduction of support for Interaction-less processing, you can now define a specific event (for Event Property IPD Wait For Event property) to initialize InteractionId, or not define an event at all.

For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case.

For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

- Not use blocks that require an Interaction ID
- And/or set the Interaction ID property to a meaningful value
- And/or assign a meaningful value to the InteractionId system variable

Reason to Stop Interaction Property

Use this property to get the stop processing reason code.

1. Click under **Value** to display the ⋯ button.
2. Click the button to open the Reason to Stop Interaction dialog box.
3. Opposite **Type**, select one of the following as the source for the reason code:
    - **Literal** to enter the reason code manually in the Value field.
    - **Configuration Server** to select a predefined reason code as Business Attribute in the Value field.

  - **Variable** to select a variable for the reason code in the Value field.

4. Click **OK** to close the dialog box.

Target Block

Use to route an voice interaction to a target (use Route Interaction for multimedia interactions). Also use for percentage routing and conditional routing using the threshold functions. The Target block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Clear Targets Property

Click the down arrow and select true or false (default). If set to true, URS retains the targets listed in the block are after the interaction moves on through the strategy and encounters other Target blocks.

- This option is only applicable if more than one Target block is used along the same path in the strategy. It has nothing to do with how agents specified within the block are evaluated for availability. If no additional Target blocks are included in the path and no agent is available, the interaction is default-routed. If more than one agent is available, URS uses the statistic specified in the Statistics property to determine which agent receives the interaction.

If Clear Targets = false and the current interaction encounters a Target block later in the strategy, the targets in this block are added to those of the next Target block. If Clear Target = true, the targets are not added to the list of any Target block that the interaction encounters later in the strategy. For example, assume two Target blocks are used in a strategy with the first Target block configured with three agents and a timeout of 30 seconds and the second Target block configured with four new agents and a timeout of 60 seconds. The Clear Targets setting determines which agents are considered as targets for an interaction.

- Clear Targets = true. In the first Target block, URS waits for 30 seconds for any of the three agents to become available. If none are available before the timeout expires, the strategy resumes. When the second Target block is encountered, URS only considers these four new agents as possible targets during the 60-second timeout. URS does not consider (clears) the first three agents as possible targets.
- Clear Targets = false and none of the agents listed in the first Target block are available. When the interaction encounters the second Target block, URS considers the first three agents and the four new agents as possible targets during the 60-second timeout set in this object. If no more Target blocks follow the second Target block and no agents are available before the timeout expires, the interaction is routed to the error port.

Condition Property

Find this property's details under Common Properties.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before routing to reserved targets. When this property is set to true, the interaction is detached from the current session.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Enable Status Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties. Note: Exceptions for Busy treatment blocks should be handled in the Target block to which they are connected and not in the Busy treatment blocks themselves. Busy treatment exceptions are raised as the error.queue.submit exception and not as exceptions listed in individual treatment blocks.

From Property

A value expression, which returns the address that this interaction is to be redirected from. Set this property to the variable DNIS for voice interactions, or to the variable InteractionID for multimedia interactions. Composer will automatically set this property to DNIS or to InteractionID when the Destination property is set (respectively) to a Target Block or to a Route Interaction block. When the Destination property is not assigned a Block Reference value, you must select the appropriate From value.

1. Click under Value to display the ... button.

2. Click the button to open the From dialog box.
3. Select one of the following:

    - **Literal.** For Value, you can specify:

        - An agent: <agent id>
        - A place: <place id>
        - A DN: <number>
        - An e-mail address: <username>@<host> or _origin or _origin.all or _udata
        - A customer number: <dn number>
        - A target format addresses: <Target DN>

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Variable**. If the variable contains a string, see Literal above. If the value is a JSON object, Value can refer to:

    - An agent: {agent: <agent id>, type:A}
    - An agent group: {agent: <name>, type:AG}
    - A place: {place: <place id>, type:AP}
    - A place group: {place: <name>, type:PG}
    - A DN: {dn: <number>, type:Q or RP or DN, switch:<switch name>}
    - An interaction queue: {id: <q name>, type:iq }
    - A workbin: {id: <wb name>, type:wb<owner>}
    - A customer number: {dn: <number>}
    - A target format addresses: Resource Object from the queue.submit.done event (the Target Block Resource Selected property).

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Configuration Server** to select the from Switch//DN if connected.
- **Resource** to select a resource using properties that will form a JSON object.

See the Orchestration Server Documentation Wiki.

4. Click **OK** to close the From dialog box.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Include Requests from Previous Blocks Property

This property controls whether the block should transition if a target from previous Target block is selected even though it may not be specified in the current block. Set it to true for cascaded target lookups. If set to false, the block will wait until a target specified in the current block is selected for routing.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Set to a meaningful value or keep the default value, which is the system variable InteractionId. Find more details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Pass Context Property

This property accepts true/false values. When set to true and Detach is also true:

- URL built with the block name is stored into this interaction's user data (user data key name is '_composer_originating_session') just before detaching the interaction. That URL will be used by the orchestration destination session (the new orchestration session started to handle the interaction after it was redirected to an other routing point) to request the context of the originating session. After the processing for this block is over, the originating session is blocked until the destination session actually reads the context. The context consists of the system and user variables.

Resource Selected Property

Click the down arrow and select the variable that specifies the target resource for the interaction.

Target Component Selected Property

Select a variable containing the agent-level target to which the interaction was routed or should be routed to definitively.

- If the target specified in <submit> and selected for routing is of type Agent, Place, Queue, or Routing Point, this contains the target itself.
- If the desired target type is Agent Group, Place Group, or Queue Group, the function returns the agent, place, or queue from the corresponding group the interaction was sent to.

The target format isName@`StatServerName.Type`. The selected variable will be updated with the target information after receiving a queue.submit.done.

Target Object Selected Property

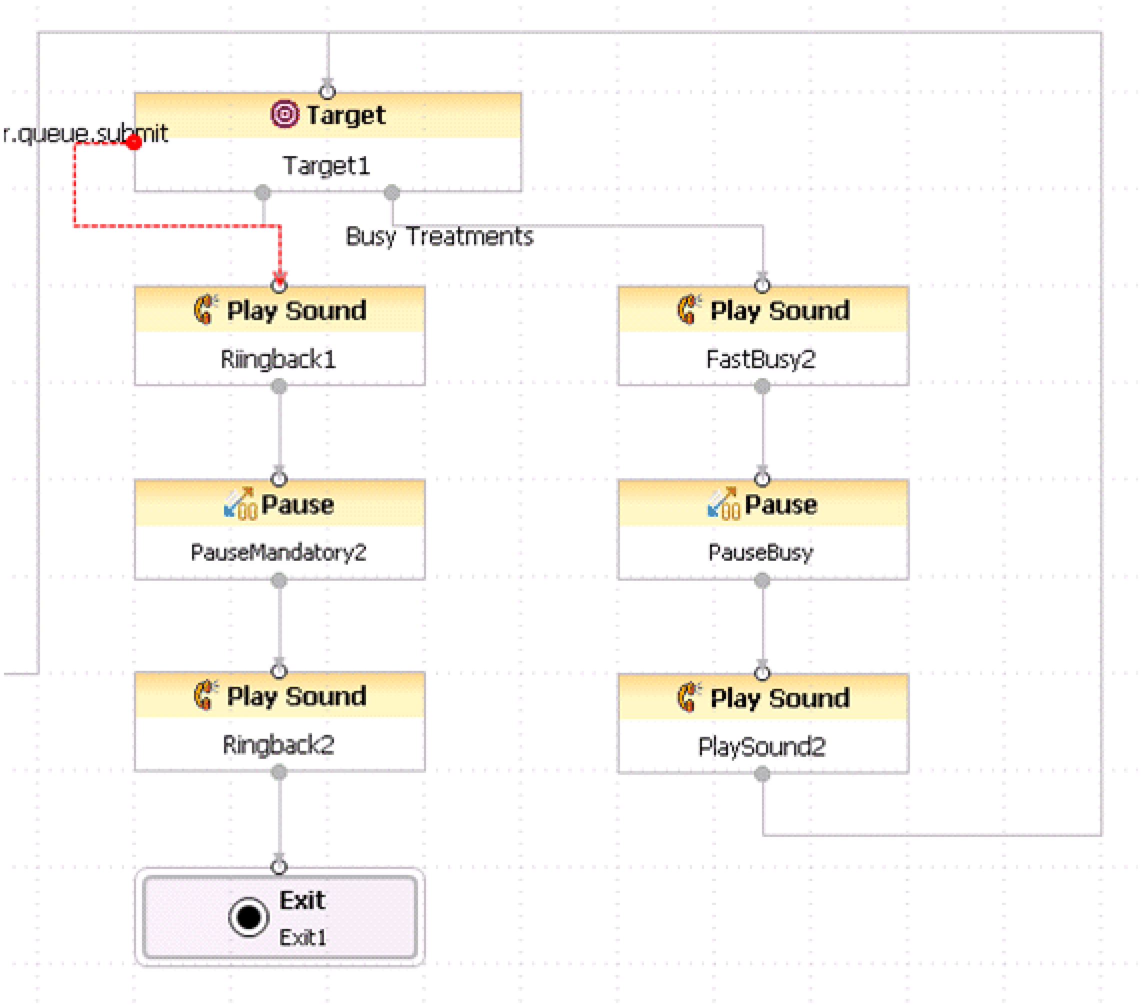
Select a variable containing the high-level target (one that you specify in a <submit>) to which the interaction was routed or should be routed to definitively. If a skill expression is used, the function returns: `?:SkillExpression@statserver.GA` or even `?GroupName:SkillExpression@statserver.GA`. The target format isName@`StatServerName.Type`. The selected variable will be updated with the target information after receiving a queue.submit.done.

Target Selected Property

Select a variable containing the DN and the switch name of the target to which the interaction was routed or should be routed to definitively. The target format is Name@SwitchName.Type. The selected variable will be updated with the target information after receiving a queue.submit.done.

Use Treatments Property

You can specify Busy treatments that will be executed while waiting for targets to become available. To enable Busy treatments, set this property to **true**. Busy treatments are similar to Mandatory treatments that you can specify using individual Voice Treatment blocks. This property provides instructions to Universal Routing Server for handling the interaction before an eligible target is available to receive it. Treatments (or Busys treatments) are played to callers while the call is waiting in a queue. You can specify any number of treatments for an interaction. The waiting interaction receives each treatment for the specified duration, in the specified order. Any time that URS finds an available target for the call, the previously-started and currently-played busy treatment (and the entire Busy treatments chain) will be interrupted in favor of routing the call to a target. Target availability is checked upon starting a treatment, on updates from Stat Server, and usually every 2 seconds (the value of this time interval is configurable by the URS option pulse_time). In order to specify treatments, this property must be set to **true**. Setting to true displays a new Busy treatments outport on the Target block. As shown below, you can connect treatment blocks to this outport that should be used for Busy treatments. Connect the last Busy treatment block back to its Target block. This indicates the completion of the Busy treatments loop.



Notes: Busy treatment blocks cannot handle exceptions or be used for exception handling. Diagram validation will check that the loop is completely correctly and that only appropriate blocks are used for Busy treatments.

Priority Property

To use this optional property, click the down arrow under Value and select the variable, which contains a value expression returning the priority that the interaction will be given in the queue. For more information, see http://www.w3.org/TR/scxml/#ValueExpressions.

Route Property

Use this property to set the SCXML <queue:submit> route attribute. Also see the Use Access Code property below.

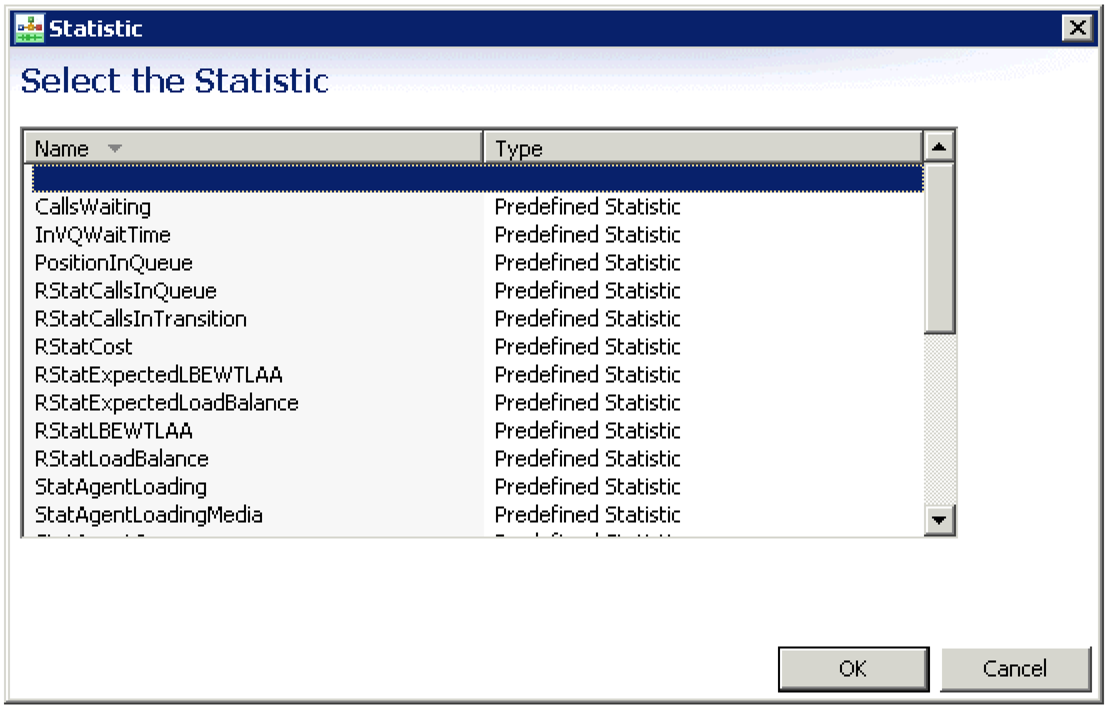
1. Click under Value to display the button.
2. Click the button to open the Route dialog box.
3. Select one of the following:
    - **Variable**. Then for Value, select the variable that contains either True (default) or False.
    - **Literal**. Then for Value, enter either True or False.

When set to False, the functional module will not attempt to route the associated interaction.

Statistic Property

Optional. If you wish to route based on the value of a statistic:

1. Click under Value to display the button.
2. Click the button to open the Statistic dialog box.


Statistic

Select the Statistic

Name	Type
CallsWaiting	Predefined Statistic
InVQWaitTime	Predefined Statistic
PositionInQueue	Predefined Statistic
RStatCallsInQueue	Predefined Statistic
RStatCallsInTransition	Predefined Statistic
RStatCost	Predefined Statistic
RStatExpectedLBEWTLAA	Predefined Statistic
RStatExpectedLoadBalance	Predefined Statistic
RStatLBEWTLAA	Predefined Statistic
RStatLoadBalance	Predefined Statistic
StatAgentLoading	Predefined Statistic
StatAgentLoadingMedia	Predefined Statistic

3. You can select a URS-predefined statistic, a custom statistic created with Statistics Builder, or a variable that contain the statistic.

4. Click OK when through in the dialog box.

In order to select a statistic, you must be connected to Configuration Server and have set the option to use URS Predefined statistics. Below are the URS predefined statistics:

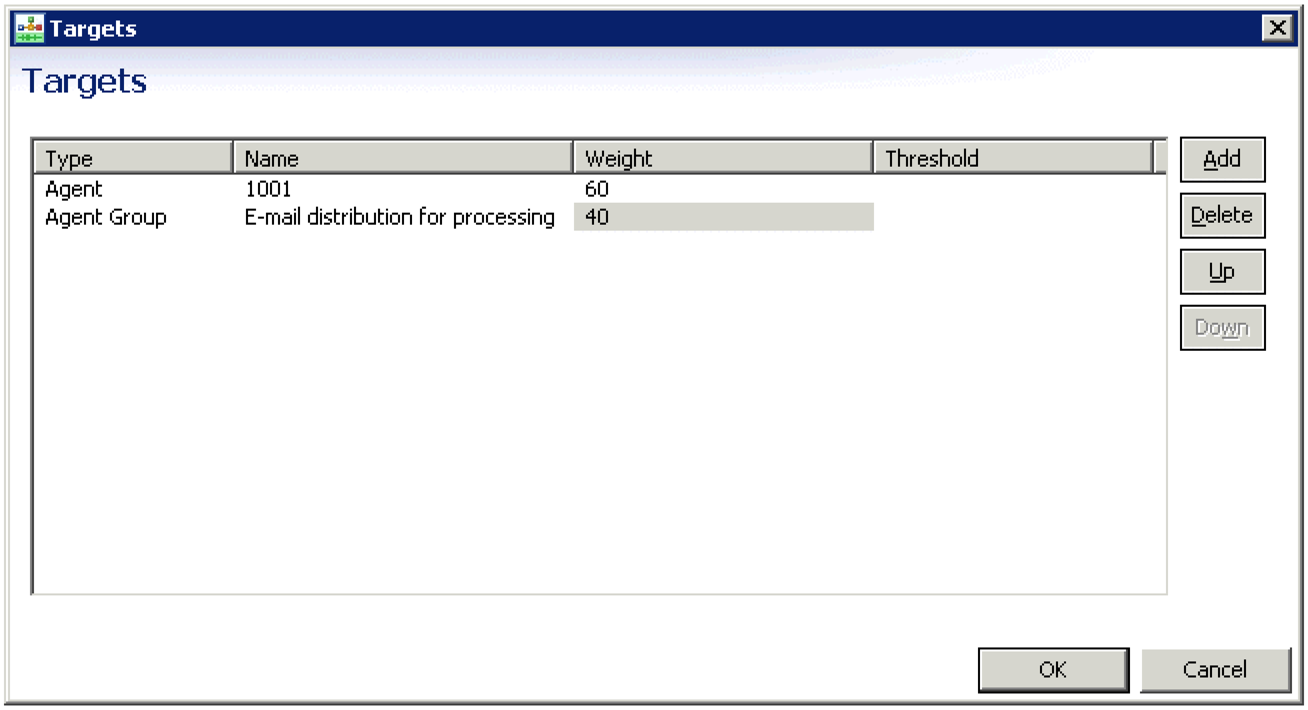
CallsWaiting	RStatLoadBalance	StatTimeInReadyState
InVQWaitTime	StatAgentsTotal	StatAgentsAvailable
PositionInQueue	StatCallsAnswered	StatAgentsBusy
RStatLBEWTLAA	StatCallsCompleted	StatAgentsInQueueLogin
RStatCallsInQueue	StatCallsInQueue	StatAgentsInQueueReady
RStatCallsInTransition	StatEstimatedWaitingTime	StatAgentLoading
RStatCost	StatExpectedWaitingTime	StatAgentLoadingMedia
RStatExpectedLBEWTLAA	StatLoadBalance	StatAgentOccupancy
RStatExpectedLoadBalance	StatServiceFactor	

For a definition of each statistic and recommended usage, consult the chapter on routing statistics in the *Universal Routing 8.1 Reference Manual*. The statistic you select is used by Universal Routing Server to determine which target to route the interaction to if more than one target is available. After defining a complete set of available agents (taking agent capacity rules into consideration, if configured), URS applies the selection criteria specified in the Target block, which can include using the minimum or maximum value of the statistic (see Statistics Order property).

Statistics Order Property

This property can work with the Statistics property.

- Select Max or Min to specify whether the interaction should be routed to the target with the maximum or the minimum value of the statistic.
- Select to route interactions to targets based on a percentage allocation. This selection causes the dialog box that opens from the Targets property to display a column where you can specify a percentage for each target.



Type	Name	Weight	Threshold
Agent	1001	60	
Agent Group	E-mail distribution for processing	40	

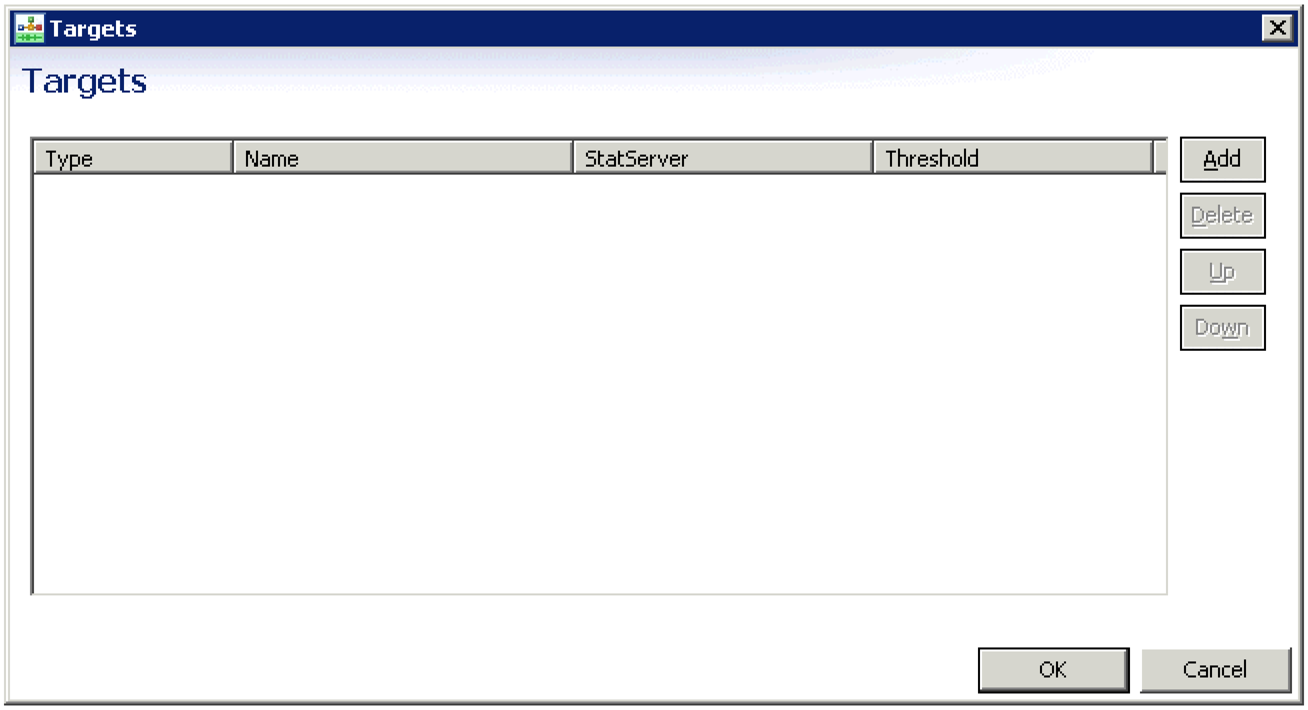
Targets Property

Use this property to specify routing targets and for percentage or conditional routing using the Resource Finder.

1. If you have not already done so, connect to Configuration Server.
2. Opposite the Targets property, click under Value to display the button.
3. Click the button. The Targets dialog box opens.
4. Click Add in the Targets dialog box.
5. Click under Type to display a down arrow.
6. Click the down arrow and select the target type: Agent, AgentGroup, ACDQueue, Campaign Group, Destination Label, Place, PlaceGroup, Queue Group, RoutePoint, Skill, or Variable.

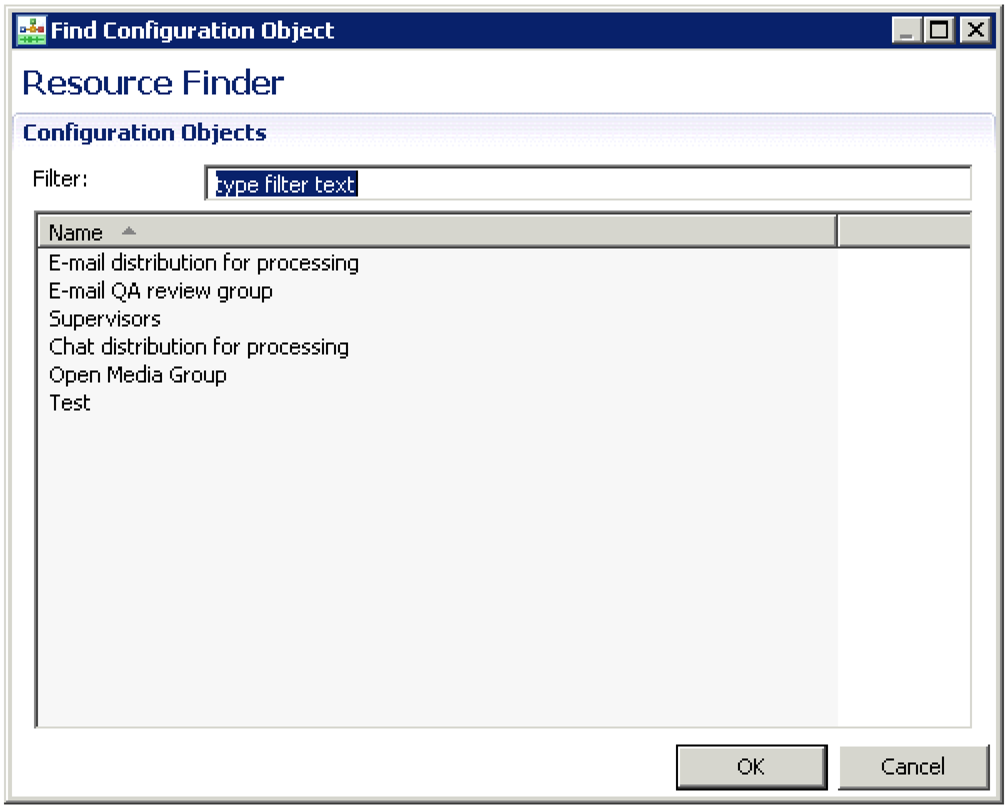
7. Click under the Name field to display the button.

The figure below shows the Targets dialog box when Min or Max is selected for the Statistics Order property. In this case, a Stat Server column appears.



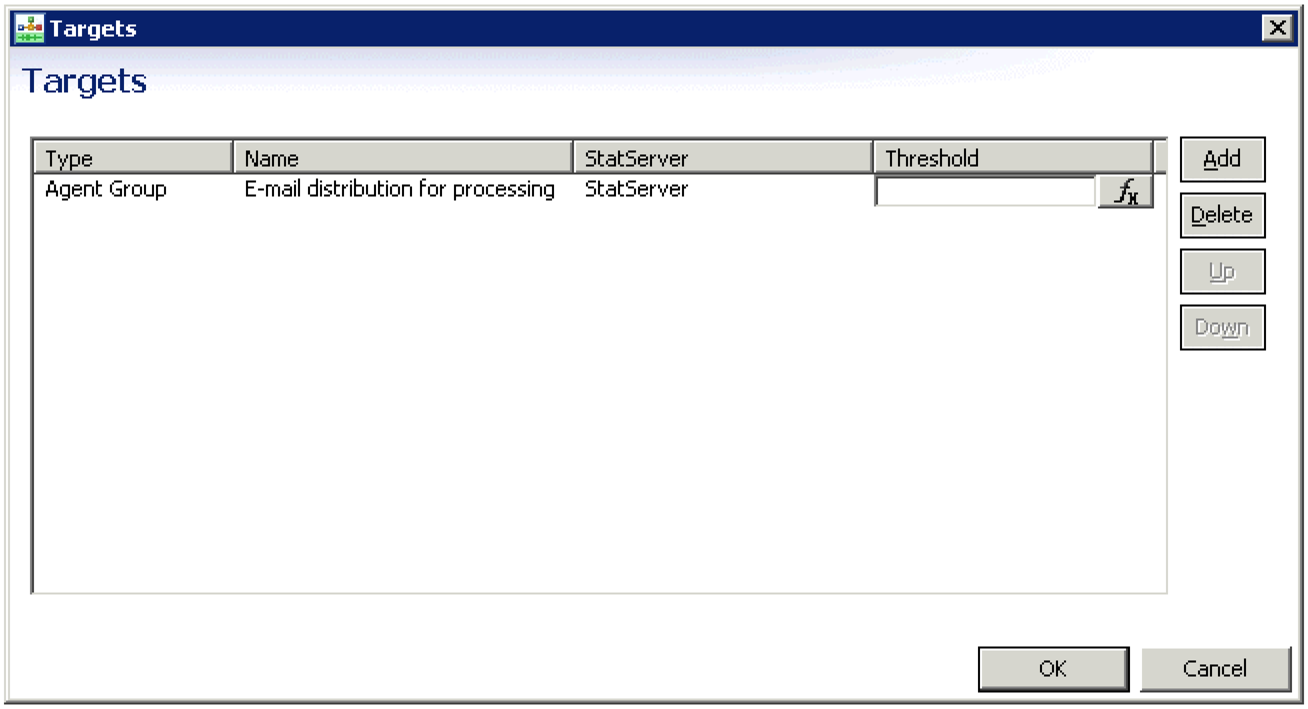
If you select Percentage for the Statistics Order property, a Weight column appears instead of a Stat Server column (see Statistics Order property above).

1. Click the button to bring up the Resource Finder. Targets of the selected type appear for selection. An example is shown below.



E-mail distribution for processing
E-mail QA review group
Supervisors
Chat distribution for processing
Open Media Group
Test

2. Select a routing target and click **OK**.
3. Select a Stat Server. An example completed dialog box is shown below.



Type	Name	StatServer	Threshold
Agent Group	E-mail distribution for processing	StatServer	

4. You have the option to add another target or click **OK** in the Targets dialog box.
5. You also have the option of using the threshold functions for conditional routing. You can use Threshold to define additional conditions the target must meet to be considered as valid routing target. Click the button under Threshold to open Expression Builder where you can create a threshold expression.
6. See Creating Threshold Expressions below.
7. Click OK when through in the Targets dialog box.

Timeout Property

If not specified, validation generates an error. Enter an integer to specify the time in seconds an interaction waits for an available target. If the timeout expires before one of the targets is available, the interaction is routed to the error port. This value can be specified as a variable (previously defined in the Entry object) or an integer. Upon entering any Target block, if a target was not selected and at least one statistic is not open and waiting time is 0, URS adjusts waiting time to the maximum specified in URS options reservation_pulling_time and treatment_delay_time values. This eliminates the possibility of not routing the first call after URS is started.

Type Property

Use to define the type of redirection processing that is to be done. For more information and individual values, see the *Orchestration Server Developer's Guide* on the Orchestration Server Wiki.

Use Access Code Property

Set to true to update the DN property of the resource reserved based on routing criteria to include the access code returned by URS. If true and the Route property is set to false, the resource object will still be modified even though the Target block will not route the call to the resource.

Use Access Code Setter Property

This property specifies the name of the function to use to update the resource with the returned access code. By default, this property points to the `updateDN(accesscode, dn)` function from `include/RouteFeature.js`. You can update the provided function so that all Target blocks that use this function benefit from the update. Or you can provide your own function to implement specific needs for a specific block

Virtual Queue Property

Use this property to define the enabled ACD virtual queue for the tenant. The virtual queue is not a physical queue but rather a logical queue to which interactions are queued if the specified targets are not available. Also see the Virtual Queue Selected property, which lets you select a variable. To define the virtual queue:

1. Opposite the Virtual Queue property, click under **Value** to display the ... button.
2. Click the ... button. The Virtual Queue dialog box opens.
3. Configuration Database are listed for selection. Select a queue.
4. Click **OK**.

Virtual Queue Selected Property

Select a variable containing a logical queue (Alias) to which interactions are queued if the specified targets are not available. Also see the Virtual Queue property below, which lets you specify the queue.

Creating Threshold Expressions

When using the Target block, Targets property, to define a threshold expression, there are two different cases:

- The threshold value contains an occurrence of sdata, acfgdata, lcfgdata: the threshold expression is not evaluated by the Orchestration platform. Instead, the whole expression is passed as a string to the underlying queue functional module which evaluates that expression. The underlying queue module is not aware of the variables defined in the SCXML application. If any variable is part of the expression, it does not get evaluated.

For example the threshold expression: `sdata('agtgrp1.GA', 'StatAgentsAvailable') > var0` is passed to the ORS as `threshold="'sdata(\'agtgrp1.GA\', \'StatAgentsAvailable\') > var0'"`

- The threshold value does not contain any occurrence of sdata, acfgdata, lcfgdata: the threshold expression can be evaluated by ORS. The result of the expression is passed to the underlying queue functional module. The expression may contain variables. For example the threshold expression: `var0 + from_sub` is passed to Orchestration Server as `threshold="var0 + from_sub"`

See Queue Interface Action Elements for more detail.

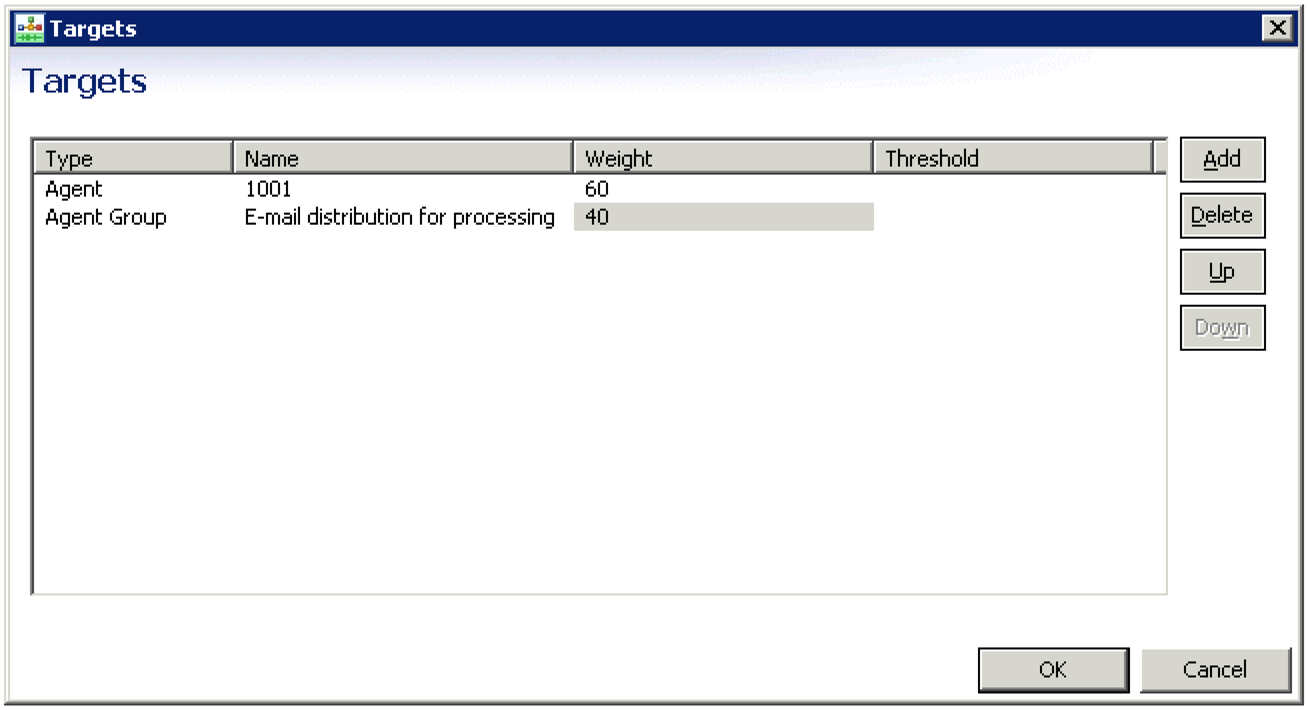
Percent and Conditional Routing

You can also use the Target Block for:

- Percentage routing or for
- Conditional routing using the threshold functions

Percentage Routing

When using the Target Block, If percentage is selected for the Statistics Order property, the Targets dialog box, which opens from the Targets property, displays a Weight column as shown below.

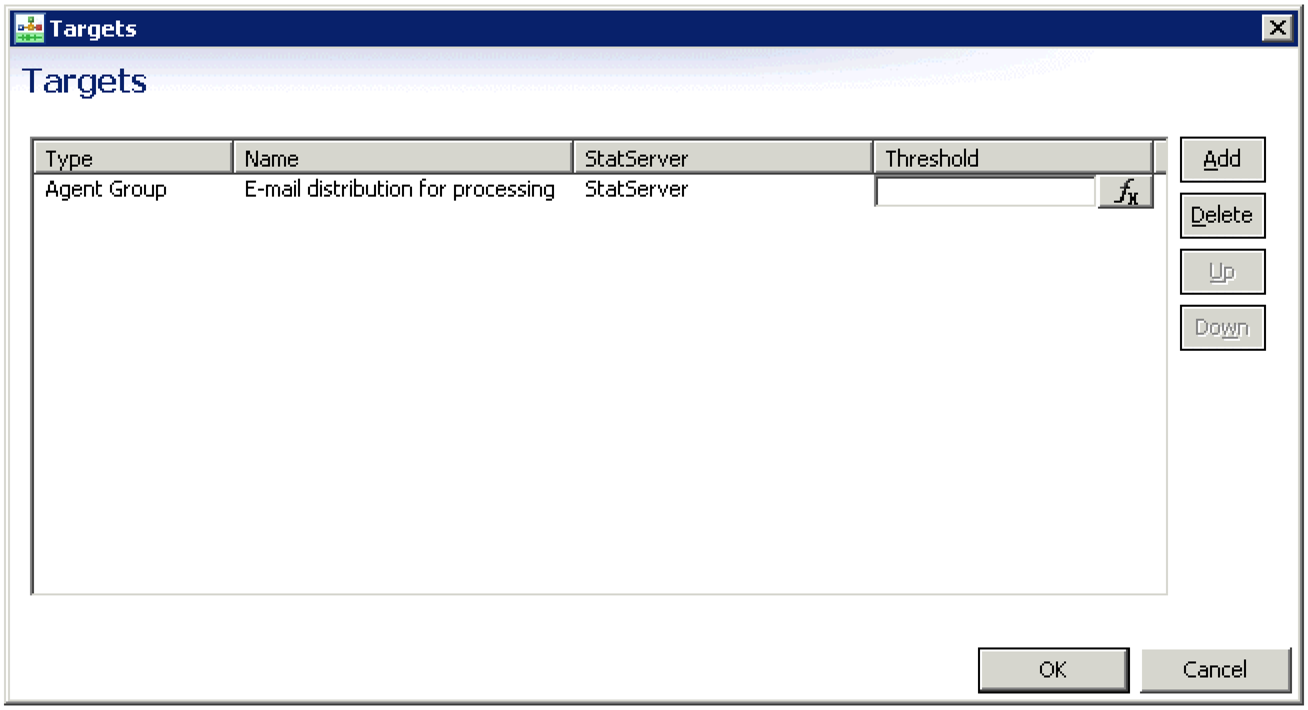


Type	Name	Weight	Threshold
Agent	1001	60	
Agent Group	E-mail distribution for processing	40	

Here you can enter numbers to specify percentage allocation for each target.

Conditional Routing

Universal Routing Server's threshold functions can be used in the Target block for conditional routing, such as "share agents by service level agreement routing" as described in the *Universal Routing 8.1 Routing Application Configuration Guide*. The Targets dialog box, which opens from the Targets property, displays a Threshold column. Click the button under Threshold to open Expression Builder where you can create a threshold expression.



Type	Name	StatServer	Threshold
Agent Group	E-mail distribution for processing	StatServer	

List Objects Manager

A List object contains strings of any nature (for example, DNIS or ANI strings), which can be used in strategies. The strings can be as simple as 800 numbers or as complex as routing conditions.

For example, you may use a List object to create lists of toll-free numbers. Rather than reference each individual 800 number in a strategy, you can logically group numbers together and name the group. Then, when you need to add new numbers or edit numbers, you do not need to edit the strategy, but only the List object properties.

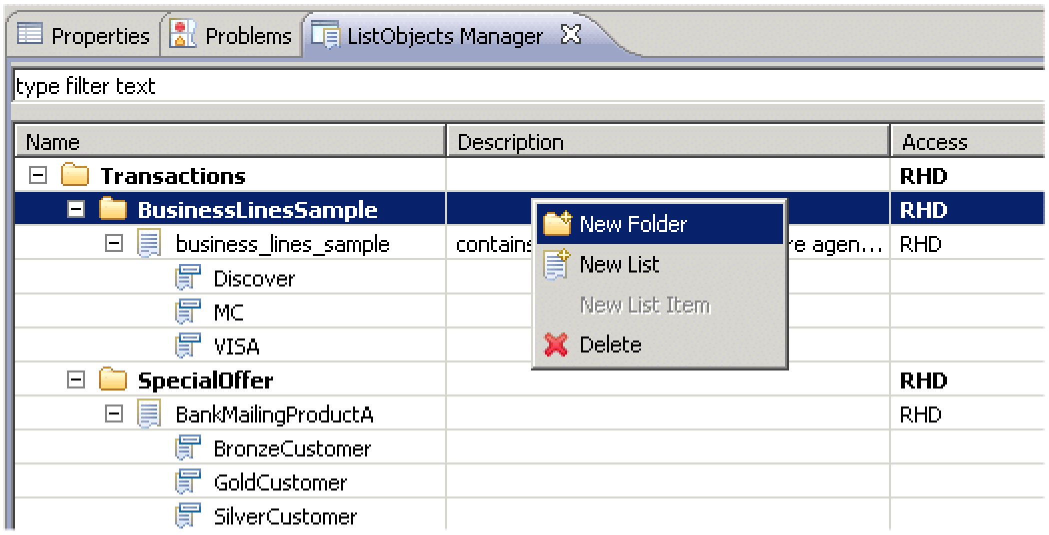
Providing key-value pairs for a List element enables you to store information in the Configuration Server Database and then retrieve it from the strategy.

You can specify a List Object variable when specifying targets in the Target selection block.

Creating List Objects

Composer supplies a List Objects Manager view to create List objects. To bring up the view and create a List object:

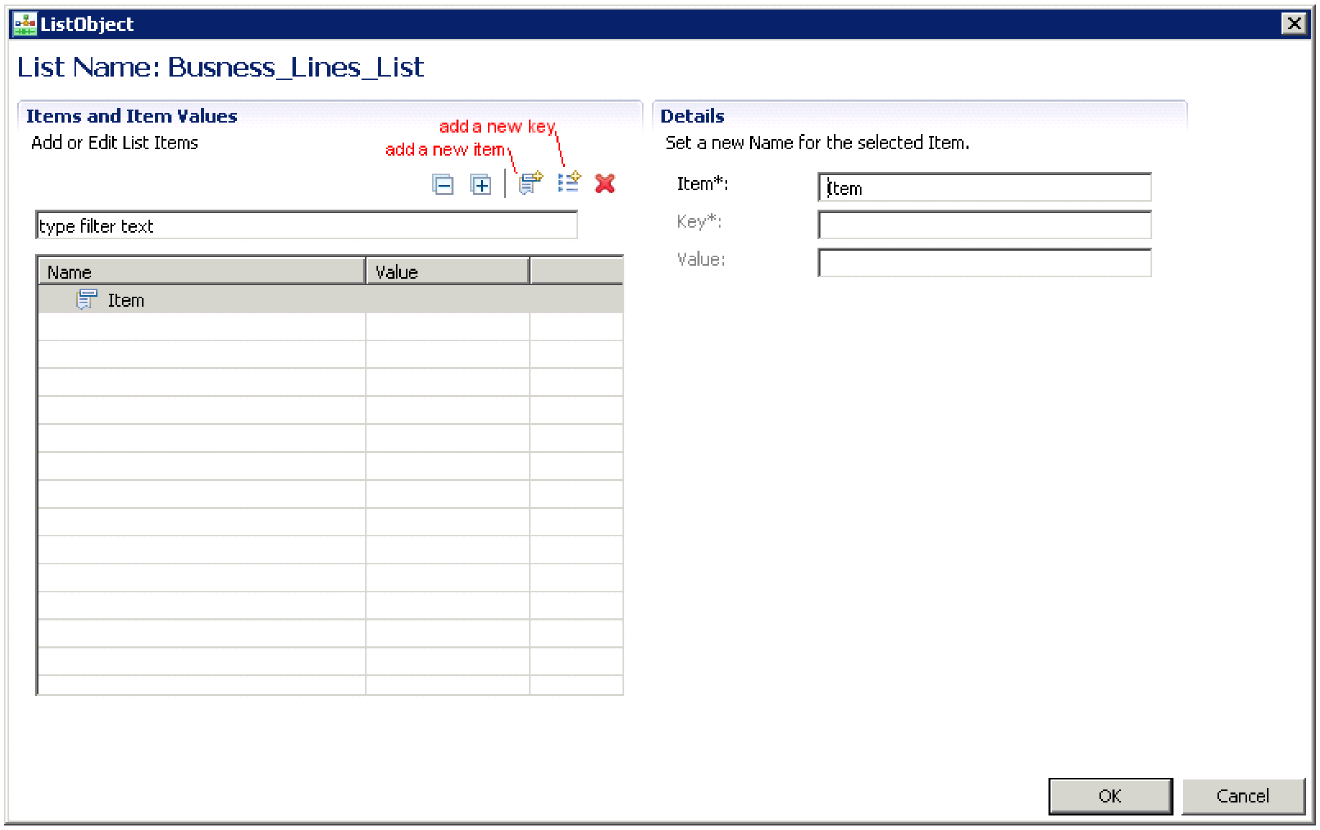
1. If you have not already done so, connect to Configuration Server.
2. From the Window menu, select **Show View** > **List Objects Manager**. You can customize the Show View menu to show List Objects Manager: **Window** > **Customize Perspective**.
3. Right-click the Transaction folder, select **New Folder**, and name the folder.



Name	Description	Access
Transactions		RHD
BusinessLinesSample		RHD
business_lines_sample	contains e agen…	RHD
Discover		
MC		
VISA		
SpecialOffer		RHD
BankMailingProductA		RHD
BronzeCustomer		
GoldCustomer		
SilverCustomer		

1. Right-click the new folder and select **New List**.
2. In the Create New List Object dialog box, name and describe the List object and click **OK**.

3. Right-click the name of the List object and select **New List Item**. The dialog box for defining a List item opens.


ListObject

List Name: Busness_Lines_List

Items and Item Values

Add or Edit List Items

type filter text

Name	Value
Item	

Details

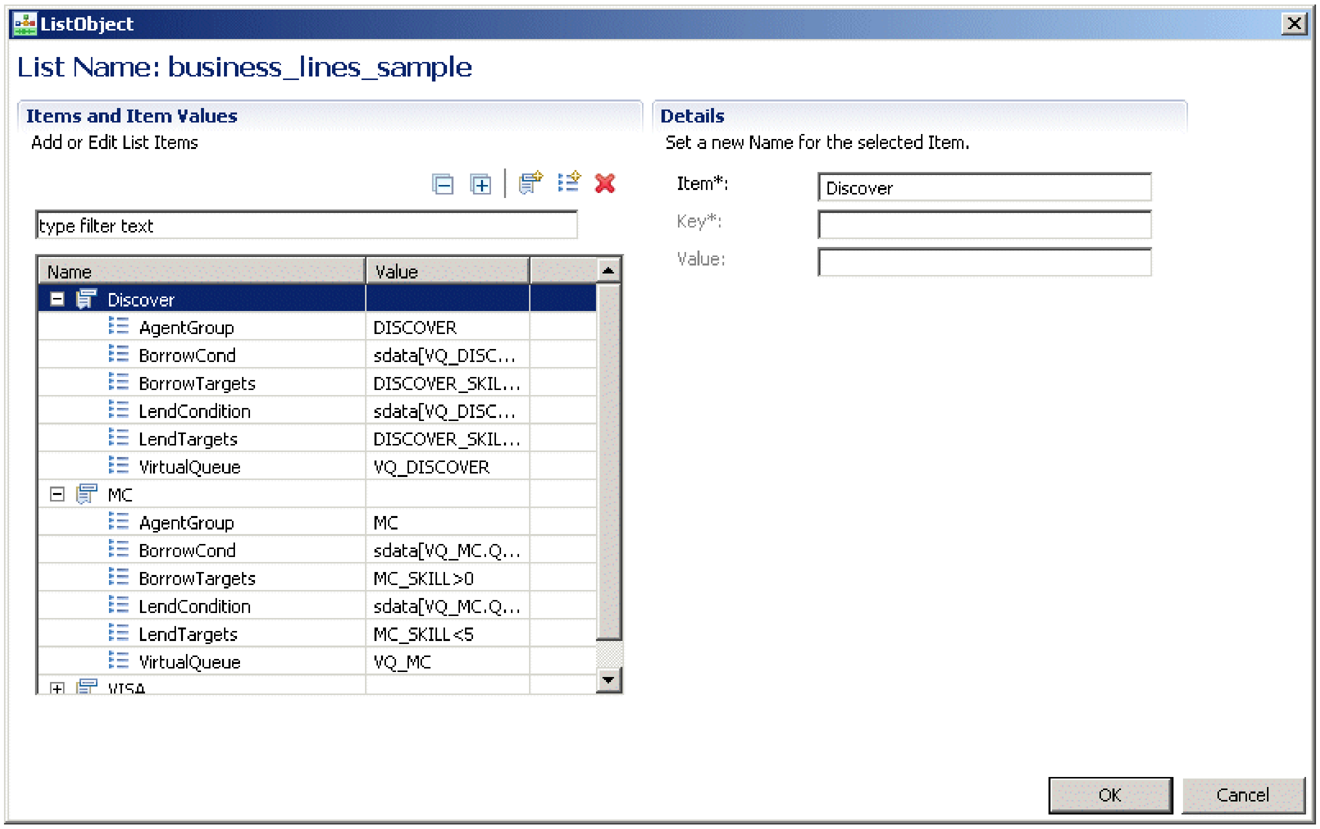
Set a new Name for the selected Item.

Item*:	Item
Key*:	
Value:	

1. On the toolbar, click the button to add a new item.
2. Under **Details**, enter a name in the Item field and click **OK**.
3. Continue adding rows in the List object in this fashion.
4. When through adding rows, in the List Object Managers tab, double-click a row.
5. When the row is selected in the List Object dialog box, click the button to add a new key.
6. Under **Details**, enter the **Key** and **Value** fields.
7. Click **OK**.
8. Continue adding key-values to rows in this fashion.

Sample

Besides individual items, parts of expressions (or an entire expression) can be stored outside of a strategy inside a List object. The sample below is a List object that contains routing information that URS can use to decide when to borrow and lend agents among business lines.



ListObject

List Name: business_lines_sample

Items and Item Values
Add or Edit List Items

type filter text

Name	Value
Discover	
AgentGroup	DISCOVER
BorrowCond	sdata[VQ_DISC...
BorrowTargets	DISCOVER_SKIL...
LendCondition	sdata[VQ_DISC...
LendTargets	DISCOVER_SKIL...
VirtualQueue	VQ_DISCOVER
MC	
AgentGroup	MC
BorrowCond	sdata[VQ_MC.Q...
BorrowTargets	MC_SKILL>0
LendCondition	sdata[VQ_MC.Q...
LendTargets	MC_SKILL<5
VirtualQueue	VQ_MC
VISA	

Details
Set a new Name for the selected Item.

Item*:	Discover
Key*:	
Value:	

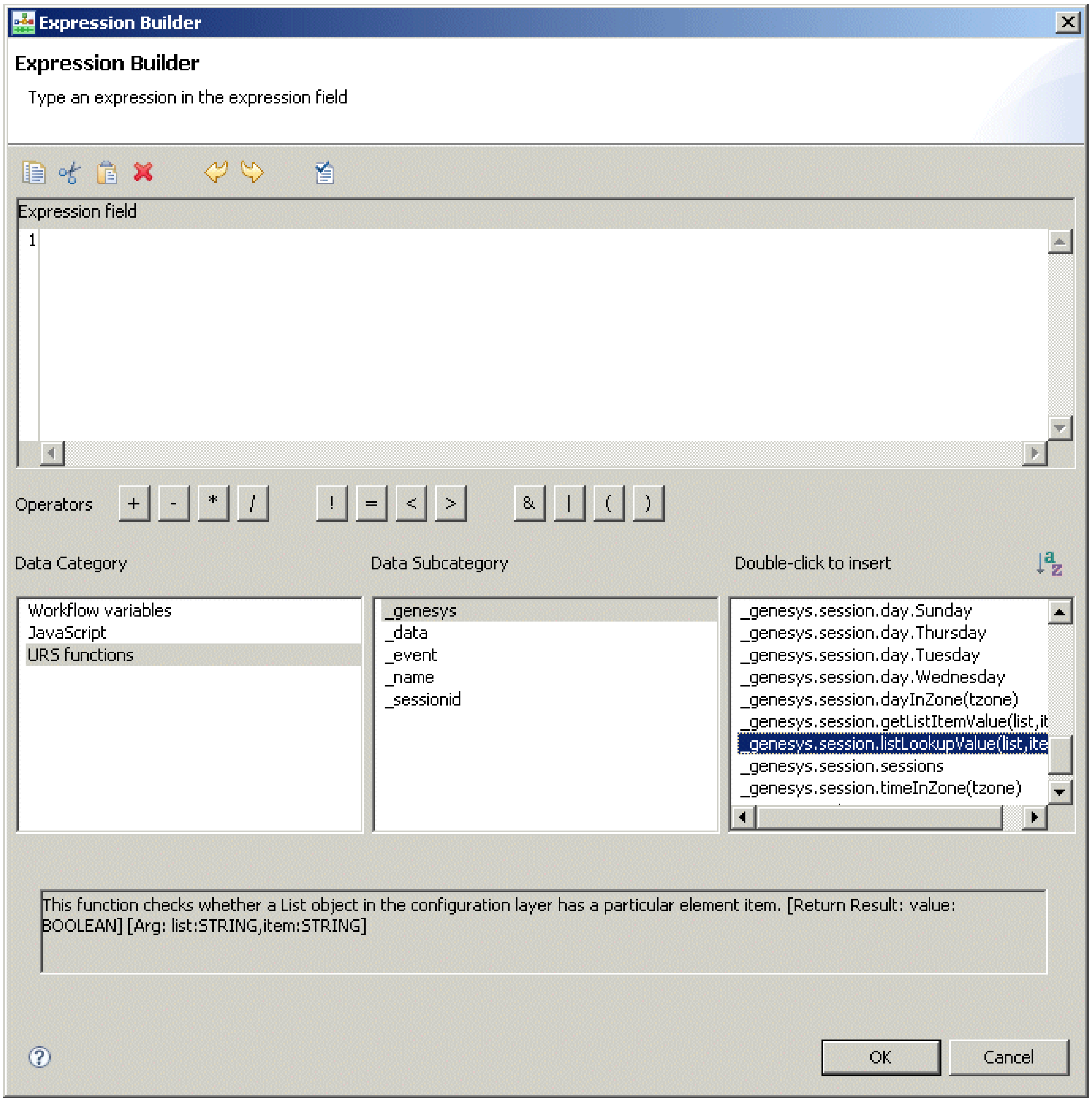
In this sample:

- An IVR has identified customers as wanting information on the MC, VISA, or DISCOVERY business line.
- The requested business line information has been passed to the calling strategy.
- The strategy has segmented interactions to take different paths based on the requested business line.
- If all agents serving the requested business line are busy, URS can use information in the List object to borrow agents from other business lines.

Expression Builder Functions for List Objects

In Expression Builder, two URS functions can be used to access List Objects:

- `_genesys.session.listLookupValue`
- `.genesys.session.getListItemValue`


Expression Builder
Expression Builder
Type an expression in the expression field
Expression field
Operators
+
-
*
/
!
=
<
>
&
|
(
)
Data Category
Data Subcategory
Double-click to insert
Workflow variables
JavaScript
URS functions
_genesys
_data
_event
_name
_sessionid
_genesys.session.day.Sunday
_genesys.session.day.Thursday
_genesys.session.day.Tuesday
_genesys.session.day.Wednesday
_genesys.session.dayInZone(tzone)
_genesys.session.getListItemValue(list,it
_genesys.session.listLookupValue(list,ite
_genesys.session.sessions
_genesys.session.timeInZone(tzone)
OK
Cancel

This function checks whether a List object in the configuration layer has a particular element item. [Return Result: value: BOOLEAN] [Arg: list:STRING,item:STRING]

Refer to *Universal Routing SCXML API Reference* help file for details on these functions (Help > Contents).

Statistics Manager and Builder

When using the Target object in a strategy, the Property Statistics property lets you route based on the value of a Property statistic. The statistic can be a Statistics URS Predefined statistic (as described in the *Universal Routing 8.1 Reference Manual*) or a statistic that you create yourself with Statistics Builder. Once you create a statistic, that statistic becomes available for selection in the Target object.

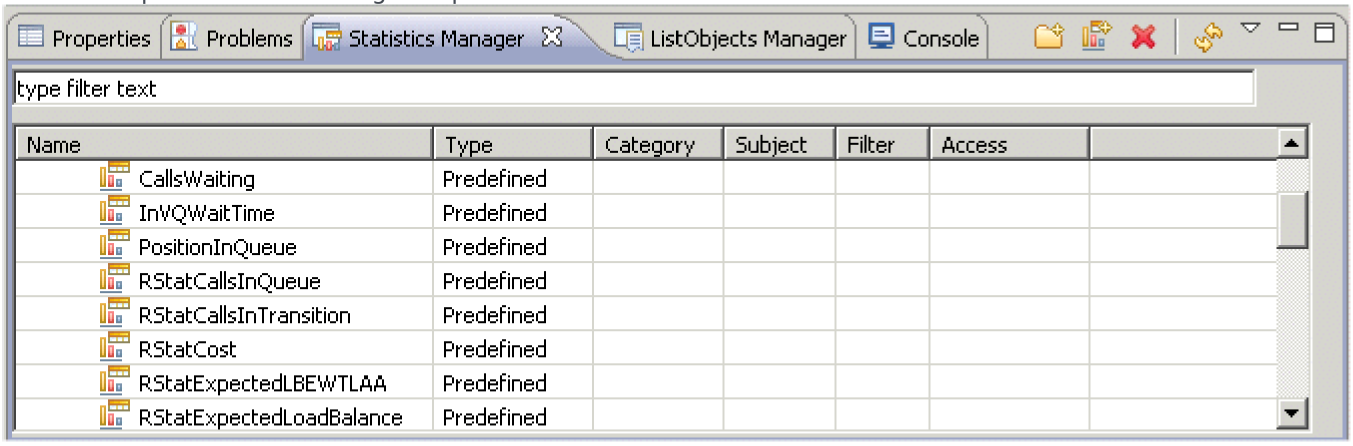
Using Statistics Manager

The Statistics Manager view lets you easily create, delete, and organize created statistics into folders.

You can connect to Configuration Server before or after opening the Statistics Manager view. When not connected, the view shows the message: Configuration Server is not connected. You must set the option to use URS Predefined statistics if you want to use the URS predefined statistics in a workflow and if those statistics do not already exist in Configuration Server.

To bring up the Statistics Manager view:

1. Click the Statistics Manager button on the main toolbar. Alternative methods: Select **Window** > **Show View** > **Statistics Manager.** You can customize the Show View menu to show Statistics Manager:**Window** > **Customize Perspective**. The Statistics Manager view appears along the bottom of the Composer window listing URS predefined statistics.



Name	Type	Category	Subject	Filter	Access
CallsWaiting	Predefined				
InVQWaitTime	Predefined				
PositionInQueue	Predefined				
RStatCallsInQueue	Predefined				
RStatCallsInTransition	Predefined				
RStatCost	Predefined				
RStatExpectedLBEWTLAA	Predefined				
RStatExpectedLoadBalance	Predefined				

StatisticsMgr.gif

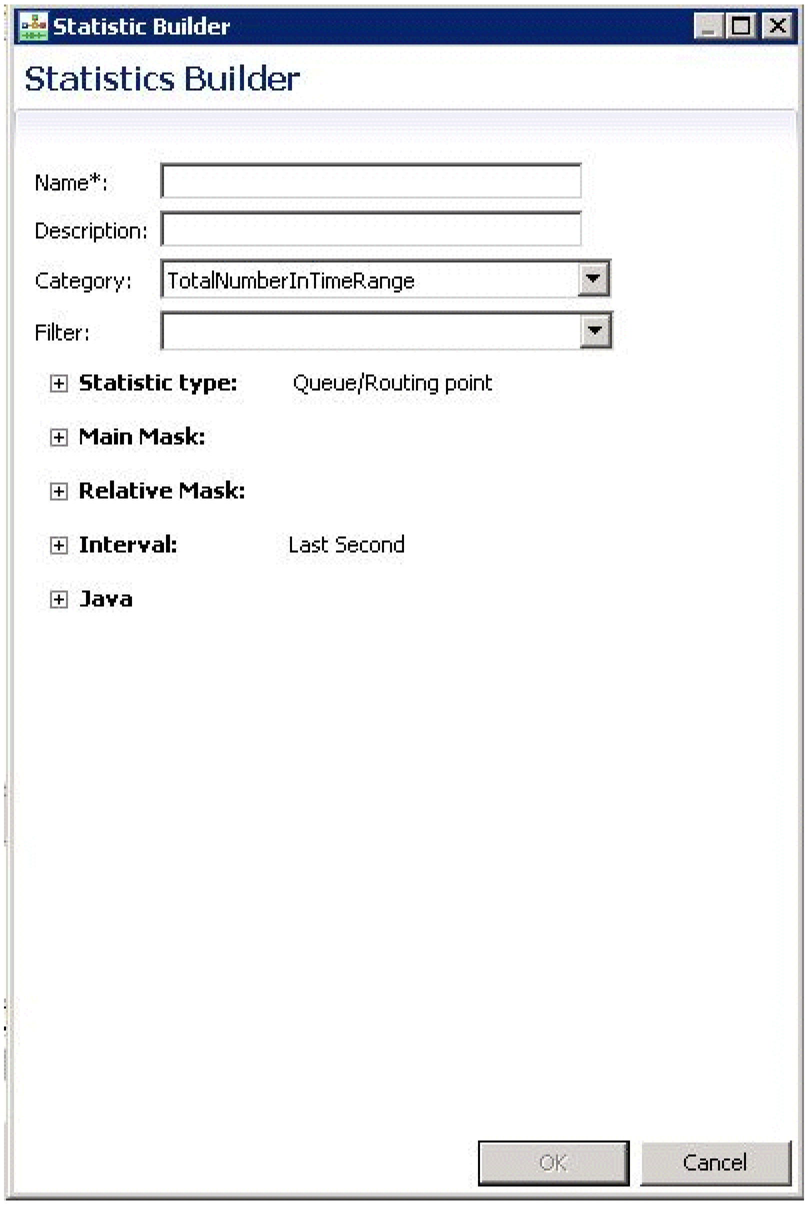
2. In the Statistics Manager view, you can do the following:

   - Double-click a statistic to edit with Statistics Builder.
   - Right-click a folder under **Name** and select **New Statistic**. Or with a folder selected, click the **Add New Statistics** button.

- Create a new folder by clicking the **Add New Folder** button.
- To delete a statistic that you have created, select the statistic and click the button to delete. Or right-click the statistic and select **Delete**. Note: You cannot delete any of the URS predefined statistics.

Using Statistics Builder

An example Statistics Builder dialog box is shown below.


Statistic Builder

Statistics Builder

Name*:

Description:

Category: TotalNumberInTimeRange

Filter:

Statistic type: Queue/Routing point

Main Mask:

Relative Mask:

Interval: Last Second

Java

To create a new statistic:

1. Enter a **Name** for the statistic.
2. Enter a **Description** for the statistic
3. Select the **Category** for the statistic, which tells Stat Server how to calculate the statistic. For information on statistical categories, see the “Statistical Categories” chapter of the *Framework Stat Server User’s Guide*.
4. Enter a **Filter** name (optional). Filters enable you to exclude interactions based on certain criteria specified in a logical condition. They enable you to restrict the Stat Server actions that are taken into account during the computation of aggregate values. In a filtered statistic, only those actions are considered that satisfy a filter condition on certain attributes of T-Events. For more information, see

Filters Section in the “Statistic Configuration Options” chapter of the *Framework Stat Server User’s Guide*. If you enter a Filter name that does not exist, the object will be created with the Statistic when you click OK.

5. Select the appropriate radio button to configure one of the following types of statistics:
    - **Queue/Routing Point**
    - **Agent/Group**
    - **Campaign**
6. Select from **Main Mask** and **Relative Mask**. A mask is a list of actions or statuses that apply to the statistic category. Depending on the type of statistic you select **(Queue/Routing Point**, **Agent/Group**, or **Campaign**), the Mask listing shows different actions or statuses. Where several masks are listed, you may select all of them. For lists of actions and statuses, see the chapters on “Stat Server Actions” and “Object Statuses” in the *Framework Stat Server User’s Guide*.
7. Enter **Interval: Last Second**. Refers to the time interval used for calculating historical aggregate values for statistics. Clients, such as CCPulse+, specify which defined time profile to use when requesting their statistics. Options are **Last Seconds**, **Last Calls**, **Growing Window**, and **Since Login**. For more information on intervals, see the Time Profiles section in the “Statistic Configuration Options” chapter of the *Framework Stat Server User’s Guide*. Note: In most routing decisions, the interval over which statistics are measured (if applicable) should have a fixed length. The Growing Window option is intended for exceptional cases and should be used infrequently.
8. **Java**. You have the option to enter:
    - **Extension**—Stat Server architecture includes support for pluggable statistical modules written in Java. This enables you to dynamically extend Stat Server functionality with new statistical types (residing in Stat Server’s Java Extensions [SSJE]) and to have Stat Server supply them to Genesys applications. It is through this functionality that Stat Server processes information from Interaction Server. For more information on this field, see the *Framework Stat Server User's Guide*.
    - **Subcategory**—The name of the Java subclass that implements statistic calculation.
9. Click **OK**.

Voice Treatment Blocks

Busy treatments can be played to callers when all the targets selected by URS are busy and the interaction is waiting for an available target. You can specify treatments using the blocks below or use the Target block Treatments property.

Connecting to Busy Treatment Port

The following treatment blocks may be connected to the Busy Treatments port of the Target block and used as busy treatments: Play Application, Play Sound, Play Message, User Input, Set Default Route, Pause, Create User Announcement, Delete User Announcement, IVR, and Disconnect. This will enable you to see all defined busy treatments in the diagram as well as use the Properties view to configure busy treatments. The table below summarizes the voice treatment blocks.

Cancel Call	Use this block to stop a currently running dialog.
Create User Announcement	Use this block to record a caller announcement.
Delete User Announcement	Use this block delete an announcement created by a caller using the Create User Announcement block.
IVR	Use to invoke an interactive voice response (IVR) unit and connect the interaction to the IVR.
Pause	Use to suspend treatment processing for a specified duration.
Play Application	Use to execute an application (such as a Composer voice application) or a script on a device, such as an IVR.
Play Sound	Use to play audio resources of the following type: • Music • BusyTone • FastBusyTone • RingBack • RecordedAnnouncement (on Stream Manager) • Silence
Play Message	Use to invoke/play audio or text-to-speech Announcement treatments.
User Input	Use to play a text-to-speech announcement, collect digits, and (optionally) verify the input digits.

Set Default Route	Use to set/change the default route number at any time.

Composer Equivalent to IRD Treatment

Composer includes treatment functionality that was previously provided through Genesys Interaction Routing Designer (IRD). The information below is provided for existing Genesys customers transitioning to Composer, who are familiar with using IRD's Treatment objects. The **Composer Busy Treatment?** column specifies whether the treatment is supported as a busy treatment. If not, it is only supported as a mandatory treatment.

- Mandatory treatments are those to which an interaction will be subject whether there are any available targets or not.
- Busy treatments are applied when all the targets selected are busy and the interaction is waiting for an available target.

Composer Equivalent	Composer Busy Treatment?	IRD Treatment	SCXML Tags	Description
Create User Announcement	no	Record User Announcement	<dialog:createann>	Records an announcement from a user and associate it with a specific user.
Delete User Announcement	no	Delete User Announcement	<dialog:deleteann>	Deletes an announcement created by a caller using the Create User Announcement treatment.
Disconnect	no	Cancel Call	<dialog:stop>	Disconnect the caller and end the call.
IVR	yes	IVR	<dialog:remote>	Invokes an interactive voice response (IVR) unit and connect the interaction to the IVR.
Pause	yes	Pause	<send> (mandatory) <pause> (busy)	Suspends treatment processing for a specified duration.
Play Application	yes	Play application	dialog:start	Executes an application or script on the IP device. It is possible to pass parameters to the application and

				get return values.
Play Message	yes	Play announcement	dialog:play	Plays an announcement for the caller.
Play Message	yes	Text to speech	dialog:play <prompts type=tts>	Generates speech from text. No recorded announcements can be used.
Play Sound	yes	Busy	dialog:playsound type="busy"	Connects the interaction to the source of a busy tone. The caller hears a busy signal.
Play Sound	yes	Fast busy	dialog:playsound type="fastbusy"	Connects the interaction to the source of a different busy tone. The caller hears a fast busy signal. This treatment is supported on a limited number of switches.
Play Sound	yes	Music	dialog:playsound type="music"	Connects the interaction to a music source.
Play Sound	yes	Ringback	dialog:playsound type="ringback"	Connects the interaction to a ringback tone source.
Play Sound	yes	Silence	dialog:playsound type="silence"	Specifies an interval without sound.
Play Sound	yes	RAN	dialog:playsound type="ran"	Similar to the Music object. The source must be a RAN port on the switch. This treatment is supported by a limited number of switches.
Set Default Route	no	Set default destination	dialog: setdialogdefaultdest	Sets or changes the default destination.
User Input	yes	Play announcement and collect digits	dialog:playandcollect	Plays an announcement for the caller and requests that the

				caller provide information by inputting digits.
User Input	yes	Text to speech and collect digits	dialog:playandcollect	Generates speech from text and then requests input from the caller in the form of digits.
User Input	yes	Collect digits	dialog:collect	Collects digits that a caller enters. For example, you might collect an account number. Don’t specify prompts in UserInput block. It will generate code for <dialog:collect>/
Digit verification in User Input	yes	Verify digits	<playandverify>	The IRD Verify Digits object prompts a caller to enter digits that will be compared to a desired response.

Cancel Call Block

Use this block to stop a currently running dialog. The Cancel Call block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Extensions Property

Select the variable to retrieve extensions data in `event dialog.stop.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Documentation Wiki.

Request ID Property

Select the variable whose value is the ID of the previously started dialog-related action.

Create User Announcement Block

Use this block to record a caller announcement. The treatment device returns an announcement ID (User ID property) for the newly created announcement, which the application can use later to trigger playback of the announcement in other treatment blocks that support playing prompts. The Create User Announcement block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Extensions Property

Select the variable to retrieve extensions data in `event dialog.createann.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Documentation Wiki.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Announcement ID Property

Select the variable (Project or workflow) that contains the identifier for the created announcement. This application variable may be used later in other blocks to work with the caller announcement created in this block. The Delete User Announcement block will accept this variable if you wish to delete this announcement.

Prompts Property

This property lets you define of a series of elements (prompts), which are pieced together. Each prompt can be described as interruptible or non-interruptible.

1. Click the Prompts row in the block's property table.
2. Click the ⋯ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under Type, select one of the following:
    - **Announcement--Plays** an announcement to the calling party. In this case, the Value field contains a number of elements (from 1 to 10. Each element is named with a number ranging from 1 to 10 and contains a number of entries describing announcement elements. Announcement prompts do not have a user association.
    - **FormattedDigits**--Used to collect digits from the caller.
    - **Text**--Essentially the same as Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when Announcement is not supported**.**
    - **User Annoucement**-- Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.
5. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
6. Under **Value**, enter the prompt parameters.
7. Click **Add** again to enter another prompt, or click **OK** to finish.

Abort Digits Property

Select the variable to contain up to 2 digits that the caller can use to abort the recording process. If aborted, an error event is generated.

Reset Digits Property

Select the variable to contain a sequence of up to 2 digits that the caller can use to restart the recording process and discard any recording made up to that point in this bock will be discarded. This is not an error condition.

Start Timeout Property

Select the variable to contain the number of seconds that the routing platform should wait for the caller to start recording the announcement.

Termination Digits Property

Select the variable to contain a sequence of up to 2 digits that the caller can use to indicate the end of the recording process. This indicates a success case.

Total Timeout Property

Select the variable to contain the number of seconds for which the routing platform should wait for the caller to complete recording.

Request ID Property

Select the variable to hold the ID associated with the treatment request from the orchestration application or the resource.

Wait For Treatment End Property

Select true or false.

- If true, the transition to the next block occurs when the treatment is finished (or if a timeout occurs).
- If set to false, processing goes to the next block once the treatment is successfully started instead of waiting for the treatment to complete. The Request ID variable holds the ID of the treatment.

User ID Property

Select or enter the variable to contain the user identifier to be associated with this recording. Can be used to trigger playback of the recording in other treatment blocks that support playing prompts.

Delete User Announcement Block

Use this block to delete an announcement created by a caller using the Create User Announcement block. The Delete User Announcement block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Extensions Property

Select the variable to retrieve extensions data in `event dialog.deleteann.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Documentation Wiki.

Announcement ID Property

Enter or select the variable to contain the identifier of the announcement to delete.

User ID Property

Select or enter a variable to contain the user identifier associated with the recording to be deleted.

IVR Block

Use this treatment block to invoke an interactive voice response (IVR) unit and connect the interaction to the IVR. Specify the IVR in a format similar to that of a target because Universal Routing Server (URS) transfers the interaction in the same way as it would when routing to a target. The IVR produces a set of DNs from the source (corresponding to a target ID), location, type, and Stat Server information. If any of these DNs are available, it sends the interaction to one of them. Note: URS does not verify that the DN to which it sends the interaction is an IVR. It may be useful to send the interaction to a queue, which distributes it to the actual IVR. When specifying an IVR target:

- The IVR itself is often configured in Configuration Layer as a DN of type Voice Treatment Port, where Voice Treatment Option scripts can run.
- The most frequent configuration for using IVRs in a workflow involves creating a Place in Configuration Layer and creating a shortcut from the place to the IVR DN. Then IVRs can be used as treatments by specifying the Place or a Place Group that corresponds to an IVR.
- Another way to specify the IVR is to set the Compatibility Mode property to "true" and then manually enter the DN as a source and the name of the switch as the location (<DN>@<name of switch>) as the Remote Resource.

All voice treatments except IVR work at the switch or the intelligent peripheral. The IVR treatment is different from the other treatments because URS routes the interaction to an IVR, just as it does for a target. Therefore, the format used to specify the IVR treatment is similar to that for a target. The IVR block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event using the **IPD Wait For Event property** to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Application Property

Select the variable to contain the identifier of the application that will be started by the specified remote resource as part of this treatment. Used only if the Compatibility Mode property = true.

Extensions Property

Select the variable to retrieve extensions data in `event dialog.remote.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Documentation Wiki.

Compatibility Mode Property

Select true or false to control if IVR information is specified in a format compatible with how targets are specified in URS as described in the *Universal Routing 8.1 Reference Manual*. See the Remote Resource property for more information.

Failover Resource Property

Select the variable to contain the address of the resource to which this interaction will be routed in case there is an error in routing to the resource specified in the Remote Resource property. This resource would be used, for example, if the connection to T-Server is lost. The format of this value depends on the value of the Compatibility Mode property.

- If Compatibility Mode = true, the value must in the targeted format: 2323@www.genesys.com\server1.AG
- If Compatibility Mode = false, the value can be a string for the target resource in the format `TargetName@StatServerName.TargetType` as described in *Universal Routing 8.1 Reference Manual*, Chapter 2, Target Properties.

Remote Resource Property

Select the variable that contains the address of the remote resource that will be used to provide the treatment. The format of this value depends on the value of the Compatibility Mode property.

- If Compatibility Mode = true, the value must in the format: `DN.SWITCH` (like `2323@switch_name`) or
- If Compatibility Mode = false, the value can be a string for the target resource in the format: targetname@statservername.targettype (like PlaceGroup@Ststserver.GP)as described in *Universal Routing 8.1 Reference Manual,* Chapter 2, Target Properties.

Treatment Duration Property

Enter the maximum duration in seconds for which this treatment can be executed. After this duration, the treatment is considered finished successfully by the platform. Limitation: It is not possible for the application to determine if the IVR treatment completed successfully or it was cut short once the treatment duration expired.

Pause Block

Use this block to suspend treatment processing for a specified duration.

- If a treatment block other than the IVR block precedes the Pause block, that treatment continues for the duration of the Pause treatment.
- In the case of a preceding IVR treatment, the Pause treatment starts only after the interaction has been returned to the routing point. Then the caller hears nothing for the specified time interval.

The Pause block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Duration Property

Select the variable to contain the time in seconds for which treatment processing should be suspended.

Unit Property

click the down arrow and select either **second** or **millisecond**.

Play Application Block

Use to execute an application (such as a Composer voice application) or a script on a device, such as an IVR.

- For URS-controlled applications where the call lands on the strategy first and then a VXML application is called using the Play Application treatment, GVP must be configured as DN type VoiceOverIP Service. Find detailed configuration instructions in Voice Platform Solution Integration Guide.
- Debugging a Play Application block will not step into the associated Callflow diagram and will not launch a GVP debugging sessions. Instead debugging will continue on to the block after the Play Application block.

Also see Passing Parameters. The Play Application block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Extensions Property

Select the variable to retrieve extensions data in `event dialog.start.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Wiki.

Language Property

To set the active language:

1. Select the Language row in the block's property table.
2. Click under Value to display a down arrow.
3. Select one of the following languages:
    - **English (US)**
    - **Spanish**
    - **Mandarin**
    - **Cantonese**
    - **Vietnamese**
    - **French**
    - **French (Canada)**
    - **German**
    - **Italian**
    - **Japanese**
    - **Korean**
    - **Russian**

Parameters Property

If the Type property is not URL, use the Parameters property to specify any treatment parameters:

1. Click the **Parameters** row in the block's property table.
2. Click the button to open the Input and Output Parameter Sync dialog box.
3. Click **Add** to add a treatment parameter.
4. Under **Parameter Name**, accept the default name or change it.
5. From the **Parameter Type** drop-down list, select input.

Note: If the Type property is set to **ProjectFile**, the dialog box shows only input variables as Composer validates against the specified callflow.

6. Under **Value**, select from among the variables shown.
7. Under **Definition**, type a description for this parameter.
8. Click **Add** again to enter another parameter, or click **OK** to finish.

To delete a parameter, select an entry from the list and click **Delete**. Also see the Passing Parameters section below. **Next Generation Interpreter (NGI) Change** In 8.1 and prior releases, a parameter in the SIP Request URI was exposed as a VoiceXML session variable as follows:

- Request URI: `sip:dialog@host;ParameterA=ValueA`
- Session Variable: `session.connection.protocol.sip.requesturi['ParameterA']` (Note: The case is preserved)

Starting from 8.1.1, the parameter name will be converted to lowercase in the session variable’s array index. Example:

- Request URI: `sip:dialog@host;ParameterA=ValueA`
- Session Variable: `session.connection.protocol.sip.requesturi['parametera']` (Note: The index has to be lowercase.)

Any parameters set in the Play Application block will be accessed in NGI with the parameter name in all lowercase.

Resource Property

Use to specify the treatment resource.

1. Click under **Value**.
2. The Resource property works with the Type property.
    - If **URL** or **Id** is the selected Type, clicking under Value displays a down arrow. Click the down arrow and select a variable. For GVP applications, the variable must contain the full URI of the VXML page.
    - If **ProjectFile** is the selected Type, clicking under Value displays the ... button. Click to open a dialog box where you can select the Project file/callflow for the treatment application. Integration with voice callflows is provided so it is possible to select a callflow diagram file as the project file. In this case, Composer will automatically substitute the actual URL of the page at the time of Code Generation.

Type Property

Identifies the type of Resource.

1. Click under **Value** to display a down arrow.
2. Select one of the following types:

    - **URL**--Indicates that the Resource is a .vxml file and not a callflow. Composer currently does not parse this .vxml file so the Input and Output Parameter Sync dialog box described under the Parameters property does not open.
    - **ProjectFile**--Indicates that the Resource is a callflow.
    - **Id**--Indicates the ScriptID parameter as described in the *Genesys Voice Platform 8.1 Reference Help* (see Sample VoiceXML Applications > CTI Interactions > Treatments (Post Initiate Transfer) > Play Application topic). The Play Application treatment is invoked without APP_URL specified in the strategy.

Use User Data Property

When set to true, Composer will automatically update the interaction’s User Data with the input/inout parameters specified in the Parameters property. Similarly, Composer will automatically read the interaction's User Data and update corresponding variables for every inout/output parameter specified in the Parameters property. Uses to avoid the procedure described in the Passing Parameters property description.

Request ID Property

Select the variable to hold the ID associated with the treatment request from the orchestration application or the resource.

Wait For Treatment End Property

Select **true** or **false**.

- If true, the transition to the next block occurs when the treatment is finished (or if a timeout occurs).
- If set to false, processing goes to the next block once the treatment is successfully started instead of waiting for the treatment to complete. The Request ID variable holds the ID of the treatment.

Passing Parameters

To pass parameters from a workflow to a callflow that is being invoked with the Play Application block, follow these steps: Note: Step 2. below is required only for SIPS and is not mandatory for CTIC flow cases.

1. Create a variable of type Input in the Variables property of the callflow’s Entry block. The variable name must be all lowercase.

2. At a point before the Play Application block in the workflow, add an ECMAScript block with the following script contents:

var input = new Object(); input.xyz = _data.VariableToPass; _genesys.ixn.setuData(input); xyz should match the variable name that you created in step 1. You can replace _data.VariableToPass with any variable, or a literal value, such as a string or number.

3. In the callflow, access the value using the variable that was created in step 1. Its value will be set automatically to the value specified in the ECMAScript block at the beginning of the callflow. Its value will be set at the beginning of the callflow to the value specified in the ECMAScript block in the workflow.

To pass parameters from a callflow (invoked from a Play Application block) back to the workflow, follow these steps:

1. Create a variable of type User in the Variables property of the callflow's Entry block. The variable name must be all lowercase.
2. In the callflow, use this variable to store any value that you want to pass back to the workflow.
3. In the callflow's Exit block, specify the variable as a return value.
4. In the workflow, after the Play Application block, the value will be available in the workflow variable _genesys.ixn.interactions[ixnid].udata.xyz, where xyz matches the name of the variable created in step 1.

Single Session Treatments

When using the Play Application, Play Sound (Music and ARM Types) Exit, and Disconnect blocks, voice applications can now optionally use a single VXML session on Media Control Platform to play/run multiple treatments instead of using one session per treatment. This enables DTMF buffering between multiple MSML treatments. For more information, see Single Session Treatments.

Play Sound Block

Use to play audio resources of the following type:

- Music
- BusyTone
- FastBusyTone
- RingBack
- RecordedAnnouncement (on Stream Manager)
- Silence

The Play Sound block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Find this property's details under Common Properties.

Extensions Property

Select the variable to retrieve extensions data in `event dialog.playsound.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Wiki.

Duration Property

The time, in seconds, that the treatment is applied.

Resource Property

Specify the location of the sound resource. Testing will be done against Genesys Stream Manager, a media server that generates and processes media streams in Real-time Transport Protocol (RTP) format. For more information on Stream Manager, start with the *Framework 7.6 Stream Manager Deployment Guide*. You will need to configure special DNs for these treatments in Configuration Server.

Sound Type Property

Identifies the type of sound.

1. Click under **Value** to display a down arrow.
2. Select one of the following types:
    - **Music**
    - **BusyTone**
    - **FastBusyTone**
    - **Ringback**
    - **Recorded Annoucement**
    - **Silence**

Single Session Treatments

When using the Play Application, Play Sound (Music and ARM Types) Exit, and Disconnect blocks, voice applications can now optionally use a single VXML session on Media Control Platform to play/run multiple treatments instead of using one session per treatment. This enables DTMF buffering between multiple MSML treatments. For more information, see Single Session Treatments.

Play Message Block

Use to invoke/play audio or text-to-speech Announcement treatments. As described in the *Genesys Voice Platform 8.1 Deployment Guide*. GVP supports automatic speech recognition (ASR) and speech synthesis (text-to-speech [TTS]) as part of a VoiceXML dialog, through supported third-party ASR and TTS engines that use open standards. The Play Message block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.   Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

- Not use blocks that require an Interaction ID
- And/or set the Interaction ID property to a meaningful value
- And/or assign a meaningful value to the InteractionId system variable

Extensions Property

Select the variable to retrieve extensions data in event `dialog.play.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Wiki.

Language Property

To set the active language:

1. Select the Language row in the block's property table.
2. Click under Value to display a down arrow.
3. Select one of the following languages:
    - English (US)
    - Spanish
    - Mandarin
    - Cantonese
    - Vietnamese
    - French
    - French (Canada)
    - German
    - Italian
    - Japanese
    - Korean
    - Russian

Prompts Property

This property lets you define of a series of elements (prompts), which are pieced together. Each prompt can be described as interruptible or non-interruptible.

1. Click the Prompts row in the block's property table.
2. Click the ▣ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under **Type**, select one of the following:
    - **Announcement**--Plays an announcement to the calling party. In this case, the Value field contains a number of elements (from 1 to 10). Each element is named with a number ranging from 1 to 10 and contains a number of entries describing announcement elements.
    - **FormattedDigits**--Used to collect digits from the caller.
    - **Text**--Essentially the same as Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when Announcement is not supported.
    - **User Annoucement**--Announcements with a user association previously created with the

Create User Announcement block can be played via this type of prompt.

5. Under **Interruptible**, select **true** or **false** to indicate if the caller can interrupt the message .
6. Under **Value**, select the variable containing the prompt parameters.
7. Click **Add** again to enter another prompt, or click **OK** to finish.

Type of Prompts Property

Click the down arrow and select Announcement or Text-To-Speech.

Request ID Property

Select the variable to hold the ID associated with the treatment request from the orchestration application or the resource.

Wait For Treatment End Property

Select **true** or **false**.

- If true, the transition to the next block occurs when the treatment is finished (or if a timeout occurs).
- If set to false, processing goes to the next block once the treatment is successfully started instead of waiting for the treatment to complete. The Request ID variable holds the ID of the treatment.

Set Default Route

Use this block to set/change the default route number at any time. Can override the default destination set by the Default Route block. Once set, this will be applicable for the entire duration of the strategy unless overridden by another Set Default Route block in the workflow execution.

The Set Default Route block has the following properties:

Name Property

Find this property's details under Property Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Property Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Destination Property

Enter the DN number for the default destination.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied.

Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki

Extensions Property

Select the variable to retrieve extensions data in `dialog. setdialogdefaultdest.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server Wiki.

User Input Block

Use to play a text-to-speech announcement, collect digits, and (optionally) verify the input digits. The User Input block has the following properties.

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Device ID Property

If specified, ORS will play treatments itself; otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Users can enter a value or select any runtime variable from the dropdown.

Hints Property

This property is for future use by Orchestration Server. Its use will be described in various action elements reference in the Orchestration Server wiki.

Abort Digits Property

Select the variable to contain a sequence of up to two keys that the caller can enter to abort the recording process. The IP considers this as a failed recording attempt.

Backspace Digits Property

Select the variable to contain a sequence of up to two keys causing the previous keystroke to be discarded.

Ignore Digits Property

Select the variable to contain a sequence of up to two keys to be treated as though the keys have not been pressed.

Reset Digits Property

Select the variable to contain a sequence of up to two keys causing all the previous keystrokes to be discarded. The digit collection resumes.

Termination Digits Property

Select the variable to contain a sequence of up to two keys causing all the digits, not including the TERM_DIGITS, to be returned to the service logic as collected digits.

Language Property

To set the active language:

1. Select the Language row in the block's property table.
2. Click under Value to display a down arrow.
3. Select one of the following languages:

    - **English (US)**
    - **Spanish**
    - **Mandarin**
    - **Cantonese**
    - **Vietnamese**
    - **French**
    - **French (Canada)**
    - **German**
    - **Italian**
    - **Japanese**
    - **Korean**
    - **Russian**

Prompts Property

This property lets you define of a series of elements (prompts), which are pieced together. Each prompt can be described as interruptible or non-interruptible.

1. Click the **Prompts** row in the block's property table.
2. Click the ⋯ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under **Type**, select one of the following:

    - **Announcement**--Plays an announcement to the calling party. In this case, the Value field contains a number of elements (from 1 to 10. Each element is named with a number ranging from 1 to 10 and contains a number of entries describing announcement elements.

- **FormattedDigits**--Used to collect digits from the caller.
- **Text**--Essentially the same as Recorded Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when RecordedAnnouncement is not supported.
- **User Announcement**--Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.

5. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
6. Under **Value**, enter the prompt parameters.
7. Click **Add** again to enter another prompt, or click OK to finish.

Timeout Prompts Property

This property defines the list of prompts to be played to the caller if a timeout occurs while waiting for input.

1. Click the button to open the Timeout Prompts dialog box.
2. Click **Add** to add a prompt.
3. Under **Type**, select one of the following:
    - **Announcement**--Plays an announcement to the calling party. In this case, the Value field contains a number of elements (from 1 to 10. Each element is named with a number ranging from 1 to 10 and contains a number of entries describing announcement elements.
    - **FormattedDigits**--Used to collect digits from the caller.
    - **Text**--Essentially the same as Recorded Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when RecordedAnnouncement is not supported.
    - **User Announcement**--Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.
4. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
5. Under **Value**, enter the prompt parameters.
6. Click **Add** again to enter another prompt, or click OK to finish.

Type of Prompts Property

Select one of the following:

- **Announcement**
- **Text-to-Speech**

Clear Input Property

Use this property to Indicate whether any information that has been input should be cleared before digit collection starts. Select **true** or **false**.

Digit Timeout Property

Select the variable to contain the number of seconds the IP should wait between DTMF digits.

Number of Digits Property

Select the variable to contain the number of digits to be collected. The maximum number of digits that can be collected is 31. The maximum number of digits can be equal to 0. In this case, no time is spent waiting for the caller to input digits, and a response is returned indicating 0 digits collected. The standard does not specify whether the response should contain a success or a failure indication, so expect an undefined behavior.

Start Timeout Property

Select the variable to contain the number of seconds the IP should wait for the caller to begin DTMF input.

Total Timeout Property

Select the variable to contain the total number of seconds the IP should wait for the caller to provide the requested DTMF input.

Verify Input Property

This property determines if the User Input block should also verify the collected input against a set of specified digits. If set to true, digits verification will be enabled and will allow for multiple attempts.

- If set to **true**, will generate <dialog:playandverify>.
- If set to **false**, block will generate <dialog:playandcollect>.

Collected Digits Variable Property

Select the variable to contain the collected digits.

Extensions Property

Select the variable to retrieve extensions data in `dialog.collect.done, dialog.playandcollect.done, dialog.playandverify.done` as described in the *Orchestration Developers Guide*, Orchestration Extensions, Dialog Log Interface section of the Orchestration Server wiki.

Request ID Property

Select the variable to hold the ID associated with the treatment request from the orchestration application or the resource.

Wait For Treatment End Property

Select **true** or **false**.

- If true, the transition to the next block occurs when the treatment is finished (or if a timeout occurs).
- If set to false, processing goes to the next block once the treatment is successfully started instead of waiting for the treatment to complete. The Request ID variable holds the ID of the treatment.

Failure Prompts Property

This property defines the prompts to be played to the caller if input verification against the specified digits fails.

1. Click the **Failure Prompts** row in the block's property table.
2. Click the ⋯ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under **Type**, select one of the following:

    - **Announcement**--Plays an announcement to the calling party. In this case, use the Value field to specify the prompt elements (can be up to 10). Each element is named with a number ranging from 1 to 10 and contains a number of entries describing the announcement elements. Announcement prompts do not have a user association.

- **FormattedDigits**--Used to collect digits from the caller.
- **Text**--Essentially the same as Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when Announcement is not supported.
- **User Announcement**--Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.

5. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
6. Under **Value**, enter the prompt parameters.
7. Click **Add** again to enter another prompt, or click OK to finish.

Retry Prompts Property

This property defines prompts to be played to the caller if input verification against specified digits fails.

1. Click the **Failure Prompts** row in the block's property table.
2. Click the ⋯ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under **Type,** select one of the following:
   - **Announcement**--Plays an announcement to the calling party. In this case, use the Value field to specify the prompt elements (can be up to 10). Each element is named with a number ranging from 1 to 10 and contains a number of entries describing the announcement elements. Announcement prompts do not have a user association.
   - **FormattedDigits**--Used to collect digits from the caller.
   - **Text**--Essentially the same as Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when Announcement is not supported.
   - **User Announcement**--Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.
5. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
6. Under Value, enter the prompt parameters.
7. Click **Add** again to enter another prompt, or click OK to finish.

Success Prompts Property

This property defines this list of prompts to be played to the caller if input is successfully verified against specified digits.

1. Click the **Failure Prompts** row in the block's property table.
2. Click the ⋯ button to open the Prompts dialog box.
3. Click **Add** to add a prompt.
4. Under **Type**, select one of the following:
    - **Announcement**--Plays an announcement to the calling party. In this case, use the Value field to specify the prompt elements (can be up to 10). Each element is named with a number ranging from 1 to 10 and contains a number of entries describing the announcement elements. Announcement prompts do not have a user association.
    - **FormattedDigits**--Used to collect digits from the caller.
    - **Text**--Essentially the same as Announcement, except all elements are of type text. This option does not allow mixing recorded announcements with text-to-speech. Use when Announcement is not supported.
    - **User Announcement**--Announcements with a user association previously created with the Create User Announcement block can be played via this type of prompt.
5. Under **Interruptible**, select true or false to indicate if the caller can interrupt the message .
6. Under **Value**, enter the prompt parameters.
7. Click **Add** again to enter another prompt, or click OK to finish.

Dtmf Verification Option Property

This attribute determines which verification scheme is used. Select one of the following:

- **Compare Digits**--Input is compared against specified digits.
- **Local Table**--Lookup Local table indexing associated with a user id.
- **Compare Dialing Plan Format**--Format compliance with a specified dialing plan.

Verification Attempts Property

Select the variable to contain the number that determine the number of attempts to be made for verifying collected digits.

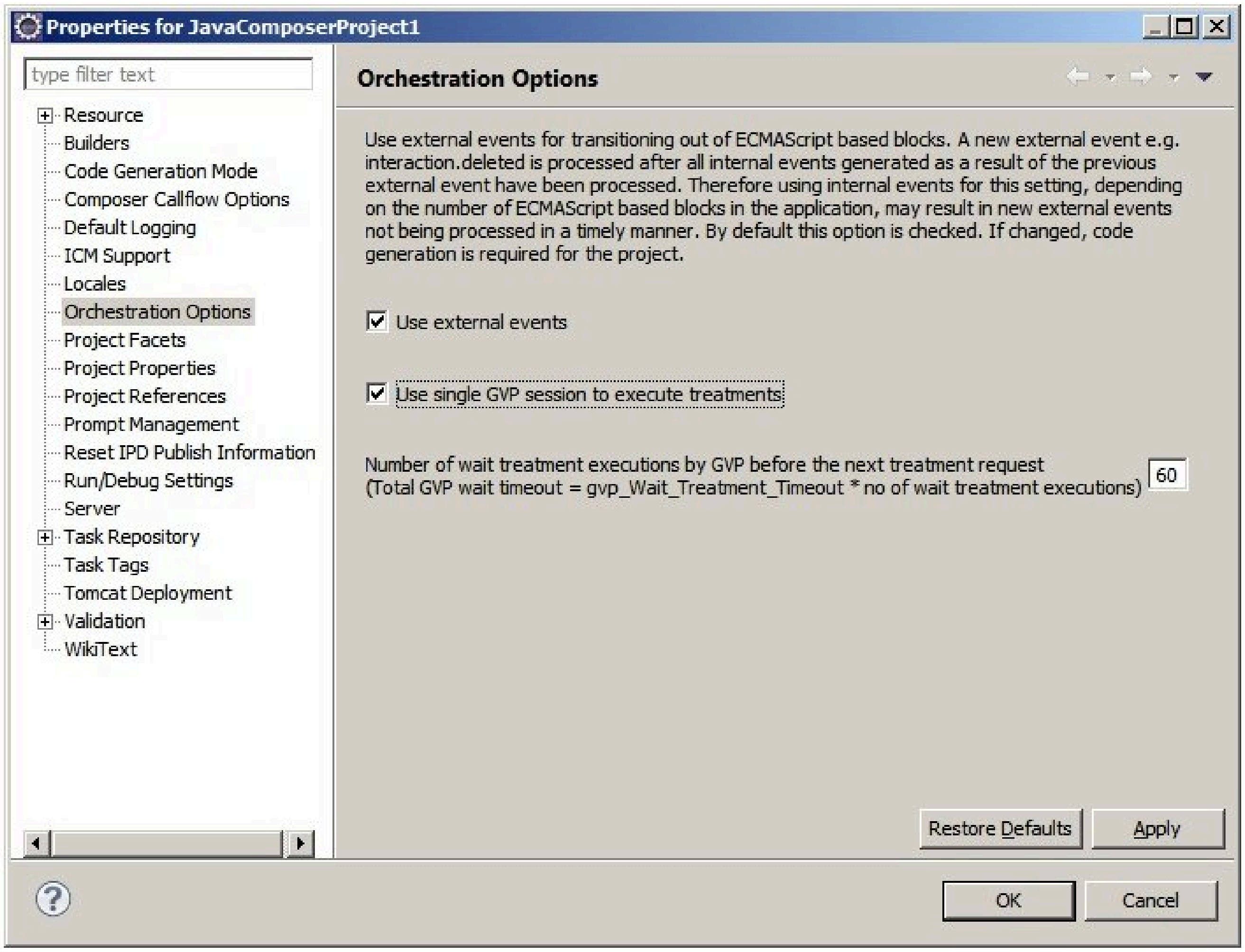
Verification Data Property

Select the variable to contain the verification data. This property determines what the input digits are compared against.

Single Session Treatments

Starting with Composer 8.1.300.78, Composer supports ORS-centric GVP single-session treatments. When using the Play Application, Play Sound (Music and ARM Types) Exit, and Disconnect blocks, voice applications can now optionally use a single VXML session on Media Control Platform to play/run multiple treatments instead of using one session per treatment. This enables DTMF buffering between multiple MSML treatments.

A new static master VXML Page is used for playing treatments in one GVP session: (e.g., `ComposerPlayTreatments.vxml`). Treatment URLs to play can be Composer diagram-generated vxml file, MCP built in treatments or Third party sub dialog URLs. The `ORSSessionID` will be provided as initial Play Application Input parameters; successive HTTP requests can pass treatment input parameters. Treatment execution results and return results will be passed back to the ORS session. The Orchestration Options dialog box is used for this functionality.



Orchestration Options

Use external events for transitioning out of ECMAScript based blocks. A new external event e.g. interaction.deleted is processed after all internal events generated as a result of the previous external event have been processed. Therefore using internal events for this setting, depending on the number of ECMAScript based blocks in the application, may result in new external events not being processed in a timely manner. By default this option is checked. If changed, code generation is required for the project.

☑ Use external events

☑ Use single GVP session to execute treatments

Number of wait treatment executions by GVP before the next treatment request (Total GVP wait timeout = gvp_Wait_Treatment_Timeout * no of wait treatment executions) 60

Workflow Block Changes

- Main workflow -> Exit /Disconnect Block: Generates code to end the GVP session if active based on `KeepGVPSessionAlive` and `isGVPSessionActive` flags.
- Play Application / Play Sound (music) -> Change in code to simulate treatments.

IPD Diagram Changes

- New event handlers have been added to handle communication between the GVP master page and the SCXML application.

1. `composer.dialog.play` - Play Application block request to IPD diagram for treatment request after the Master VXMl page is in action.
2. `composer.dialog.start.done` - Maste vxml page request to ORS session for treatment done.
3. `error.composer.dialog.start` - Master vxml page request to ORS session for treatment error.
4. `composer.dialog.exit` - Exit/Disconnect block request to IPD diagram for exiting GVP session.
5. `composer.workflow.proceed` - IPD diagram to Play Application block event to proceed further (compare to dialog.start.done or dialog.start.requestid).
6. `composer.workflow.exit` - IPD diagram response event to `composer.dialog.exit` request from Exit or Disconnect block event to proceed exit (After GVP session exists response "this event" is thrown).

Play Application Parameter Passing

In the traditional ORS treatments model, Play application treatment parameter passing happens via User Data. which has been sent along with the treatment request to GVP. In the Single GVP session model, User Data availability will be a limitation as the successive treatment requests happens via events. However, there are no changes to the input/output parameters defining mechanism in the Play Application block. That means the input/output parameters in Play Application blocks will be sent to the master VXML via events and then be propagated to each subdialog in system_treatment_params json object.

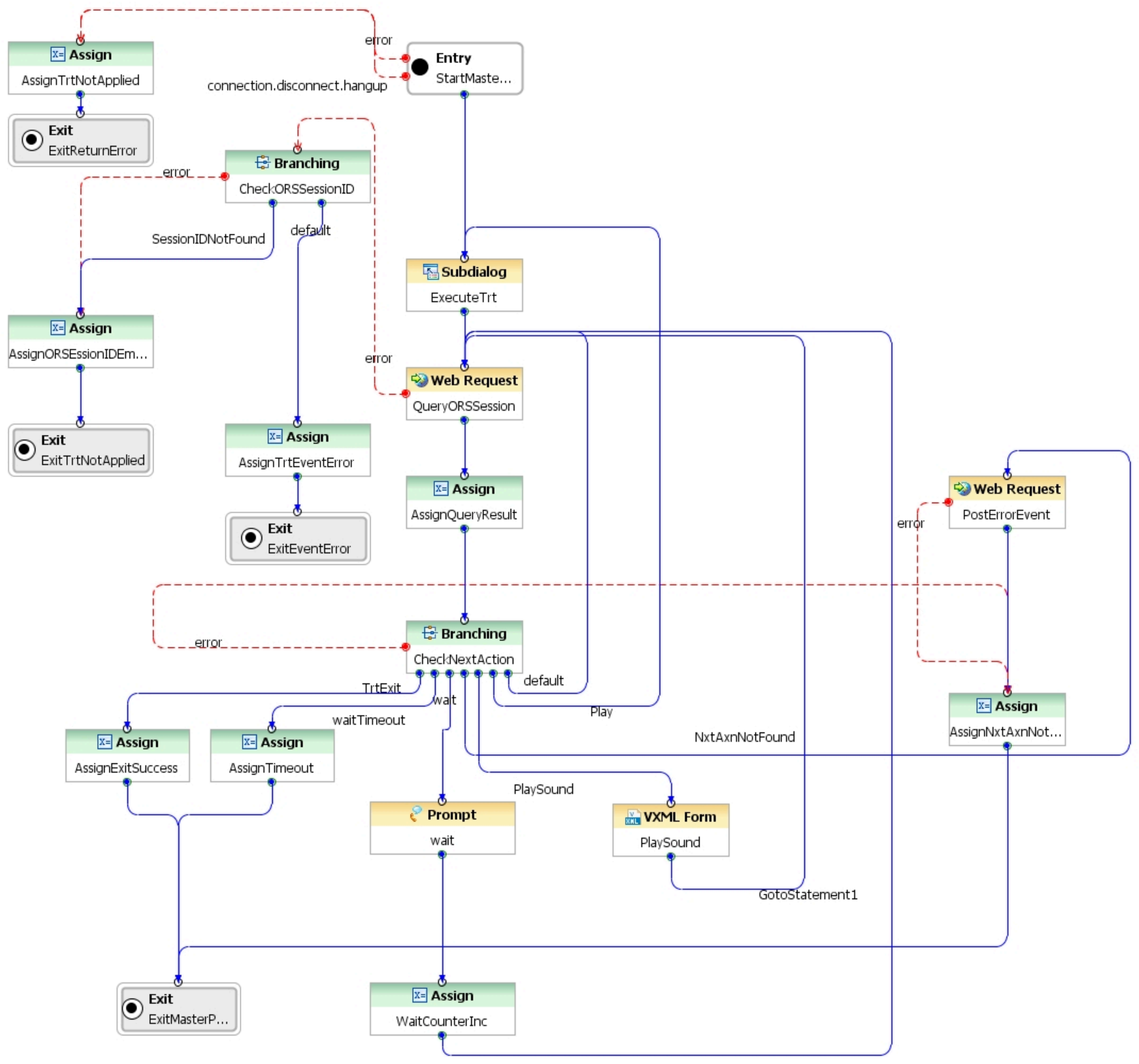
To receive Input parameters:

- In sub-callflows, instead of using InteractionData blocks to access userdata, use the system_treatment_params object's properties. First <form> in the Sub-callflow should have the following lines to receive the input parameters and the treatment language:

Note: Even if the Play Application block doesn't send or receive parameters, *the above <*`var`*> tags must be added to the Sub-Callflows since the Master VXML always sends these parameters.*

To Return Output Parameters to Play Application blocks: No changes here; variables defined in the Exit block will return to the Play Application block (if `Wait for Treatment End` is set to true).

Master VXML Flow Page:



GVP Wait and Session Timeouts:

The GVP Master VXML application can wait to receive the next treatment request from the ORS session. The GVP wait timeout before requesting ORS is now controlled on the ORS side instead of GVP side due to Prompt queuing impacts. Following are the options available to control GVP wait timeout between treatments and GVP session timeout:

1. `gvp_Wait_Treatment_Timeout` variable on the IPD diagram file will be used to delay responses to GVP when no treatment is ready to be executed. The Default value of this variable is 1s and this can be configured at the ERS object -> Application Params section in Configuration Manager. If configured the new value will take effect.

2. Orchestration Option -> Property page -> Number of wait treatment requests should GVP wait to end the session. When there is no treatment to play for a while, this option is used to control the GVP session timeout.
3. Total GVP session timeout = Number of wait requests * `gvp_Wait_Treatment_Timeout`. For example, 10 continuous wait requests with a delay of 3 seconds will provide 30-seconds of GVP Session timeout for a idle case -> 10 * 3s = 30 seconds.

Frequently Asked Questions

How can I run treatments in a separate GVP session while not disturbing the Master VXML session?
Using Assign blocks, set flag `_data.keepGVPSessionAlive` to `false` before your treatment block and reset to `true` after your treatment block.

What if ORS logic does not have any treatments after the first Play Application treatment?
The Master VXML session will poll ORS (IPD Diagram SCXML) based on the `gvp_Wait_Treatment_Timeout` value until it reaches the maximum GVP Session Timeout (configurable via Composer Project properties) and ends the session as GVP Session timeout.

What if the workflow wants to end the Master VXML session in the middle of a call?
The workflow may have to send the 'exit' treatment request to GVP to end the session.

How do I send and receive parameters in this GVP Single Session model?
Passing input parameters to the GVP session will continue to work using the Udata methodology with no changes. Receiving output parameters will happen via `composer.dialog.start.done` event (`composer_treatment_result` option).

What if the "GVP Keepalive" option is unchecked in the Orchestration Options property page?
Code generation is required whenever there are changes in the Orchestration Options properties page. If the "GVP Keepalive" option is unchecked, code generation will generate code for normal ORS-MCP paradigm treatments.

How do I end the Master VXML treatment session in a workflow diagram?
Use SCXML state block to generate an "exit" request to the GVP Master Session.

What if the treatment URL is wrong or not available?
The Master VXML will try to execute the treatment URL as subdialog and return back the return results. If the URL cannot be executed, `error.composer.dialog.start` will be returned back as error result to IPD diagram.

How do I cancel the Master VXML treatment dialog using ORS Cancel Call?
The Master VXML session's dialog `requestID` will be available in the `_data.gvpMasterTreatmentRequestID` variable of the application. To manually end the GVP master VXML session use the following code snippet in a SCXML State block.

```
<state id="CancelMasterVXML">
<onentry>
            <script>
```

```
                        var composerRequestID = _sessionid+'_'+generateComposerTrtRequestID();
            </script>
<if cond="_data.keepGVPSessionAlive==true &&_data.isGVPSessionActive==true">
                        <send event="'composer.dialog.exit'">
                                    <param name="treatment_id" expr="'exit'"/>
                                    <param name="treatment_url" expr="''"/>
                                    <param name="treatment_GVPRequestID"
expr="composerRequestID"/>
                                    <param name="treatment_waitForEnd" expr="'true'"/>
                        </send>
                        <else/>
                                    <raise event="'proceedNext'"/>
            </if>
            </onentry>
            <transition event="composer.workflow.exit" cond="_event.data.requestID ==
composerRequestID"
            target="$NextBlock$">
<script>
                        _data.isGVPSessionActive =false;
                        _data.keepGVPSessionAlive = false;       // Optional - Set
keepGVPSessionActive to false
            to completely end the Single GVP session and run ordinary ORS treatments
onwards.
</script>
            </transition>
            <transition event="proceedNext" target="$NextBlock$">
            </transition>
```

What if the treatment URL VXML page is a Main type callflow (does not have the `<return>` tag, but instead the `<exit>` tag)?
If a Main type callflow diagram is used, a treatment VXML page would have an `<exit>` tag, which will end the GVP session and return back to ORS. As a result, successive treatment requests will start a new Master VXML page. For this reason, Sub callflows are required to maintain a Master VXML page session.

When an ORS/workflow diagram does not have any treatment to execute, what will GVP do and how often will it poll an ORS session?
GVP will execute "wait" (silence) treatments while waiting for the next treatment request from ORS. GVP polls ORS based on the `gvp_Wait_Treatment_Timeout` value until the GVP session timeout is reached.

eService Blocks

The eServices blocks are used within the Workflow block to create a routing workflow for specialized processing of multimedia interactions. Also see the topic: IRD Functionality Included in Composer.

Composer Route Block Name	Block Has Usable Output Data to be used in App?	Purpose
An **ECMAScript function** allows you to manually attach Classification categories to interactions	Attach Categories	Segment interactions to different logical branches based on the different categories.
Email Forward	No Redirect E-mail Reply from External Resource	The Forward Type property specifies the type of functionality by allowing you to select Forward, Reply to Customer, or Redirect.
Email Response	Auto-acknowledgement Autoresponse Create Notification	Combines the functionality of IRD's Acknowledgement, Autoresponse, and Create Notification objects.
Chat Transcript	Yes	Generates a reply e-mail to a chat interaction and attaches a chat transcript.
Classify Interaction	Yes	Classifies a text-based interaction based on content, and attach one or more Classification categories to the interaction.
Create E-mail	Yes	Creates an e-mail to be sent out
Identify Contact	Yes	Identifies a contact based on the interaction User Data. Returns a list of matching Contact IDs based on the User Data. Creates a contact record in the UCS Database or update the UCS Database record of the matching contact
Send E-mail	No	Sends an -email message created with Create E-mail block
Screen Interaction	Yes	Screens a text-based interaction for specific content (specific words or patterns)
Create Interaction block	Yes	Creates an interaction record in the Universal Contact Server Database for a customer contact.
Create SMS	Yes	Creates a Short Message Service (SMS) message via an external SMS server

Render Message	Yes	Requests Universal Contact Server to create message content.
Send SMS	No	Send an SMS message created using the Create SMS block
Queue Interaction	No	Places a non-voice interaction in an existing queue
Route Interaction	No	Sends a non-voice interaction to one or more target objects: Agent, AgentGroup, PlaceGroup, Skill, or target contained in a variable.
Stop Interaction	No	Sends a request to Interaction Server to stop processing this interaction.
Update Contact	No	Updates customer profile information in the UCS Database, based on data attached to an interaction.
External Service(ESP)	Yes	Exchanges data with third party (non-Genesys) servers that use the Genesys Interaction SDK or any other server or application that complies with Genesys Interaction Server (GIS) communication protocol.
Special Note on Validation and Off-Line Mode: When using Composer in offline mode (not connected to the Configuration Server), you can edit block properties that depend on information from Configuration Server. Later, when you connect to Configuration Server and validate the Interaction Processing diagram, Composer will validate the values you entered in off-line mode.		

eServices ECMAScript Functions

These functions are available for use in Expression Builder.

- For classification segmentation, an ECMAScript function determines if a particular category name or ID exists in the array of category objects represented by an application variable. This variable can be the output of the Classify Interaction block, enabling the Branching block to be used for segmentation based on category.
- For manually attaching categories to an interaction, the User Data block can be used and then a branching block can be (optionally) used to segment interactions to different logical branches based on

the different categories.

For a summary of the eServices blocks, see Composer Equivalent to IRD Multimedia.

Composer Equivalent to IRD Multimedia

Composer includes multimedia functionality (processing non-voice interactions) that was previously provided through Genesys Interaction Routing Designer (IRD). The information below is provided for existing Genesys customers transitioning to Composer, who are familiar with using IRD's Multimedia objects. The IRD To Composer Migration Guide, available on the Genesys Documentation Wiki, details the migration process.

Composer Route Block Name	Block Has Usable Output Data to be used in App?	Equivalent IRD Block/ Object	Purpose
An ECMAScript function allows you to manually attach Classification categories to interactions	No	Attach Categories	Segment interactions to different logical branches based on the different categories.
Email Forward	No	Forward E-mail Redirect E-mail Reply from External Resource	The Forward Type property specifies the type of functionality by allowing you to select Forward, Reply to Customer, or Redirect.
Email Response	Yes	Autoknowledgement Autoresponse Create Notification	Combines the functionality of IRD’s Acknowledgement, Autoresponse, and Create Notification objects.
Chat Transcript	Yes	Chat Transcript	Generates a reply e-mail to a chat interaction and attaches a chat transcript.
Classify Interaction	Yes	Classify	Classifies a text-based interaction based on content, and attach one or more Classification categories to the interaction.
Create E-mail	Yes	Create E-mail	Creates an e-mail to be sent out
Identify Contact	Yes	Identify Contact	Identifies a contact based on the interaction User Data. Returns a list of matching Contact IDs based on the User Data. Creates a contact record in the UCS Database or update the UCS Database record of the matching contact

Send E-mail	No	Send E-mail	Sends an -email message created with Create E-mail block
Screen Interaction	Yes		Screens a text-based interaction for specific content (specific words or patterns)
Create Interaction block	Yes	Create Interaction	Creates an interaction record in the Universal Contact Server Database for a customer contact.
Create SMS	Yes	Create SMS	Creates a Short Message Service (SMS) message via an external SMS server
Render Message	Yes	Render Message Content	Requests Universal Contact Server to create message content.
Send SMS	No	Send SMS	Send an SMS message created using the Create SMS block
Queue Interaction	No	Queue Interaction	Places a non-voice interaction in an existing queue
Route Interaction	No	Route Interaction	Sends a non-voice interaction to one or more target objects: Agent, AgentGroup, PlaceGroup, Skill, or target contained in a variable.
Stop Interaction	No	Stop Interaction	Sends a request to Interaction Server to stop processing this interaction.
Update Contact	No	Update Contact	Updates customer profile information in the UCS Database, based on data attached to an interaction.
External Service (ESP)	Yes	External Service	Exchanges data with third party (non-Genesys) servers that use the Genesys Interaction SDK or any other server or application that complies with Genesys Interaction Server (GIS) communication protocol.

Special Note on Validation and Off-Line Mode: When using Composer in “offline” mode (not Server connected to the Configuration Server), you can edit block properties that depend on information from Configuration Server. Later, when you connect to Configuration Server and validate the Interaction Processing diagram, Composer will validate the values you entered in off-line mode.			
Other Composer Blocks for Multimedia Processing			
Interaction Process Diagram (IPD)	NA	Business Process	
Interaction Queue	NA	Interaction Queue	Allows you to define an Interaction Queue used for multimedia interactions.
Media Server	NA	Media Server	Represents an existing Media Server, such as a chat or e-mail server. Allows you to get interactions into an IPD.
Workflow	NA	Strategy	Points to an existing Workflow resource (.workflow diagram) to which an interaction can be sent for processing.

Also see Context Services Blocks Overview.

Chat Transcript Block

Use to create (but not send) an e-mail message that is generated from your site’s Standard Response Library and which has the customer’s chat transcript attached. Use the Send Email block to send the message out.

Use Case

1. A customer engages in a chat interaction with an agent.
2. The agent asks the customer if he wants to receive a chat transcript
3. The customer agrees, and the agent presses a button on his desktop that initiates an open media interaction into the Genesys system. The context of the interaction includes information provided by the agent desktop, including a customer ID and the subject of the chat.
4. This interaction initiates a routing workflow.
5. The routing workflow uses the information, such as the customer ID and the subject of the chat to identify additional customer details, such as customer name, from the Universal Contact Server database.
6. The routing workflow creates and sends an e-mail message that includes the chat transcript. The e-mail message uses text from the standard response library, which was retrieved based on the subject of the chat, and personalized to include the customer's first and last name.

Special Note on Cc, From, and Exclude Addresses Properties

The Literal and Variable types can have a value set to an actual e-mail address, e.g., joe@test.com, or refer to the name of a previously configured e-mail address from Configuration Server (e.g., if “Tech Support” is configured as a Configuration Server `E-mail Accounts` Business Attribute, then “Tech Support” can be the value for the Literal type and the platform will use that e-mail address).

The Chat Transcript block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Email Server Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Do Not Thread Property

Find this property's details under Common Properties.

Output Queue Property

Find this property's details under Common Properties.

CC Property

Find this property's details under Common Properties.

Exclude Email Addresses Property

Find this property's details under Common Properties.

Field Codes Property

Find this property's details under Common Properties.

From Property

Find this property's details under Common Properties.

Standard Response Property

Find this property's details under Common Properties.

Subject Property

Find this property's details under Common Properties.

To Property

Find this property's details under Common Properties.

Use Subject From SRL Property

Find this property's details under Common Properties.

Interaction ID Property

Find this property's details under Common Properties.

Output Result Property

Find this property's details under Common Properties.

Detach Property

Find this property's details under Common Properties.

Detach Timeout Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Classify Interaction Block

Use to have Universal Routing Server instruction Classification Server to assign one more category codes (configured in Knowledge Manager as described in the *eServices 8.1 User Guide*) to a text-based interaction. Once a category code is assigned, other types of processing can occur based on the category code.

ECMAScript Functions

- For classification segmentation, an Server Functions ECMAScript function (containsCategoryName) determines if a particular category name exists in the array of category objects represented by an application variable. This variable can be the output of the "Classify Interaction" block, enabling the Branching block to be used for segmentation based on category.
- Another Server Functions ECMAScript function (attachCategory) allows you to manually attach Classification categories to interactions, and then (optionally) segment interactions to different logical branches based on the different categories. The category or categories to be attached to an interaction can be based on classification applied to the interaction at a prior point in the routing workflow.

Classification Versus Screening

The primary difference between classification and screening (Screen Interaction block) is as follows:

- The result of screening for certain words or patterns is a Screening Rule name and the value of true or false.
- The result of classification based on content analysis (you must have the Content Analyzer option installed as described in the *eServices 8.1 User Guide*) is a category code, which can be associated with a Standard Response or used for other purposes, such as segmentation.

For more information on screening and classification, see the chapter on IRD objects in the *Universal Routing 8.1 Reference Manual*. Also see the *Universal Routing 8.1 Business Process User's Guide*.

Note: If your site's categories are not yet defined in Knowledge Manager, you may wish to use the Screen Interaction block, which performs word matching instead of using category codes.

Use Case

1. An e-mail arrives on a route point, initiating a routing workflow.
2. The routing workflow examines the content of the e-mail and attempts to classify the text based on classification categories set up in Knowledge Manager.

3. An auto-response e-mail is created, using a standard response from the standard response library. The response selected is based on the categorization of the email performed in step 2.
4. The auto-response email is sent to the customer.

The Classify Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Classification Categories Property

Use to select the location to store resulting classification categories after classification.

1. Click under Value to display the button.
2. Click the button to open the Classification Categories dialog box. For **Type**, select **User Data** or **Variable**.If you select **User Data**, specify the User Data key for the classification results. If you select **Variable**, select the variable that contains the User Data key for the classification results.

Categories Property

Use this property to select individual categories and sub-categories of the Root Category to be used in the classification process.

1. Click under Value to display the button.
2. Click the button to open the Categories dialog box.
3. Click Add to open the Select Items dialog box.

4. From the Type dropdown menu, do one of the following:

    - If you are Server connected to Configuration Server, select **Configuration Server**. Select one or more categories for the **Value**.
    - Select **Literal** and enter the categories in the **Value** field. Use commas to separate the categories.
    - Select **Variable** and select the variable that contains the categories from the **Value** field.

Classification Data Property

Use this property to select whether to search for classification data in the interaction's User Data, from a variable, or from the UCS database.

1. Click under **Value** to display the button.
2. Click the button to open the Classification Data dialog box.
3. For **Type**, select **UCS**, **User Data** or **Variable**. If you select **UCS**, don't specify fields. Data available as part of the interaction is automatically picked up for classification by ORS. If you select **User Data**, specify the key-value pairs. If you select **Variable**, select the variable that contains the key-value pairs.

Classification Server Property

Select the Application name for the Classification Server from those in the Configuration Database. If a Classification Server is not selected, the platform will internally select one.

1. Click under Value to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.

    - You can also select **Literal** or **Variable** from the **Type** dropdown menu. If you select **Literal**, enter the name of the classification server in the **Value** field.   If you select **Variable**, select the variable from the **Value** field.
    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Classification Server from the **Value** field.

Confidence Level Property

Select a variable that contains a number from 1 to 100 that reflects the minimum relevance percentage that each classification category must have in order for Classification Server to consider an interaction as belonging to that category.

Root Category Property

Select the name of the top-level category to be used for the classification.

1. Click under Value to display the ... button.
2. Click the ... button to open the Root Category dialog box.
3. The next step depends on whether you are connected to Configuration Server.

   - If you are connected, select **Configuration Server** from the **Type** dropdown menu. A tree of classification categories appears in the **Value** field. Next, select the name of the top-level (root) classification category. This is a directory that appears in Configuration Manager in the `Business Attributes` > `Category Structure` folder. For more information on Root folders, see the *Universal Routing 8.1 Reference Manual*.
   - You can also select **Literal** or **Variable** from the **Type** dropdown menu. If you select **Literal**, enter the name of the root category in the **Value** field.If you select **Variable**, select the variable that contains the root category from the **Value** field.

Subcategories Property

Click the down arrow and select one of the following:

- **Do not include children of selected categories.**
- **Include immediate children of selected categories.**
- **Include all children (recursively) of selected categories.**

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Create E-mail Block

Use this block to create an e-mail message to be sent out to a customer or another agent and to specify the interaction queue where the outbound e-mail should be placed. The selected interaction queue appears as a Queue Reference block in the interaction process diagram.

To create the e-mail text, you have the following options:

- Use text from your Standard Response Library (SRL). In this case, you must specify the SRL identifier defined in eServices/Multimedia Knowledge Manager.
- Use text associated with the default/active Standard Response for a specified Knowledge Manager Category code. In this case, you must specify the Category code defined in Knowledge Manager.

Category trees are a means of organizing and gaining access to the library of Standard Responses. In the Universal Contact Server Database, each Standard Response must be associated with at least one Category code. Content Analyzer can classify an incoming e-mail in terms of the Category tree.

See the *Knowledge Manager 8.1 Help* for more information on Standard Responses and Category codes.

Use Case

1. An inbound interaction initiates a routing workflow.
2. Based on the interaction the customer is identified and the User Data of the interaction is updated.
3. The User Data is then assigned to variables and is used to create an e-mail response with the First Name, Last Name and the contact address of the customer.
4. The e-mail created for this customer uses the User Data to find an appropriate response from the Standard Response Library as well. The Standard Response selected is based on some purpose inferred from the customer's original e-mail

Special Note on From and To Properties

The Literal and Variable types can have a value set to an actual e-mail address, e.g., joe@test.com, or refer to the name of a previously configured e-mail address from Configuration Server (e.g., if “Tech Support” is configured as a Configuration Server E-mail Accounts Business Attribute, then “Tech Support” can be the value for the Literal type and the platform will use that e-mail address).

The Create E-mail block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Email Server Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Do Not Thread Property

Find this property's details under Common Properties.

Output Queue Property

Find this property's details under Common Properties.

From Property

Find this property's details under Common Properties.

Include Original Message Into Reply Property

Find this property's details under Common Properties.

Standard Response Property

Find this property's details under Common Properties.

Subject Property

Find this property's details under Common Properties.

To Property

Find this property's details under Common Properties.

Use Subject From SRL Property

Find this property's details under Common Properties.

Create New Interaction Property

Select true or false to indicate whether a new interaction record should be created in the Universal Contact Server Database for this outbound e-mail. The default is false.

Interaction ID Property

Find this property's details under Common Properties.

Output Result Property

Find this property's details under Common Properties.

Detach Property

Find this property's details under Common Properties.

Detach Timeout Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Create Interaction Block

This block is hidden from the eServices palette by default. To make the block visible, right-click on the eServices palette title bar and select **Customize** from the Palette Group menu. A dialog box opens where you can de-select **Hide**.

Use this block to create an interaction record in the Universal Contact Server Database, for a customer contact. This saves the current interaction being processed in the database.

Note: For "native" Genesys interaction types such as e-mail and chat, interactions are automatically created in the UCS database. The Create Interaction block would be used primarily for creating interactions in UCS when there are new interactions coming into the Genesys system through the Open Media interface. This is the primary mechanism for making interaction data available to an agent desktop application.

Use Case

1. An incoming fax initiates a routing strategy (routing workflow).
2. The fax server has OCR (character recognition) to extract the customer details from the fax. These are identified and attached to the interaction.
3. The routing strategy creates an interaction in the UCS database, associated with this customer.

The Create Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Data MIME Type Property

Use this property for an interaction with binary content to specify the Types MIME Type for the binary content. Click the down arrow and select one of the following MIME types or a variable that contains the MIME type. This information helps Agent Desktop Application to choose the proper application to process or display the binary part of the interaction. This field cannot be empty if the Data User Data

Key contains a value.

application/msword	audio/mpeg	message/sipfrag
application/octet-stream	audio-mpeg4-generic	message/tracking-status
application/postscript	image/g3fax	multipart/alternative
application/rtf	image/gif	multipart/form-data
application/vnd.ms-powerpoint	image/jpeg	multipart/mixed
application/vnd.ms-project	image/tiff	multipart/parallel
application/vnd.visio	message/delivery-status	multipart/voice-message
application/voicexml+xml	message/http	text/html
application/xml	message/news	text/plan
application/xml-dtd	message/partial	text/richtext
application/zip	message/rfc822	text/xml
audio/basic	message/sip	

Data User Data Key Property

If the interaction contains binary data, select the variable that contains the User Data key whose value is the interaction binary content.

Structured Text MIME Type Property

If the interaction contains structured text, use this property to specify the MIME type. Select the variable that contains the MIME type for the interaction's structured text or select one of the following:

- **text/html**
- **text/plain**
- **text/rich text**
- **text/html**

What you select here assists the Agent Desktop Application in choosing the proper application to process or display this part of interaction. This field is mandatory if the Structured Text User Data Key property contains a value.

Structured Text User Data Key Property

If the interaction contains structured text, select the variable that contains the User Data whose value is the interaction structured text. The value will be stored in the Universal Contact Server Database as interaction’s structured text.

Text User Data Key Property

If the interaction contains plain text, select the variable that contains the User Data key whose value is the interaction plain text. The value will be stored in the UCS Database as the interaction's plain text.

Exceptions Property

Find this property's details under Common Properties.

Also see Exception Events for eServices/UCS Blocks. Exceptions supported: 201, 202, 203, 204, 502, 512, 701, 710, 716, 732.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Orchestration Interaction ID Property

Select the variable that contains the interaction identifier for Orchestration Server.

Output Interaction ID Property

Select the variable that contains the identifier for the created interaction.

Bind With Contact Property

Select true to have Universal Contact Server look for a customer contact record and associate it with the interaction. If a contact record is not found, it will be created.

Can Be Parent Property

Select true if this interaction can have child interactions.

Do Not Thread Property

Find this property's details under Common Properties.

Tenant Property

Find this property's details under Common Properties.

Universal Contact Server Property

Find this property's details under Common Properties.

Update Interaction User Data Property

Find this property's details under Common Properties.

Create SMS Block

Use this block to create an outbound message, which can be sent out as a Short Message Service (SMS) message to an external SMS Server. SMS refers to the common text messaging service available on cellphones and other handheld devices. You can also specify the interaction queue where the SMS message should be placed. This version of Composer supports native SMS, which means that the SMS message is sent via SMS Server and not via an e-mail to SMS gateway.

Also see the Render Message block.

The Create SMS block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

SMS Server Property

To select an SMS Server for sending the message:

1. Select the SMS Server row in the block's property table.
2. Click under Value to display a down arrow.
3. Select an SMS Server from those in the Configuration Database.

Exceptions Property

Find this property's details under Common Properties.

Interaction Queue Property

Select the interaction queue where the SMS Server should place the outbound SMS. Only the interaction queues that have been created using Composer in the current Composer Project are shown for the Configuration Server values. The interaction queues are sorted per parent Interaction Process Diagram.

To define the output queue:

1. Click under Value to display the button.
2. Click the button to open the Select an Output Queue dialog box.
3. Click the Type down arrow and do one of the following
   - If you are connected to Configuration Server, select **Configuration Server** and then select an output queue as the Value. The **Independent Objects** > **Same queue** choice allows you to put the outbound e-mail in the same interaction queue that initiated the current interaction.
   - Select **Literal** and then enter the name of the output queue as the **Value**.
4. Click OK.

Interaction Subtype Property

Select an Interaction Subtype code (defined in the Configuration Database Business Attributes folder), which can represent an Acknowledgement, Autoresponse, or Outbound New Standard Response subtype. The default is OutboundNew.

Message Destination Number Property

Enter the mobile telephone number of the person to whom the message is to be sent.

See the Getting and Using E-mail Address topic for more information. While the topic focuses on the supplying a To e-mail address, the same ideas apply to getting a customer's mobile telephone number with Context Services blocks and using it for this property.

Message Source Number Property

Enter the mobile telephone number from which the SMS message should appear to come.

Message Text Property

Use this property to specify the Short Message Server text.

1. Click under Value to display the button.

2. Click the ⋯ button to open the Select SMS Text Message dialog box.
3. Select **User Data**, **Literal** or **Variable** from the Type dropdown menu.

If you select **Literal**, enter the text in the **Value** field.

- If you select **User Data**, enter the User Data keys in the **Value** field.
- If you select **Variable**, select the variable from the **Value** field that contains the text.

4. Click **OK** to close the dialog box.

Interaction ID Property

Find this property's details under Common Properties.

Output Result Property

Find this property's details under Common Properties.

Detach Property

Find this property's details under Common Properties.

Detach Timeout Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Email Forward Block

Use to send an incoming e-mail to an external address, such as for agent collaboration. This block combines the functionality of IRD's Forward E-mail, Redirect E-mail, and Reply E-mail from External Resource objects. The Forward Type property specifies the type of functionality by allowing you to select Forward, Reply to Customer, or Redirect.

- The difference between Forward and Redirect is as follows: Use the Forward functionality when there is an expectation of getting a response back. Use the Redirect functionality when there is no expectation of getting a response back.

- Reply To Customer works with the Forward functionality. It takes the resulting external resource reply inbound e-mail as input, extracts the external resource reply text from it, creates a customer reply outbound e-mail, and submits the e-mail to Interaction Server using the specified interaction queue.

Note: An "external resource" is a name for any object outside the contact center. It may be an external agent or another contact center. Configure external e-mail addresses as E-Mail Accounts Business Attributes in the Configuration Database.

Use Case for Forward and Reply to Customer

1. An e-mail about a product defect arrives to the contact center, initiating a routing workflow.
2. Based on content analysis, the contact center determines that it must be replied to by an outsource partner responsible for supporting the defective product.
3. The routing workflow invokes the Forward functionality of this block, and the e-mail is sent to an external address. The e-mail uses text from the Standard Response Library to indicate to the outsourcer what the service-level agreement is on such customer inquiries.
4. The outsourcer partner replies to the e-mail with a response on the defective product.
5. The routing workflow then employs the Reply to Customer capability. It take the response from the outsourcer partner, reformats it appropriately, and sends it as a response to the original customer inquiry.

Use Case for Redirect

Use the redirect functionality to send an incoming e-mail to an external address without expecting a response or when there is no need for further processing.

1. An inbound e-mail interaction initiates a routing workflow.
2. Based on a content analysis of the e-mail, the e-mail is re-directed to the Brokerage business unit that is outside of the contact center.
3. The e-mail is handled directly by a broker (knowledge worker) in the Brokerage business unit. The contact center does not expect a response, or to be involved in further processing of the email.

Special Note on Cc, From, and Exclude Addresses Properties

The Literal and Variable types can have a value set to an actual e-mail address, e.g., joe@test.com, or refer to the name of a previously configured e-mail address from Configuration Server (e.g., if Tech Support is configured as a Configuration Server `E-mail Accounts` Business Attribute, then Tech Support can be the value for the Literal type and the Orchestration platform will use that e-mail address.

The E-mail Forward block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Email Server Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Do Not Thread Property

Find this property's details under Common Properties.

Forward Type Property

As described at the start of this topic, select one of the following:

- **Forward**—forward to an external resource with the expectation of getting a response back
- **Reply to Customer**—takes the reply inbound e-mail as input, extracts the reply text from it, and

creates a customer reply outbound e-mail

- **Redirect**—forward to an external resource with no expectation of getting a response back

CC Property

Find this property's details under Common Properties.

Exclude Email Addresses Property

Find this property's details under Common Properties.

Field Codes Property

Find this property's details under Common Properties.

From Property

Find this property's details under Common Properties.

Include Original Message Into Reply Property

Find this property's details under Common Properties.

Standard Response Property

Find this property's details under Common Properties.

Subject Property

Find this property's details under Common Properties.

To Property

Find this property's details under Common Properties.

Interaction ID Property

Find this property's details under Common Properties.

Output Result Property

Find this property's details under Common Properties.

Detach Property

Find this property's details under Common Properties.

Detach Timeout Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Email Response Block

Use to send an e-mail in response to incoming interaction resulting from inbound e-mail or an open media request. This block combines the functionality of IRD’s Acknowledgement, Autoresponse, and Create Notification objects.

Autoresponse Use Case

1. An inbound e-mail initiates a routing workflow.
2. URS prioritizes the interaction,
3. The interaction is screened and customers are segmented based on tier.
4. If the screened e-mail is identified as one that does not require any agent input, then the e-mail is provided with autoresponse as an Re: with the text from to the original e-mail included
5. The next step in the routing workflow stops processing the interaction.

Acknowledgement Use Case

1. A Genesys user sends an e-mail request to create a ticket on specific problem involving a T-Server.
2. Genesys identifies the customer contact and sends out an acknowledgement e-mail.
3. The acknowledgement e-mail uses custom fields (Field Codes), personalizing a standard e-mail with the customer's name.
4. The acknowledgment e-mail also contains the ticket number and contact information for the technical support engineer assigned to the ticket .
5. The technical support engineer is also copied in on the email

Create Notification Use Case

Use Create Notification to create a notification e-mail that can be sent to a customer as a reply to an inquiry. (e.g phone call, e-mail, SMS, Chat, etc). This e-mail may itself contain the response to the inquiry or it may point the customer to the location of the information; for example, a page on the enterprise website, a link to youtube, and so on. And the e-mail can be classified as either Acknowledgement, Autoresponse or Notification. The response may be a template from knowledge management but not necessarily. Use case:

1. A Customer logs into a bank web site using his username and password.
2. The bank web site provides a way to send an inquiry to customer support.

3. The customer sends an inquiry to the bank's customer support, asking about the status of a check he had deposited yesterday through an ATM. He is wondering when the funds will be available to him through his account.
4. The customer support analyst provides a response to the customer. This response is only available to the customer through a secure log-in on the bank's web site, due to its sensitive nature.
5. The contact center also sends a notification email to the customer's Gmail address, telling him that there is a response awaiting him on the bank's website, providing a URL to that part of the bank's web site.
6. The customer logs into the bank's web site and retrieves the response to his inquiry.

Special Note on From and To Properties

The Literal and Variable types can have a value set to an actual e-mail address, e.g., joe@test.com, or refer to the name of a previously configured e-mail address from Configuration Server (e.g., if “Tech Support” is configured as a Configuration Server `E-mail Accounts` Business Attribute, then “Tech Support” can be the value for the Literal type and the platform will use that e-mail address).

The E-mail Response block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Email Server Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Do Not Thread Property

Find this property's details under Common Properties.

Open Media Property

Select true or false to indicate if the e-mail response is a result of an incoming open media interaction. Note: If you select true, this Email Server property above should reflect the e-mail server that has been adapted to handle the appropriate media type.

Output Queue Property

Find this property's details under Common Properties.

Response Type Property

Select one of the following Interaction Subtypes:

- **Acknowledgement**
- **Autoresponse**
- **Notification**

For more information, see the Create Notification Use Case section above.

CC Property

Find this property's details under Common Properties.

Exclude Email Addresses Property

Find this property's details under Common Properties.

Field Codes Property

Find this property's details under Common Properties.

From Property

Find this property's details under Common Properties.

Include Original Message Into Reply Property

Find this property's details under Common Properties.

Standard Response Property

Find this property's details under Common Properties.

Subject Property

Find this property's details under Common Properties.

To Property

Find this property's details under Common Properties.

Use Subject From SRL Property

Find this property's details under Common Properties.

Interaction ID Property

Find this property's details under Common Properties.

Output Result Property

Find this property's details under Common Properties.

Detach Property

Find this property's details under Common Properties.

Detach Timeout Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Identify Contact Block

This block is hidden from the eServices palette by default. To make the block visible, right-click on the eServices palette title bar and select Customize from the Palette Group menu. A dialog box opens where you can de-select Hide. This block can be used for various purposes. You can:

- Identify a contact based on the User Data of the current interaction.
- Return a list of matching Contact IDs based on the User Data. This occurs only if a single matching contact record is found or if the Return Unique property is set to false. Contact attribute values (first name, last name, email address, and so on) are returned only when a single matching contact is found (no matter what is the value of the Return Unique property).
- Create a contact record in the Universal Contact Server (UCS) Database with information in the User Data if a matching contact is not found.
- Update the interaction's User Data with data returned by UCS.

Important! See Mandatory User Data For UCS Blocks. Also see the section on Contact Identification and Creation in the *eServices 8.1 User's Guide*.

Use Case

1. An inbound interaction initiates a routing workflow.
2. Based on data attached to the interaction, the contact is identified from UCS, and the interaction's user data is updated.
3. The user data is then assigned to variables and is then used to provide an automated response that is personalized with the First Name, Last Name and the contact address of the contact.

The Identify Contact block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Context Services Exception Events. Also see Exception Events for eServices UCS Blocks where the following exceptions are supported: 201, 203, 502, 510, 701, 710, 716, 730.

Interaction ID Property

Find this property's details under Common Properties.

Contact Count Property

Select the variable whose value will contain the number of identified contacts.

Contact Created Property

Select the variable whose value will be set to true if a contact has been created in the database; otherwise the variable will be set to false.

Contact List Property

Select the variable whose value will contain the list of identified contacts, or the identifier of the newly created contact.

Result Property

Select the variable which will contain the data returned by the Universal Contact Server.

Variables Mapping Property

Find this property's details under Common Properties.

Create Contact Property

- Select **true** to specify that a new contact record should be created in the database when no matching contact is identified based on the interaction's User Data.
- Select **false** to specify that a new contact record should not be created in the database when no matching contact is identified based on the interaction's User Data.

Return Unique Property

- Select **true** to not have matching Contact IDs returned when multiple contacts are identified in the database based on the interaction User Data.
- Select **false** to have matching Contact IDs returned when multiple contacts are identified in the database based on the interaction User Data.

Tenant Property

Find this property's details under Common Properties.

Universal Contact Server Property

Find this property's details under Common Properties.

Update Interaction User Data Property

Find this property's details under Common Properties.

Update User Data Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Render Message Block

Note: This block is hidden from the eServices palette by default. To make the block visible, select Customize from the Palette Group menu.

Use the Render Message block to request Universal Contact Server to create message content. You can create message content using text from either the Message Text to Render property, the Result property, or User Data. This block causes Universal Routing Server to generate a request to Universal Contact Server for the method RenderMessageContent. The primary reason for this block is to create message content for use in the Create SMS block, which does not allow you to use either Standard Response Library text or Field Codes when defining the message text.

Important! See Data for UCS Blocks Mandatory User Data For UCS Blocks.

Accessing the Rendered Message

The results of this request to the Universal Contact Server can be accessed by the Composer developer in two ways. You may:

- Assign the result to an application variable.
- Have the result attached to the current interaction.

Use Case

1. An inbound e-mail interaction initiates a routing strategy (routing workflow).
2. The e-mail is classified based on the email subject and keywords.
3. The routing workflow logic creates a message to send to the customer by SMS. The SMS message text is rendered by pulling data from the Standard Response Library, using Field Codes to insert the customer's name into the message text.
4. The SMS interaction is created in the routing workflow and then sent to the customer.

The Render Message block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties. Also see Events for eServices UCS Blocks Exception Events for eServices UCS Blocks. Exceptions supported: 105, 201, 203, 510, 701, 710, 716, 732.

Interaction ID Property

Find this property's details under Common Properties.

Result Property

Click the down arrow under Value and select the variable to hold the rendered text.

- When the **Message Text to Render** property is **Variable**, **Literal**, **UserData** or **UserDataVariable**, the `Result` variable holds the rendered text.
- When the **Message Text to Render** property is **ConfigServer**, the `Result` variable holds a JSON object having the three properties: Subject, Text and StructuredText.

Field Codes Property

When using a standard response to render message text, use this property to assign values to Field Code variables that have been defined in Knowledge Manager (as described in the eServices 8.1 User's Guide) and used in that standard response. Universal Contact Server requires values for Field Codes when using standard responses that include Field Codes.

1. Click under Value to display the button.
2. Click the button to open the Field Codes dialog box.
3. Click **Add**. A second dialog box opens for specifying Field Codes and values.
4. Type the name of the Field Code.
5. Select **Literal** or **Variable**.
    - If you select **Variable**, select the variable the contains the Field Code value.
    - If you select **Literal**, enter the value for the Field Code.

Message Text to Render Property

Use to specify the content to be rendered using Universal Contact Server.

1. Click under Value to display the button.
2. Click the button to open the Message Text to Render dialog box.
3. The next step depends on the source for the rendered text.

   - If you connected Configuration Server and wish to use a standard response, select **Configuration Server** from the **Type** dropdown menu. In the **Value** field, select the name of the standard response from the tree. Note: For a standard response to be selectable in this dialog box, you must first define it in Knowledge Manager for the same tenant selected when connecting to Configuration Server. You must also approve its use and make it active. The Knowledge Manager procedures for setting the Acknowledgement and Approved flags, and making the standard response Active are covered the *eServices 8.1 User's Guide*. See "Filling Out the Additional Tab" in the chapter on Knowledge Management Basics.
   - To type the rendered text, select **Literal** and enter the text in the **Value** field.
   - To have the rendered text taken from the interaction's User Data, select **UserData**. In the **Value** field, enter the User Data keys.
   - To have the rendered text taken from a variable in the User Data, select **UserDataVariable**. In the **Value** field, select the name of the variable.
   - To have the rendered text taken from a variable (Project or Workflow), select **Variable**. In the **Value** field, select the name of the variable.

Tenant Property

Find this property's details under Common Properties.

Universal Contact Server Property

Find this property's details under Common Properties.

Update Interaction User Data Property

Find this property's details under Common Properties.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Screen Interaction Block

Use this block to filter a text-based interaction for specific content (specific words or patterns) based on evaluation of one or more screening rules by Classification Server. You then have the option of segmenting incoming interactions to different logical branches based on the result of the screening query. Screening rules are created in Knowledge Manager as described in the *eServices 8.1 User's Guide*.

Key-Value Pairs

If Classification Server finds a match, it returns a response that contains the following:

Key	Value
ScreenRuleName	The name of the screening rule.
Id	The actual identifier of the screening rule in the UCS Database.
ScreenRuleMatch	True (if Classification Server finds a match), or False (if it does not find a match).

Use Case

1. An e-mail arrives on a route point, initiating a routing workflow.
2. The e-mail is screened for an account number
3. If a screening rule match is found, the e-mail is provided with an auto-response and then queued to specific queue.
4. If no screening rule match is found, the e-mail is moved to a workbin of the starter agent group.
5. If the e-mail processing or screening itself fails, then interactions are moved to an general interaction queue.

The Screen Interaction block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Common Properties.

Classification Categories Property

Use to select the location to store resulting classification categories after classification.

1. Click under Value to display the button.
2. Click the button to open the Classification Categories dialog box.
3. For Type, select **User Data** or **Variable**.

    - If you select **Variable**, select the variable that contains the key-value pairs to add to the interaction's User Data.
    - If you select **User Data**, specify the key-value pairs to add to the interaction’s User Data.

Key Value Pairs Property

Select the location to store the resulting key-value pairs after screening rules are applied.

1. Click under Value to display the button.
2. Click the button to open the Key-Value Pairs dialog box.
3. For Type, select **User Data** or **Variable**.

    - If you select **User Data**, specify the key-value pairs to add to the interaction’s User Data.
    - If you select **Variable**, select the variable that contains the key-value pairs to add to the interaction's User Data.

Matching Rules Property

Select the location to store the IDs of the matched rules and the keywords that matched after screening rules are applied.

1. Click under Value to display the button.
2. Click the button to open the Matching Rules dialog box.
3. For Type, select **User Data** or **Variable**.
    - If you select **User Data**, you don't need to enter the Value field.
    - If you select **Variable**, select the variable for storing the matched rules and keywords.

Result Type Property

Click the down arrow and select the type of screening result:

- **All**--Select to return screening rule IDs, key-value pairs, and categories.
- **Rules**--Select if you wish to apply all screening rules.
- **Matching Pairs**--Select to return matched pairs of Screening Rule IDs and specific strings of words in the e-mail that matched the screening rules. Classification Server can return specific strings that were matched during the screening process. For example a rule screening for credit card numbers could return the following key-value pair: "Key_credit_number" "1111-2222-3333-4444"
- **Categories**--Select to return only the classification categories associated with the screening rules. This can be used later in the strategy to select a Standard Response.

Classification Server Property

Select the Application name for the Classification Server from those in the Configuration Database. If a Classification Server is not selected, the platform will internally select one.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.
    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Classification Server from the **Value** field. Otherwise:
    - You can select **Literal** and enter the name of the classification server in the **Value** field.
    - You can select **Variable** and select the variable from the **Value** field.

Generate Outports Property

Use to segment interactions to take different paths. Select **true** or **false**. When set to true, Composer will generate one outport for each selected screening rule. If the screening data matches the screening rule, then the processing will continue via the corresponding screening rule.

Language Property

Click the down arrow to select the language of the incoming interaction. The selected language determines which screening rules are shown. You must select a language in order for code to be generated.

1. Click under **Value** to display the button.
2. Click the button to open the Language dialog box.
3. The next step depends on whether you are connected to Configuration Server.

   - If you are connected and want to select a language defined as a Business Attribute in the Configuration Database, select **Configuration Server**. A tree of languages appears in the **Value** field for selection. Otherwise:
   - You can select **Literal** and enter the name of the language in the **Value** field.
   - You can select **Variable** and select the variable that contains the language from the **Value** field.

Root Category Property

Select the name of the top-level category to be used for the classification.

1. Click under Value to display the button.
2. Click the button to open the Root Category dialog box.
3. The next step depends on whether you are connected to Configuration Server.

   - If you are connected, select **Configuration Server** from the Type dropdown menu. A tree of classification categories appears in the **Value** field. Select the name of the top-level (root) classification category. This is a directory that appears in Configuration Manager in the `Business Attributes` > `Category Structure` folder. For more information on Root folders, see the *Universal Routing 8.1 Reference Manual*. Otherwise:
   - You can select **Literal** and enter the name of the root category in the **Value** field.
   - You can select **Variable** and select the variable that contains the root category from the **Value** field.

Screening Data Property

Use this property to select whether to search for screening data in the interaction's User Data, from a variable, or from the UCS database. This property works with the Screening Rules property below.

1. Click under **Value** to display the button.
2. Click the button to open the Screening Data dialog box.

3. For **Type:**

    - Select **UCS** to have Classification Server take the screening data from the UCS Database. Leave the **Value** field empty.
    - Select **User Data** to have Classification Server take screening data from the UCS Database. Leave the **Value** field empty.
    - Select **Variable** to have Classification Server take the screening data from a variable. Select the variable.

Screening Rules Property

Use to specify the screening rules to apply on the screening data specified above.

1. Click under **Value** to display the … button.
2. Click the … button to open the Screening Rules dialog box.
3. Click **Add** to open the Select Items dialog box.
4. From the **Type** dropdown menu, do one of the following:

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

- If you are connected to Configuration Server, select **Configuration Server**. Select one or more screening rules for the Value.

- Select **Literal** and enter the screening rules in the **Value** field.Use commas to separate the screening rules.
- Select **Variable** and select the variable that contains the screening rules from the **Value** field.

Send Email Block

Use this block to send an e-mail waiting in a queue that was previously created using the Create Email block. Note: The Send Email block should only be used if the output queue (Create Email block) property is set. If this property is not set, the Orchestration platform will automatically send the e-mail via an internal Orchestration queue, and therefore the Send Email block is not needed.

Special Note on Cc, From, and Exclude Addresses Properties

The Literal and Variable types can have a value set to an actual e-mail address, e.g., joe@test.com, or refer to the name of a previously configured e-mail address from Configuration Server (e.g., if Tech Support is configured as a Configuration Server `E-mail Accounts` Business Attribute, then Tech Support can be the value for the Literal type and the platform will use that e-mail address). The Send E-mail block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

Email Server Property

Find this property's details under Common Properties.

E-mail to Send Property

Select the variable containing the e-mail to send.

Message Delivery Notification Property

Select true or false to indicate if the message being sent should include a request for a return message indicating whether and how the original message was delivered.

- If one of the SMTP servers involved in the transport of the original e-mail fails to deliver it, the return message will come into the system with Configuration Database Interaction Subtype InboundNDR. It contains no additional information.
- If the original e-mail is successfully delivered, the return message will come into the system with a Configuration Database Interaction Subtype InboundReport. It uses attached data to indicate delivery statuses such as delayed, delivered, relayed, and so on.

For details on Interaction Subtype handling, see the E-mail Server Java: Advanced Topics section of the Ongoing Administration and Other Topics chapter of the *eServices/Multimedia 8.1 User's Guide*.

Message Disposition Notification Property

Select **true** or **false** to indicate if the message should include a request for a return message indicating what happened to the original message after it was delivered. For example, the return message may indicate whether the original message was displayed, printed, deleted without displaying, and so on.

Exceptions Property

Find this property's details under Common Properties.

CC Property

Find this property's details under Common Properties.

Exclude Email Addresses Property

Find this property's details under Common Properties.

From Property

Find this property's details under Common Properties.

Header Fields Property

Use this property to specify additional message headers through a list of key-value pairs of literal values or application variables. Header fields are used to add/overwrite headers to the e-mail message when it is sent. They are passed to E-mail Server and the latter will create the e-mail message with these custom headers and will take the form of <*headername*>:<*headervalue*>. Examples of typical header names are: `Received`, `Return-Path`, `X-MIMETrack`, `Subject`, `Sender`, `From`, `To`, `Cc`, `Bcc` To enter the Header Fields property:

1. Click under Value to display the button.
2. Click the button to open the Header Fields dialog box.
3. Click **Add** to open the Key Value Pairs dialog box.
4. For **Key**, enter a key (see above description).
5. For **Value**, enter the literal value or select a variable for the value. The Header Fields dialog box reflects your entry.

Subject Property

Use this property to override the subject specified in the Create E-mail block. Enter the Subject line of the e-mail.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Send SMS Block

Use this block to send an Short Message Service (SMS) message created with the Create SMS block. This version of Composer supports native SMS, which means that the SMS message is sent via SMS Server and not via an e-mail to SMS gateway.

Note: The Send SMS block should only be used if the interaction queue (Create SMS block) property is set. If this property is not set, the Orchestration platform will automatically send the SMS via an internal Orchestration queue, and therefore the Send SMS block is not needed.

The Send E-mail block has the following properties:

Name Property

Find this property's details under Common Properties.

Block Notes Property

Find this property's details under Common Properties.

SMS Server Property

Select an SMS Server to send the message.

1. Click under Value to display the ... button.
2. Click the ... button to open the Select SMS Server dialog box.
3. The next step depends on whether you are connected to Configuration Server. Otherwise:
    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the SMS Server object from the **Value** field.
    - You can select **Literal** and enter the name of the server in the **Value** field.
    - You can select **Variable** and select the variable from the **Value** field that contains the name of the server.

SMS to Send Property

Select the variable containing the message to send.

Exceptions Property

Find this property's details under Common Properties.

Message Source Number Property

Use this property to override the Message Source Number specified in the Create SMS Block, which is the mobile telephone number from which the SMS message should appear to come.

1. Click under **Value** to display the button.
2. Click the button to open the Select the SMS Message Source Number dialog box.
3. From the **Type** dropdown menu, do one of the following:

    - If you are Server connected to Configuration Server, select **Configuration Server**. Select a Configuration Server Business Attribute for **Value**.
    - Select **Literal** and enter the Cc address in the **Value** field.
    - Select **Variable** and select the variable that contains the number from the **Value** field.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Update Contact Block

Note: This block is hidden from the eServices palette by default. To make the block visible, right-click on the eServices palette title bar and select Customize from the Palette Group menu. A dialog box opens where you can de-select Hide. Use this block to update customer profile information in the Universal Contact Server Database, based on data attached to an interaction. You may specify certain information to update by specifying certain key-value pairs that represent the data to update. If certain parameters are not specified, then all contact attributes will be updated by default, based on all user data available. Important! See Mandatory User Data For UCS Blocks. Also see the section on Contact Identification and Creation in the *eServices 8.1 User's Guide*.

Use Case

1. A customer sends an e-mail to the contact center, initiating a routing workflow.
2. The routing workflow reads the user data from the e-mail.
3. The user data is used to identify that this customer is an existing contact in the UCS database based on the customer's first and last name
4. The routing workflow does a comparison of the customer's e-mail address in the e-mail, with the one in UCS database for this contact. They are different.
5. The routing workflow updates the contact information in the UCS database with the new email address

The Update Contact block has the following properties:

Name Property

Find this property's details under Common Properties.

Exceptions Property

Find this property's details under Context Services Exception Events. Also see Exception Events for eServices UCS Blocks where the following exceptions are supported: 201, 203, 502, 510, 701, 710, 716, 730.

Interaction ID Property

Find this property's details under Common Properties.

Contact Attributes Property

Use this property to set the new contact attribute values.

1. Click under **Value** to display the button.
2. Click the button to open the Contact Attributes dialog box.
3. Click **Add** to open Configure Contact Attributes dialog box where you specify the attributes.
4. Click the down arrow opposite Name and select an attribute. You may select a user-defined attribute or a predefined attribute, such as:
    - **Contact ID**
    - **Customer ID**
    - **Customer Segment**
    - **E-mail Address**
    - **First Name**
    - **Last Name**
    - **Phone Number**
    - **PIN**
    - **Title**
5. Click the down arrow opposite **Type** and select **Literal** or **Variable**.
6. If you selected **Literal**, enter the **Value** field and click **OK**.
7. If you selected **Variable,** select the variable that contains the value and click**OK**. The Name and Value fields in the Configure Attributes dialog box reflect your entries.
8. Click **Add** again to continue entering customer attributes in this fashion.

`Tenant Property` Find this property's details under Common Properties.

Universal Contact Server Property

Find this property's details under Common Properties.

Use Contract Attributes Property

- Select **true** to have the new contact attribute values taken from the Contact Attributes property.
- Select **false** to have the new contact attributes values taken from the interaction’s User Data.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Using eServices Blocks

This page contains general information on working with eServices blocks.

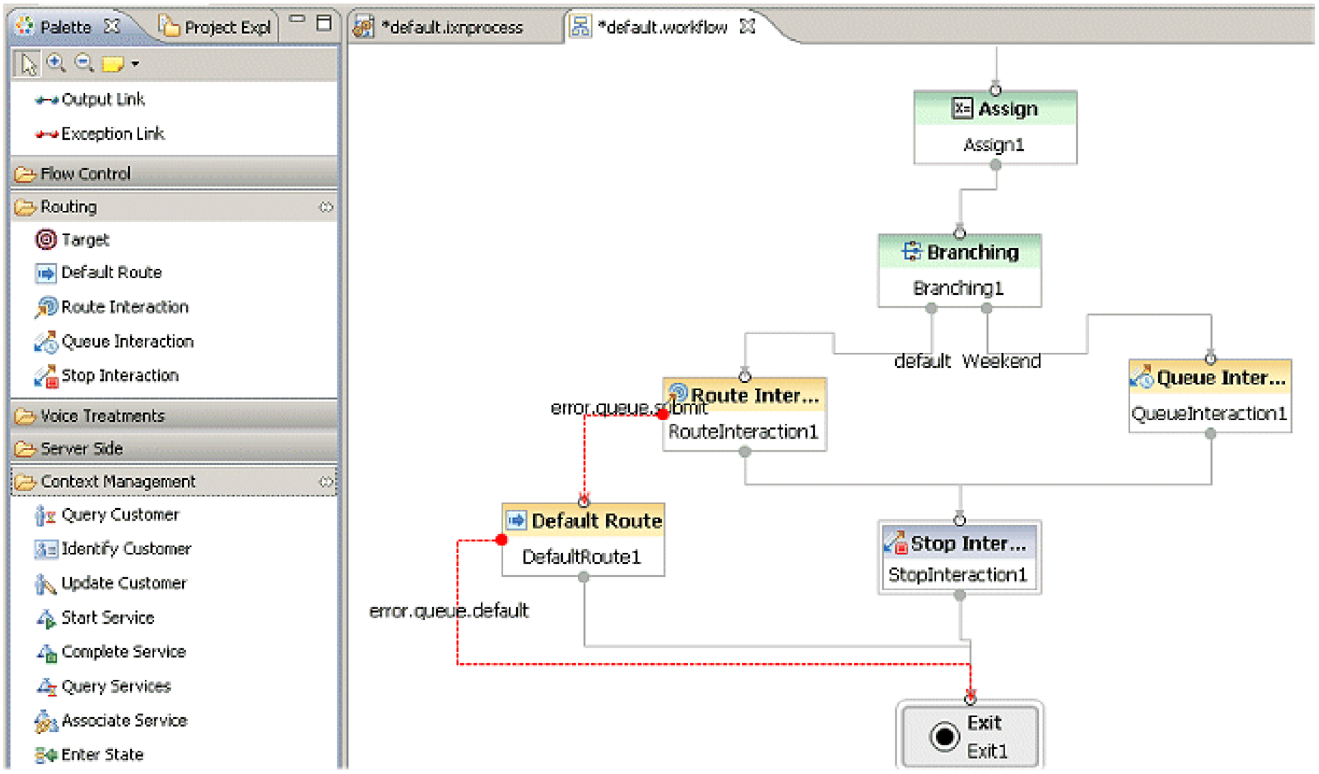
Example Multimedia Workflow

In general, an interaction process diagram (IPD) for multimedia interactions works like this:

1. A media server (such as E-mail Server Java) directs Interaction Server to place an interaction into an inbound interaction queue.
2. Using the Interaction Queue View property, the interaction is then taken out of the queue and submitted to a routing workflow.
3. The workflow performs specialized processing and eventually routes the interaction to a target, but not necessarily the final target. For example, an e-mail interaction may be placed in an agent queue for construction of a response.
4. The target processes the interaction and places it into another queue where another workflow may process it. For example, a workflow may send an agent’s draft e-mail response to a queue for Quality Assurance checking.
5. The cycle of going from queue/view/workflow continues until processing is stopped or the interaction reaches some final (usually an outbound) queue.

Example Diagram

The figure below shows a sample multimedia workflow diagram in Composer Design perspective.



For other sample diagrams, see the Sample Applications topic. The default.workflow shown above works as follows:

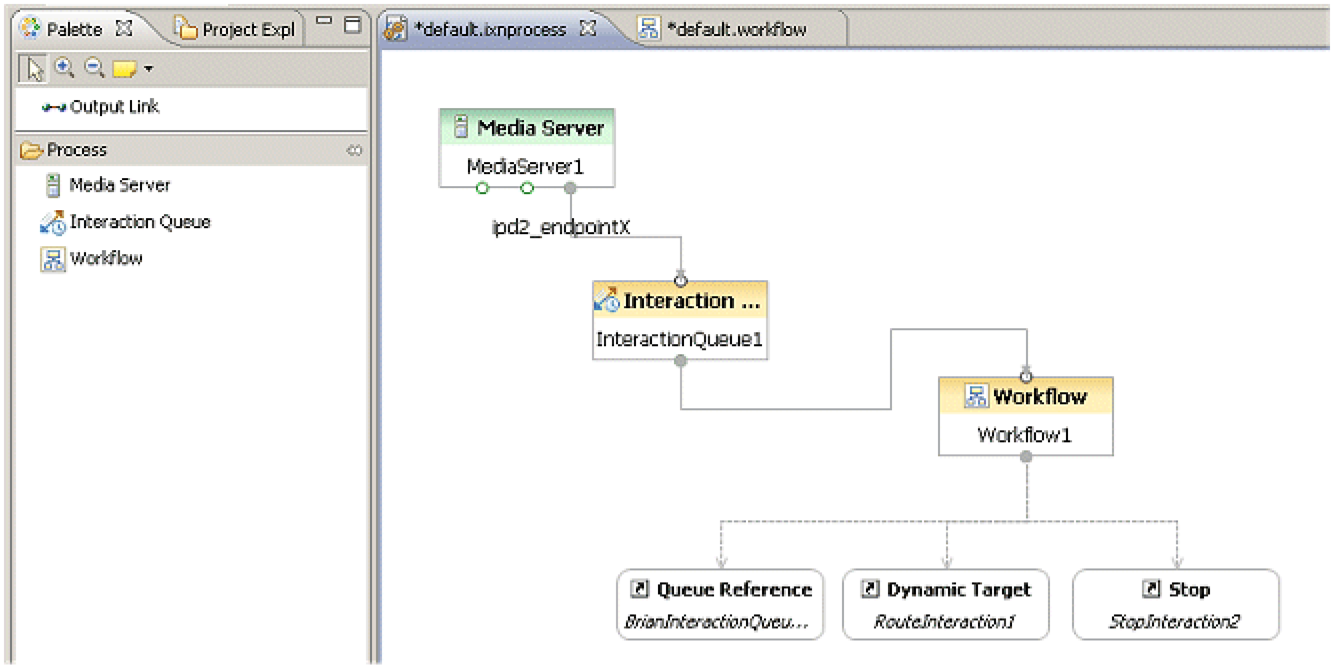
- The Entry block defines a variable called Today.
- The Assign block gives a value to the Today variable. In Expression Builder this is defined as:

```
data.Today==(_genesys.session.day.Saturday)||(data.Today==_genesys.session.day.Sunday)
```

- The Branching block Conditions property contains an expression used for segmenting interactions based on whether today is a week day or the weekend. The expression determines whether an interaction goes to a queue for the weekend crew or whether it is routed to a target.
- The Stop block notifies Interaction Server to stop processing and whether to notify Universal Contact Server about the interaction.

Associated IPD

The figure below shows the IPD containing the Workflow block that points to the workflow diagram above. Three work-flow generated blocks are automatically generated in this example.



eService Preparation

Genesys eService/Multimedia lets you process non-voice interactions in your contact center. At the center of the collection of components are:

- Interaction Server: Works with Universal Routing Server and Stat Server to process non-voice interactions by executing interaction process diagrams.
- Universal Contact Server (UCS): Works with its database to deliver customer contact history and information to the agent desktop.
- In addition, there are a number of other Multimedia servers that facilitate the handling of non-voice media, including E-mail Server Java, SMS Server, and Chat Server.

This help system assumes you have already installed and configured the eService/Multimedia components as described in the *eService/Multimedia 8.1 Deployment Guide*.

Working with Returned Data

A few Composer Route blocks will return data back to the application:

- Email Response (Output Result)
- Create Email (Output Result)
- Create SMS (Output Result)

- Identify Contact (Result Property)
- Render Message (Result Property)
- Query Customer (Result Property)

Each qualifying block will expose a output result property (or equivalent) that will specify an application variable to store the results. These results will then be available in other blocks in the application for further processing. The format of returned data is usually JSON. Any post-processing work to be done on returned results can be done in the existing Assign block. It provides access to ECMAScript functions and supports writing simple or complex expressions to extract values out of JSON strings and arrays.

Mandatory User Data for UCS Blocks

When working with the Update Contact and Render Message blocks (which map to Universal Contact Server services), certain properties must exist in the interaction User Data. These properties are:
Update Contact Block

- ContactId

Render Message Block

- ContactId (if some contact-related Field Codes (as described in the *eServices 8.1 User's Guide*) are used in the text to render)
- InteractionId (if some interaction-related Field Codes are used in the text to render)
- OwnerEmployeeId (if some agent-related Field Codes are used in the text to render)

As is the case with IRD, these properties are not set in the blocks themselves. Instead, the properties are assumed to be put in the interaction's User Data by some other block earlier in the workflow, such as the Identify Contact block or Create Interaction block with the Update User Data property set to true. In case no other block does this, the User Data block may be used for this purpose. Note: If those properties are not available, an explicit UCS error message (missing parameter) shows in the Orchestration Server log.

Common Properties for Workflow Blocks

The following properties are common to multiple blocks. Their descriptions are placed here to minimize duplication of content:

Block Notes Property

Can be used for both callflow and workflow blocks to add comments. When migrating IRD strategies into Composer, this property is the equivalent of the IRD Object Comments feature. For the IRD objects migrated into Composer blocks, Composer supplies the note text by combining the IRD equivalent object name plus the original comments from the IRD object.

Categories Property

Use this property to select individual categories and sub-categories of the Root Category Property to be used in the classification process.

1. Click under **Value** to display the button.
2. Click the button to open the Categories dialog box.
3. Click **Add** to open the Select Items dialog box.
4. From the **Type** dropdown menu, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server**. Select one or more categories for the **Value**.
    - Select **Literal** and enter the categories in the Value field. Use commas to separate the categories.
    - Select **Variable** and select the variable that contains the categories from the Value field.

CC Property

Use this property to specify the Cc addresses on the e-mail transmitting the chat transcript. Also see Special Note below.

1. Click under **Value** to display the button.
2. Click the button to open the Select Cc Addresses dialog box.
3. Click **Add** to open the Select Items dialog box.
4. From the **Type** dropdown menu, do one of the following:

- If you are connected to Configuration Server, select **Configuration Server**. Select a Configuration Server `E-mail Accounts` Business Attribute for the Value.
- Select **Literal** and enter the Cc address in the **Value** field.
- Select **Variable** and select the variable that contains the Cc address from the **Value** field.

5. If applicable, repeat the above steps to add another e-mail address.
6. Click **OK** to close the dialog box.

Condition Property

The Condition property indicates that the log will be active only if the given condition is true at runtime. To provide a condition setting for a log:

1. Select the **Condition** row in the block's property table.
2. Type the condition to evaluate against.

For example, assume in Entry block, there is a variable "MyVar==3". Assume also that you would like to log the session ID (GVPSessionID variable in Entry block) for all sessions where MyVar=3. In this case you must set the condition to "AppState.MyVar=3". If this condition is true, then GVPSessionID will be written to the log, otherwise it will be ignored.

Destination Property

Use this property to specify the routing destination.

1. Click under Value to display the button.
2. Click the button to open the Destination dialog box.
3. Select one of the following:

- **Block Reference**. For Value, specify any Target or Route Interaction block in the diagram where the Route property is set to False. When Destination is set to a Route Interaction block, the block-generated SCXML code automatically uses the following Route Interaction block properties: Queue For Existing Interaction and Queue For Outgoing Interaction.
- **Literal.** For Value, you can specify:
    - An agent: <agent id>
    - A place: <place id>
    - A DN: <number>
    - An e-mail address: <username>@<host> or _origin or _origin.all or _udata
    - A customer number: <dn number>

    - A target format addresses: <Target DN>

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Variable**. If the variable contains a string, see Literal above. If the value is a JSON object, Value can refer to:
    - An agent: {agent: "<agent id>", type:"A"}
    - An agent group: {agent: "<name>", type:"AG"}
    - A place: {place: "<place id>", type:"AP"}
    - A place group: {place: "<name>", type:"PG"}
    - A DN: {dn: "<number>", type:"Q or RP or DN", switch:"<switch name>"}
    - An interaction queue: {id: "<q name>", type:"iq" }
    - A workbin: {id: "<wb name>", type:"wb"<owner>"}
    - A customer number: {dn: "<number>"}
    - A target format addresses: Resource Object from the queue.submit.done event (the Target Block Selected Property Resource Selected property).

See the Orchestration Server Documentation Wiki for those literals that apply to multimedia interactions only.

- **Configuration Server** to select the from Switch//DN if Server connected.
- **Resource** to select a resource using properties that will form a JSON object.

See the Orchestration Server Documentation Wiki

4. Click OK to close the Destination dialog box.

Detach Property

Use for multi-site routing. Controls whether the Orchestration Platform should <detach> an interaction from the current session before sending the e-mail with the chat transcript. When this property is set to true, the interaction is detached from the current session. The Chat Transcript block will always <detach> the new interaction before continuing with the current interaction.

Detach Timeout Property

Use to specify how long to attempt to <detach> if an initial attempt fails with an invalidstate error. Specify the timeout in milliseconds. If set to 0, no further attempt to detach is made. After the timeout, if the <detach> is not successful, no further attempts will be made and the block will attempt to reclaim the interaction back into the current session using <attach>.

Do Not Thread Property

Select **true** to instruct NOT to thread under another interaction (which is specified under key `ParentID` in the User Data) in the contact's history in the Universal Contact Server Database.

Email Server Property

Note: This property is not mandatory as the platform will choose an e-mail server if not provided. Select the Application name for the E-mail Server that URS should notify about the e-mail (via Interaction Server).

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.

- If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the E-mail Server Java object from the **Value** field.
- You can also select **Literal** and enter the name of the e-mail server in the **Value** field.
- You can also select **Variable** and select the variable from the **Value** field.

Enable Status Property

This property controls whether or not a block contributes code to the application. Diagrams visually indicate when a block is disabled. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later. You can also right-click a block and select **Toggle Enable Status**. The ORS Debugger skips over deactivated blocks.

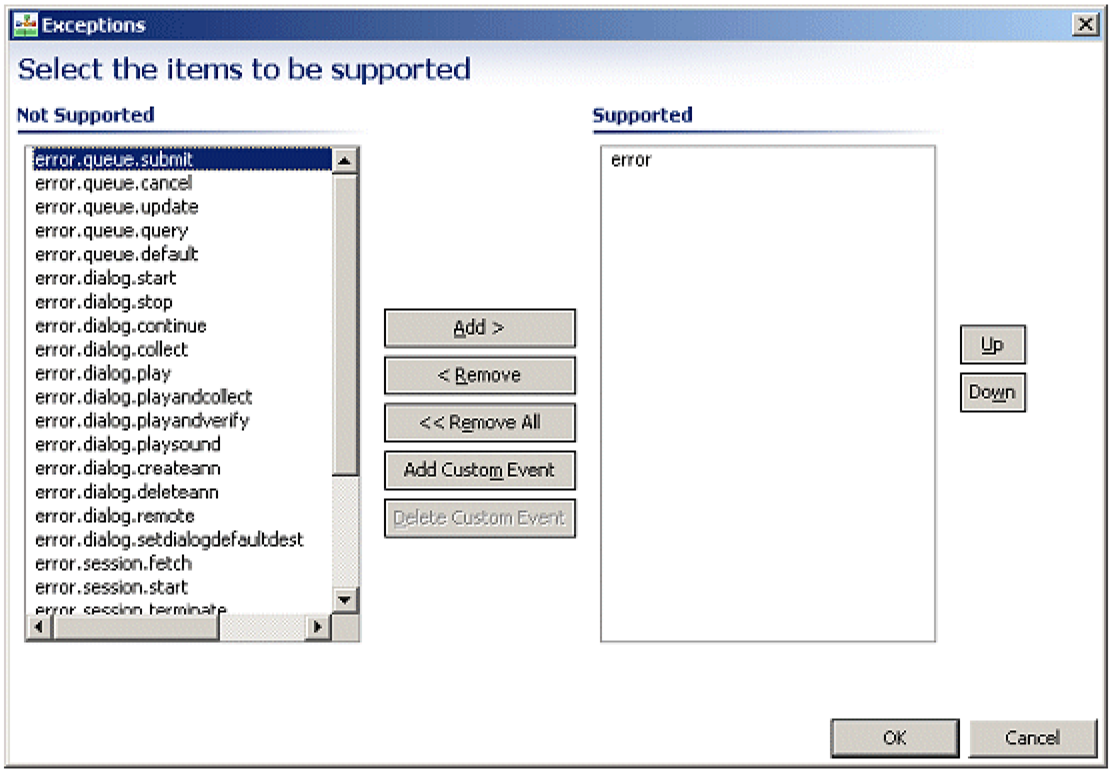
Exceptions Property

Use this property to define which exception events the block is designed to handle. These are SCXML events that are either thrown by the interpreter, or generated in response to a caller action.

Note: A catch handler called all has been added to catch all exception events.

To handle (support) a specific event:

1. Click the **Exceptions** row in the block's property table.
2. Click the button under **Value** to open the Exceptions dialog box. The Exceptions dialog box for the Entry block is shown below as an example. Each block will have its own list of exceptions.



error.queue.submit
error.queue.cancel
error.queue.update
error.queue.query
error.queue.default
error.dialog.start
error.dialog.stop
error.dialog.continue
error.dialog.collect
error.dialog.play
error.dialog.playandcollect
error.dialog.playandverify
error.dialog.playsound
error.dialog.createann
error.dialog.deleteann
error.dialog.remote
error.dialog.setdialogdefaultdest
error.session.fetch
error.session.start
error.session.terminate

error

3. From the list of events on the **Not Supported** pane, select the event that you want to handle.
4. Click the **Add** button to move the event to the **Supported** pane.

To explicitly not handle (not support) a specific event marked as supported:

1. Click the **Exceptions** row in the block's property table.
2. Click the ⋯ button to open the Exceptions dialog box.
3. From the list of events on the **Supported** pane, select the event that you do not want to handle.
4. Click the **Remove** or **Remove All** button to move the event (or all events) to the **Not Supported** pane.
5. Optional. You can also select Add Custom Event.

To rearrange (reorder) the sequence of events on the **Supported** pane:

1. Click the **Exceptions** row in the block's property table.
2. Click the ⋯ button to open the Exceptions dialog box.
3. From the list of events on the **Supported** pane, select an event that you want to rearrange.
4. Do one of the following:
   - To move the event higher in the sequence, click the **Up** button.
   - To move the event lower in the sequence, click the **Down** button.

Notes:

- Exceptions for Busy treatment blocks should be handled in the Target block to which they are connected and not in the Busy treatment blocks themselves. Busy treatment exceptions are raised as the `error.queue.submit` exception and not as exceptions listed in individual treatment blocks.
- Each block has its own predefined set of events on the Exceptions property dialog box. Genesys recommends that you not remove any of the predefined events from the **Supported** list.
- Before generating code, each supported event must be handled by connecting its red node on the side of the block to the inport (input node) of another block.
- The events in the Entry block are global in scope.
- Events defined in other blocks are local to that block only. When an event is thrown, if a handler for that event is declared in the current block, that local event handler is called.
- If there is no local event handler for the event, but there is a global event handler declared in the Entry block, then the global event handler from the Entry block is called.

Enable Status Property

This property controls whether or not a block contributes code to the application. You may wish to use this property if there is a need to temporarily remove a block during debugging or, for other reasons during development, temporarily disable a block. This saves the effort of having to remove the block and then add it back later. You can also right-click a block and select **Toggle Enabled Status**.

Exclude Email Addresses Property

When sending a response, you may not want the email to copy in all the addresses (To, From) in the original email. Use this property to exclude specific email addresses that need to be removed from the “To” and “Cc” fields.

1. Click under **Value** to display the button.
2. Click the button to open the Exclude Addresses dialog box.
3. Click Add to open the Select Items dialog box.
4. From the **Type** dropdown menu, do one of the following:
    - Select **Literal** and enter the address to exclude in the **Value** field.
    - If you are Server connected to Configuration Server, select **Configuration Server**. Select an E-mail Accounts Business Attribute for the Value.
    - Select **Variable** and select the variable that contains the address to exclude from the **Value** field.
5. If applicable, repeat the above steps to add another e-mail address.
6. Click OK to close the dialog box.

Field Codes Property

When using a standard response to render message text, use this property to assign values to Field Code variables that have been defined in Knowledge Manager (as described in the *eServices 8.1 User's Guide*) and used in that standard response. Universal Contact Server requires values for Field Codes when using standard responses that include Field Codes.

1. Click under **Value** to display the button.
2. Click the button to open the Field Codes dialog box.
3. Click **Add**. A second dialog box opens for specifying Field Codes and values.
4. Type the name of the Field Code.
5. Select **Literal** or **Variable**.
    - If you select **Literal**, enter the value for the Field Code.
    - If you select **Variable**, select the variable the contains the Field Code Value.

From Property

Use this property to specify the address to appear in the "From" field of the outbound e-mail.

1. Click under **Value** to display the button.
2. Click the button to open the Select From E-mail Address dialog box.
3. From the **Type** dropdown, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the from address from the **Value** field in the form of an Configuration Server E-mail Accounts Business Attribute.
    - Select **Literal** from the dropdown menu and then enter the From address in the **Value** field.
    - Select **Variable** from the dropdown menu and then select the variable that contains the from address from the **Value** field.
4. Click **OK** to close the dialog box.

Include Original Message Into Reply Property

Select **true** or **false** to indicate if the text from the parent interaction copied into the forwarded e-mail.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId.

Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction.

Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS.

With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all.

For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case.

For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Logging Details Property

Logging details contains the expression that will be logged at runtime by the ORS platform. If logging details are specified, then logging is generated for the block; if no logging details are specified, no logging is generated.

To create logging details:

1. Click the **Logging Details** row in the block's property table.
2. Click the ⋯ button to open the Logging Details dialog box.
3. In the Logging Details dialog box, click **Add** to open Expression Builder.
4. Create an expression to be used for logging details, such as an expression based on the variables whose content you wish to log.

Log Level Property

To assign a value to the Log Level property:

1. Select the **Log Level** row in the block's property table.
2. In the **Value** field, select one of the following from the drop-down list:
    - **Project Default**. The block uses the project's default log level, which can be configured through the project properties.
    - **Info**. This is an Informational level to log application-specific data.
    - **Debug**. This is a Debug level used for application debugging.
    - **Error**. This is an Error level to log error details.
    - **Warn**. This is a Warning level to flag any application warnings.
    - **Alarm**. This is an Alarm level to send the message as an alarm to the Genesys management framework.

Language Property

To set the active language:

1. Select the **Language** row in the block's property table.
2. Click under **Value** to display a down arrow.
3. Select one of the following languages:
    - **English (US)**
    - **Spanish**
    - **Mandarin**
    - **Cantonese**
    - **Vietnamese**
    - **French**
    - **French (Canada)**
    - **German**
    - **Italian**
    - **Japanese**
    - **Korean**
    - **Russian**

Name Property

The Name property is present in all blocks in Composer. The Name property is the first property for all blocks. Use the Value field in the **Name** property row of the block's property table to name the block.

- Block names should conform to ECMAScript and SCXML identifier naming conventions. There is no limit to the maximum number of characters.
- Names may consist only of numbers, letters, or initial underscores (_).
- Names should only begin with a letter or underscore.
- Except for the Entry and Exit blocks, you should give all blocks a descriptive name. For example, if an Input block asks the caller to input an account number, then the name of the block could be Input_Account_Number.
- The name of the block is used as the “Name” of the <form> tag that gets generated for that block.

To provide a name for a block:

1. Select the **Name** row in the block's property table.
2. In the **Value** field, type a block name that conforms to the restrictions above.

Output Queue Property

Select the output queue for the new interaction.

Only interaction queues that were created in the current Composer Project are shown for the Configuration Server values. The interaction queues are sorted per parent Interaction Process Diagram. To define the output queue:

1. Click under **Value** to display the button.
2. Click the button to open the Select an Output Queue dialog box.
3. Click the **Type** down arrow and do one of the following
    - If you are connected to Configuration Server, select **Configuration Server** and then select an output queue as the **Value**. The **Independent Objects** > **Same queue** choice allows you to put the outbound interaction in the same interaction queue that initiated the current interaction.
    - Select **Literal** and then enter the name of the output queue as the Value.
4. Click **OK**.

Output Result Property

Use this property to specify a variable where the new interaction will be saved. The results will then be available in other blocks in the application for further processing.

The format of returned data is JSON. Any post processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It already supports writing simple or complex expressions to extract values out of JSON strings and arrays.

Prompts Property

Use the Prompts property to specify the audio prompts that are played to the caller in a Play Message or User Input block. You can specify prerecorded prompts, text, and several standard data types.

To add, delete, or arrange prompts:

1. Click the **Prompts** row in the block's property table.
2. Click the ▦ button to open the Prompts dialog box.
3. Click **Add** to enable **Prompt Type**, **Interruptible**, and **Value** fields.
4. From the **Type** drop-down list, select a Type: **Announcement**, **RecordedAnnouncement**, **FormattedDigits**, or **Text**. See table below.
5. **Interruptible**—Select **true** or **false**. Indicates whether the caller can interrupt the message. Instructs URS to send a message to T-Server indicating that the announcement is interruptible.
6. **Value***—Enter data for the selected **Type**.

Place the audio files in the `Resources\Prompts\{APP_LANGUAGE}` folder under the Java Composer Project. Audio files can be added to the project by copying and pasting from the Windows file system into the Java Voice Project in the Project Explorer.

Note: By default, Genesys supplies `.vox` files only for `mulaw 8Khz`. If you are using any other audio format for playback of audio files, replace the files with the corresponding audio files in the required audio format.

Standard Response Property

Use this property to select the text from your Standard Response library. You can enter either a category code or a standard response identifier as described in the "Creating Standard Responses" section in the "Genesys Knowledge Management: Basics" chapter of the *eServices/Multimedia 8.1 User's Guide*.

1. Click under **Value** to display the ▦ button.
2. Click the ▦ button to open the Select a Standard Response dialog box, which organizes standard responses by categories.
3. From the **Type** dropdown menu, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server**. Then select the standard response identifier or category code from the **Value** field.
    - Select **Literal**. Then for **Value** enter the standard response identifier or category code. See the variable field below for more information.
    - Select **Variable.** Then for **Value**, select a variable containing one of the following:
        - A category code from a variable whose value is set to an existing category code in the form "'`gdata:config\\CA.<id>`'", where <id> is the category ID.   Example: `msgsrc=`

gdata:config\\CA.00005a5FS3GW005G

- A standard response identifier in the form "'gdata:config\\SR.<id>'", where <id> is the standard response identifier. Example: msgsrc= gdata:config\\SR.00005a5FS3GW005A

4. Click **OK** to close the dialog box.

Subject Property

This property is enabled if the **Use Subject From SRL** property is set to **false**. Enter the subject to appear in the Subject field of the outbound interaction.

Tenant Property

This mandatory property is set by default to Variable(TenantID), whose value is assigned by Orchestration Server. To override this value, click the button to open the Tenant dialog box where you can select another variable to contain the TenantID or enter the value as a literal.

Note: UCS will fail executing the requested task if passed the Tenant_Name instead of the TenantID.

To Property

Use this property to specify the address to which the interaction is to be sent. See the Getting Using Email Addresses topic for information on getting the e-mail address of a customer using Context Services blocks and using it in the **To** property. Also see the Note below.

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Select To E-mail Address dialog box.
3. From the **Type** dropdown menu, do one of the following:
   - If you are connected to Configuration Server, select **Configuration Server**. Select the to address from the **Value** field in the form of an Configuration Server E-mail Accounts Business Attribute.
   - Select **Literal** and enter the To address in the **Value** field.
   - Select **User Data**.
4. Click **OK** to close the dialog box.

Universal Contact Server Property

Specify the Universal Contact Server to use.

1. Click under Value to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.

    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Universal Contact Server object from the **Value** field. If not connected:
    - Select **Literal** and enter the name of the Universal Contact Server in the **Value** field.
    - Or select **Variable** and select the variable from the **Value** field.

Update Interaction User Data Property

Select **true** to have the UCS returned values be attached to the interaction User Data in the form of key-value pairs; otherwise select **false**.

When set to **true**, the following user data key-values are added to this interaction's user data:

- `ContactCreated`: true or false.
- `NumberOfContactsFound`: Number of contacts identified with the given User Data.
- `ContactIdList`: A string or an array of strings with the list of matching contact IDs. Note: if the **Return Unique** property is set to true, `ContactIdList` is not returned if multiple contacts are identified.

Update User Data Property

Select **true** to have the contact attribute values returned by the Universal Contact Server be part of the User Data of the response.

Select **false** to have the contact attribute values returned by the Universal Contact Server be part of the parameters of the response.

Universal Contact Server returns contact attribute values only when a unique contact is found/created. Also note the following:

- If a unique contact is identified or created and this property is false, the contact attribute values will also be returned in the parameter part of the ESP response and will be added in the interaction's User Data as well.
- When this property is set to **true** AND if a unique contact is identified or created, Universal Contact Server returns the contact attribute values in the User Data part of its response. The User Data part of the response is automatically added to the interaction's User Data.

- If this property is set to **false** AND a unique contact is identified/created, the contact attribute values are passed back to the Orchestration platform in the parameter part of the response. In that case, you might choose to add them to the interaction's User Data by setting the Update Interaction User Data property.
- Regardless of the value of the Update User Data property, Universal Contact Server ALWAYS returns the contact ID in the User Data part of its response to Orchestration Server when a unique contact is identified/created.

Uri Property

The Uri property specifies the Location of the Method or File depending on the value of the **Type** property.

To set a URL destination for the Uri property (Type property is set to URL):

1. Select the **Uri** row in the block's property table.
2. In the **Value** field, type a valid URL, or select a Property variable from the drop-down list.

To set a Project destination for the Uri property (Type property is set to **ProjectFile**):

1. Click the **Uri** row in the block's property table.
2. Click the ⋯ button to open the Uri dialog box.
3. Select a workflow in the list.
4. Click **OK** to close the dialog box.

Use Subject From SRL Property

Select **true** or **false** to indicate whether to get the e-mail Subject from the Standard Response Library.

Variables Mapping Property

Use this property to map individual contact attribute values to variables.

Note:

- Contact attribute values are returned only when a unique contact is identified.
- Only primary contact attribute values are returned.

To use variables mapping, open the Variables Mapping dialog box.

Variables Project and Workflow

This page discusses Project, Workflow, and internal variables.

Types of Variables

You have the option of defining two types of variables:

1. Project variables in the Project Variables dialog box, which opens when you click the Access Project Variables button on the toolbar with the IPD in focus. Use Project variables when you need to share information across different workflows. Once defined, Project variables are accessible for use in expressions in Expression Builder.

2. Workflow variables in the Entry block. Use workflow variables when you need to share information across different blocks in the same workflow. For example, the Assign block allows you to assign entered values or values created in Expression Builder to variables. Once defined, workflow variables are accessible for use in expressions in Expression Builder.

As can be seen in the Entry block Applications Variables dialog box, the workflow variables subtypes are as follows

- **System**—Pre-defined system variables hold Project and application-related values. See Default Routing System Variables below. You cannot delete system variables, but applications can modify their values.

- **User**—User-defined custom variables that you create by clicking the Add button in the Application Variables dialog box shown below and selecting User. Applications can delete and modify these types of variables.

- **Input**—Variables which are supplied as input to the called diagram. Created by clicking the Add button in the Application Variables dialog box shown below and selecting User. During runtime, Input variables get automatically filled from the calling context. Typically Input variables are created on the Subworkflow side to notify the Main Workflow about the Parameter passing details while designing the Application flow. Composer does auto-synchronization of the Input variables in the Subroutine block and also in the Play Application block (CTI).

Workflow Variables

The following block properties can be specified as workflow variables:

- Target Block properties: Statistic, Timeout, Target Name (if type = Variable), Target Component Selected, Target Object Selected, Target Selected, Virtual Queue Selected, Virtual Queue
- Play Application properties: Resource
- Play Sound Block properties: Resource, Duration
- Play Message Block properties: Prompts > Values field in Prompts dialog box

- User Input Block properties: Prompts > Values field in Prompts dialog box. AbortDigits, BackspaceDigits, Collected Digits Variable, IgnoreDigits, Number of Digits, Termination Digits, ResetDigits, Resource, StartTimeout, DigitTimeout, TotalTimeout, Verification Attempts, Verification Data
- Set Default Route Block: Destination property
- Route Interaction Block: Statistics property
- Subroutine Block: Parameters property
- Stop Interaction Block Reason to Stop Interaction Property
- Context Services: All blocks have certain properties that allow you to select a variable.
- eService: All blocks have certain properties that allow you to select a variable.

Upgrading from Composer 8.0.2 or earlier

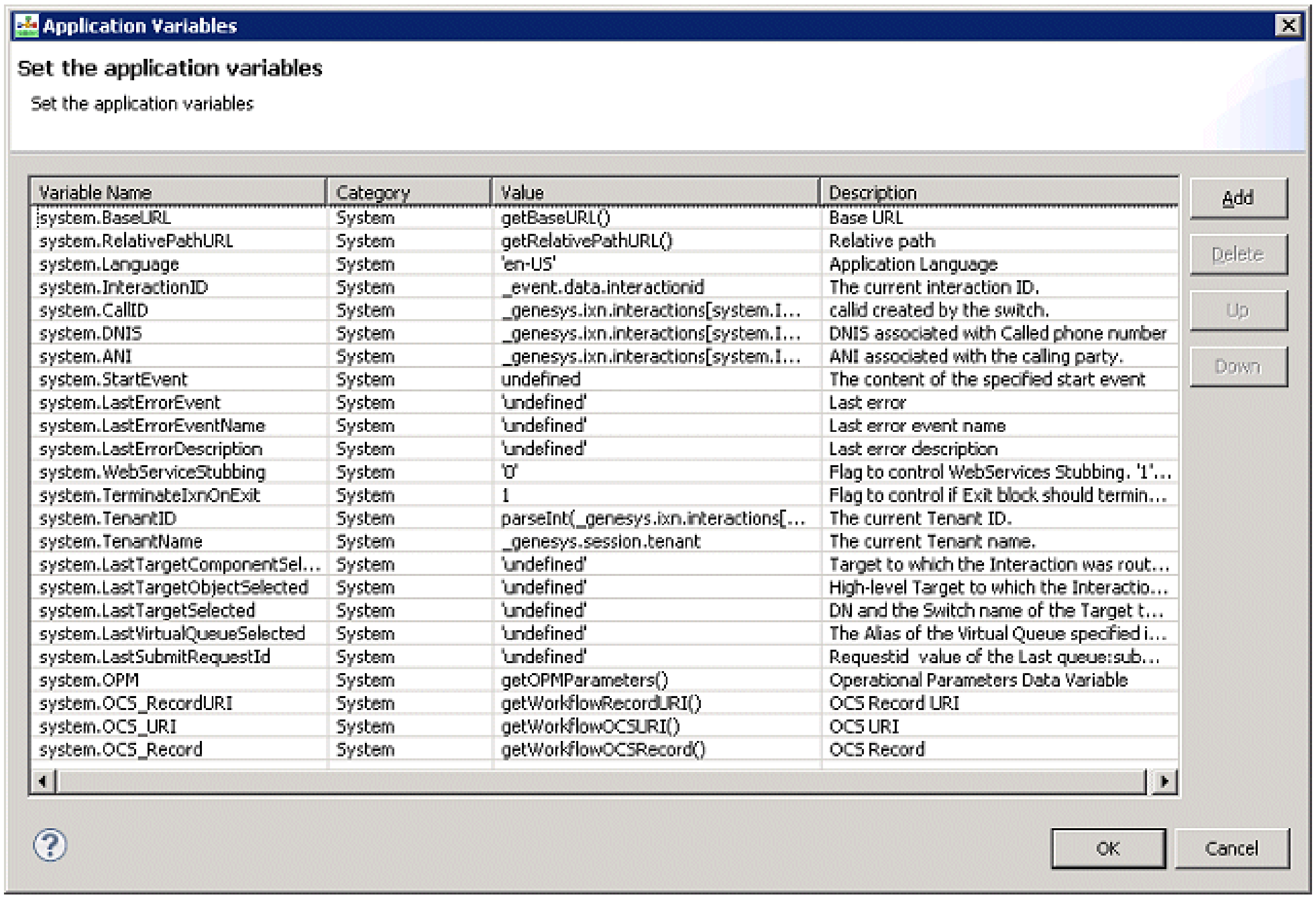
Prior to 8.0.3 release, Composer defined workflow variables in the data model of the SCXML application so they were required to be accessed by prefixing the name of a workflow variable with "_data.". For example, if you defined a workflow variable named var1, you would access it as _data.var1. Starting with 8.0.3, Composer defined these variables in the ECMAScript scope so the variable is accessed simply as var1.

Application Variables Dialog Box

Note: When using the ORS Debugger, are not displayed correctly in the variables view toolbar if the value contains XML or variables that are of type E4X.

To view workflow variables:

1. In the Properties tab for the Entry block, click opposite Variables under Value to display the button. Or use the toolbar button.
2. Click the button to open a dialog box for defining and initializing variables for the workflow.



Variable Name	Category	Value	Description
system.BaseURL	System	getBaseURL()	Base URL
system.RelativePathURL	System	getRelativePathURL()	Relative path
system.Language	System	'en-US'	Application Language
system.InteractionID	System	_event.data.interactionid	The current interaction ID.
system.CallID	System	_genesys.ixn.interactions[system.I...	callid created by the switch.
system.DNIS	System	_genesys.ixn.interactions[system.I...	DNIS associated with Called phone number
system.ANI	System	_genesys.ixn.interactions[system.I...	ANI associated with the calling party.
system.StartEvent	System	undefined	The content of the specified start event
system.LastErrorEvent	System	'undefined'	Last error
system.LastErrorEventName	System	'undefined'	Last error event name
system.LastErrorDescription	System	'undefined'	Last error description
system.WebServiceStubbing	System	'0'	Flag to control WebServices Stubbing. '1'...
system.TerminateIxnOnExit	System	1	Flag to control if Exit block should termin...
system.TenantID	System	parseInt(_genesys.ixn.interactions[...	The current Tenant ID.
system.TenantName	System	_genesys.session.tenant	The current Tenant name.
system.LastTargetComponentSel...	System	'undefined'	Target to which the Interaction was rout...
system.LastTargetObjectSelected	System	'undefined'	High-level Target to which the Interactio...
system.LastTargetSelected	System	'undefined'	DN and the Switch name of the Target t...
system.LastVirtualQueueSelected	System	'undefined'	The Alias of the Virtual Queue specified i...
system.LastSubmitRequestId	System	'undefined'	Requestid value of the Last queue:sub...
system.OPM	System	getOPMParameters()	Operational Parameters Data Variable
system.OCS_RecordURI	System	getWorkflowRecordURI()	OCS Record URI
system.OCS_URI	System	getWorkflowOCSURI()	OCS URI
system.OCS_Record	System	getWorkflowOCSRecord()	OCS Record

To add a new variable in the Application Variables dialog box:

1. Click **Add**. Composer add a row for variable and generates a temporary name and number; for example: var7.
2. Select the row and supply the **Name**, **Type**, **Value**, and **Description** fields.
3. Click **OK**.

System Variables

- **system.Language**—Holds the application language setting. The value should be the RFC 3066 language tag of an installed language pack. Examples of valid RFC 3066 language tags include en-US and fr-FR. This setting also acts as a default language for the application.
- **system.CallID**—Call identifier created by the switch. It is initialized from `_genesys.ixn.interactions[system.InteractionID].voice.callid` (voice only).
- **system.DNIS**—Number that the caller dialed. It is initialized from `_genesys.ixn.interactions[system.InteractionID].voice.dnis` (voice only).
- **system.ANI**—Caller's phone number. It is initialized from `_genesys.ixn.interactions[system.InteractionID].voice.ani` (voice only).

- **system.LastErrorEvent**—Stores the last error that was handled in a block.
- **system.LastErrorEventName**—Stores the name of the error that was handled in a block.
- **system.LastErrorDescription**—Stores the description of the last error that was handled in a block.
- **system.WebServiceStubbing**— Flag to control Web Services Stubbing (1 = ON).
- **system.TerminateIxnOnExit**—Used to automatically stop an interaction that was not stopped by the Route Interaction, Queue Interaction, or Stop Interaction block in a multimedia workflow. New workflow entry blocks have this variable pre-populated with 1.
- **system.TenantID**—The current Tenant identifier. It is initialized from `_genesys.session.tenantid` or from `_genesys.ixn.interactions[system.InteractionID].tenantid` (if available). See the Update Contact, Identify Contact, Create Interaction, or Render Message block for more information.
- **system.TenantName**—The current Tenant name. It is initialized from `_genesys.session.tenant`.
- **system.LastTargetComponentSelected**—Target to which the interaction was routed definitively. See the Target Component Selected property of the Target block.
- **system.LastTargetObjectSelected**—High-level target to which the interaction was routed definitively. See the Target Object Selected property of the Target block.
- **system.LastTargetSelected**—DN and Switch name to which the interaction was routed definitively. See the Target Selected property of the Target block.
- **system.LastVirtualQueueSelected**—The Alias of the Virtual Queue specified in the target list where the interaction was routed. See the Virtual Queue Selected property of the Target block.
- **system.LastSubmitRequestId**—RequestId value of the last <queue:submit> execution. This variable is automatically updated when a successful (queue.submit.done) or unsuccessful (error.queue.submit) event is received. <queue:submit> is generated when using Target or RouteInteraction blocks. <queue:submit> may also be used if using SCXMLState or BeginParallel blocks.

Operational Parameter Management (OPM): These parameters are defined and provisioned in Genesys Administrator Extension (GAX)

- **system.OPM**—Used by the OPM Block (**App_OPM** is used in callflow diagrams).
- **system.ThisDN**— Initially set to the same value as system.DNIS. The value is updated by the interaction party state changed event handler (see IPD/Events property below) to the value of focusdeviceid. This variable becomes the default value for properties: ForceRoute/From, SingleStepTransfer/From, Target/From.
- **system.ParentInteractionID**— In case of Transfer scenario, this variable is assigned the ID of the parent interaction of the current interaction being processed.
- **system.OriginatingSession**— In case of context passing (see 'Pass Context' property description above), this variable holds the context of the originating session.

Outbound Contact Server (OCS) variables used by Outbound blocks:

- **system.OCS_RecordURI**— Its default value is set when the application starts executing from data passed into the application by the GVP or Orchestration platform. For workflows (SCXML), it is initialized from the userdata key "GSW_RECORD_URI". For callflows (VXML), `session.com.genesyslab.userdata.GSW_RECORD_URI` is used. This variable points to the 'current' record as determined by OCS and is provided to the application as a convenient way to communicate actions back to OCS for the current record.

- **system.OCS_URI**— Holds the OCS resource path in the format "`http(s)://<ocs host>:<ocs port>`". Its default value is deduced from `OCS_Record_URI`. The application can change this variable's value to use a different OCS application for all Outbound blocks in the workflow. Any downstream blocks will use the new value.
- **system.OCS_Record**— Holds the Record Handle deduced from the value of `OCS_Record_URI`.

Upgrading from Composer 8.1.1 or earlier

The system variables have been renamed in Composer 8.1.2 to improve the usability and to support new features. When upgrading a workflow that was initially developed with Composer 8.1.1 or earlier, the old set of system variables is kept, in addition to the new one, to ensure compatibility as some of those variables might be used in the application (for block properties or even in ECMA script code). However, users are now encouraged to use the variables that are "system." prefixed.

Composer 8.1.1	Composer 8.1.2
ANI	system.ANI
App_Language	system.language
App_Last_Error_Description	system.LastErrorDescription
App_Last_Error_Event	system.LastErrorEvent
App_Last_Error_Event_Name	system.LastErrorEventName
App_Last_Submit_Request_Id	system.LastSubmitRequestId
App_Last_Target_Component_Selected	system.LastTargetComponentSelected
App_Last_Target_Object_Selected	system.LastTargetObjectSelected
App_Last_Target_Selected	system.LastTargetSelected
App_Last_VirtualQ_Selected	system.LastVirtualQueueSelected
App_RelativePathURL	system.RelativePathURL
App_StartEvent	system.StartEvent
App_Terminate_Ixn_On_Exit	system.TerminateIxnOnExit
CallID	system.CallID
COMPOSER_WSSTUBBING	system.WebServiceStubbing
DNIS	system.DNIS
InteractionID	system.InteractionID
Tenant_Name	system.TenantName
TenantID	system.TenantID

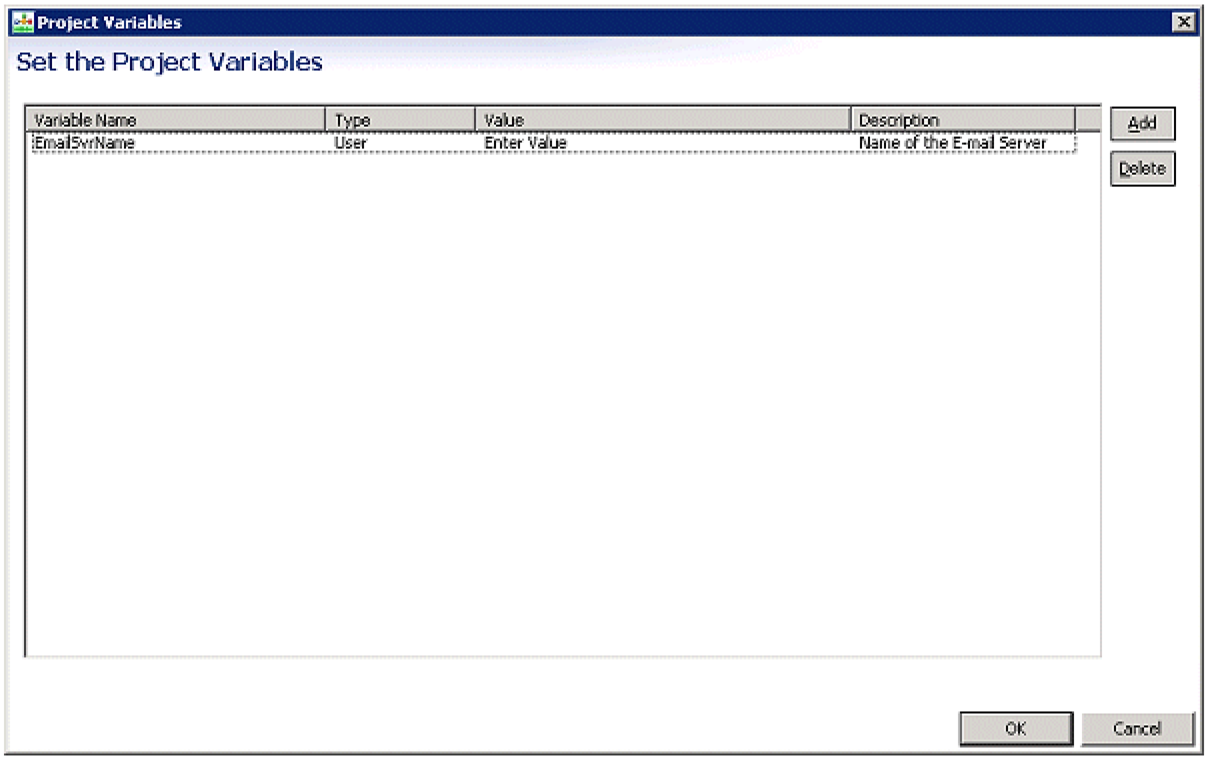
Project Variables

Project variables encompass all the workflows in a Project. These types of variables are defined in the data model of the interaction process diagram (IPD) SCXML application and so are required to be accessed by prefixing the name of a project variable with "_data.". For example, if you define a project variable named var1, you would access it as _data.var1. Project variables are also accessible in Expression Builder. Select Project Variables and then Variables under the Data Subcategory.

Use Project variables when information needs to be shared across workflows in an IPD. For example, if you want to get the e-mail address in one workflow and would like to create and send out an email in another workflow present in the same project. Genesys suggests defining at least one appropriately named Project variable like varProjectXYZ. Any properties that accept a variable will show this variable prefixed with _data in their list.

To define a project variable in Composer Design perspective:

1. Click the default.ixprocess tab (or the tab for a renamed IPD) to bring it into focus (tab is highlighted).
2. Click the Access Project Variables toolbar button. This opens the Project Variables dialog box. An example dialog box with one entry is shown below.



Variable Name	Type	Value	Description
EmailSvrName	User	Enter Value	Name of the E-mail Server

1. Click **Add**. The variable **Name**, **Type**, **Value**, and **Description** fields become editable.
2. Name the variable.
3. Specify an initial value if appropriate.
4. Describe the variable.
5. Click **OK**.

Internal Variables Naming

Starting with 8.1.1, Composer changes its naming policy for internal variables, which are variables that do not appear in any Variable edition dialog. They can be seen only in the generated SCXML code. Composer uses those variables to temporarily store data during an application execution.

Most Composer users will not be affected by this change. However, it is possible that some advanced users may have written applications that use those variables although they were not available "out-of-the-box." In such cases, those users will need to upgrade their application to use the variables with the new names.

Example

In 8.1.0, for a DB Data block named DbDataBlock Composer could declare in the SCXML application the variables:

App_DbDataBlock, App_DbDataBlockDBResult, App_DbDataBlockDBResultColumnNames, App_DbDataBlock_cursor, App_DbDataBlock_mapping

In 8.1.1, for a DbData block named DbDataBlock Composer could declare in the SCXML application only one variable named App_DbDataBlock.

Various properties of this variable will be used, such as App_DbDataBlock[‘requestid’], App_DbDataBlock[‘data’], App_DbDataBlock[‘DBResult’], App_DbDataBlock[‘DBResultColumnNames’], App_DbDataBlock[‘cursor’], App_DbDataBlock[‘mapping’]

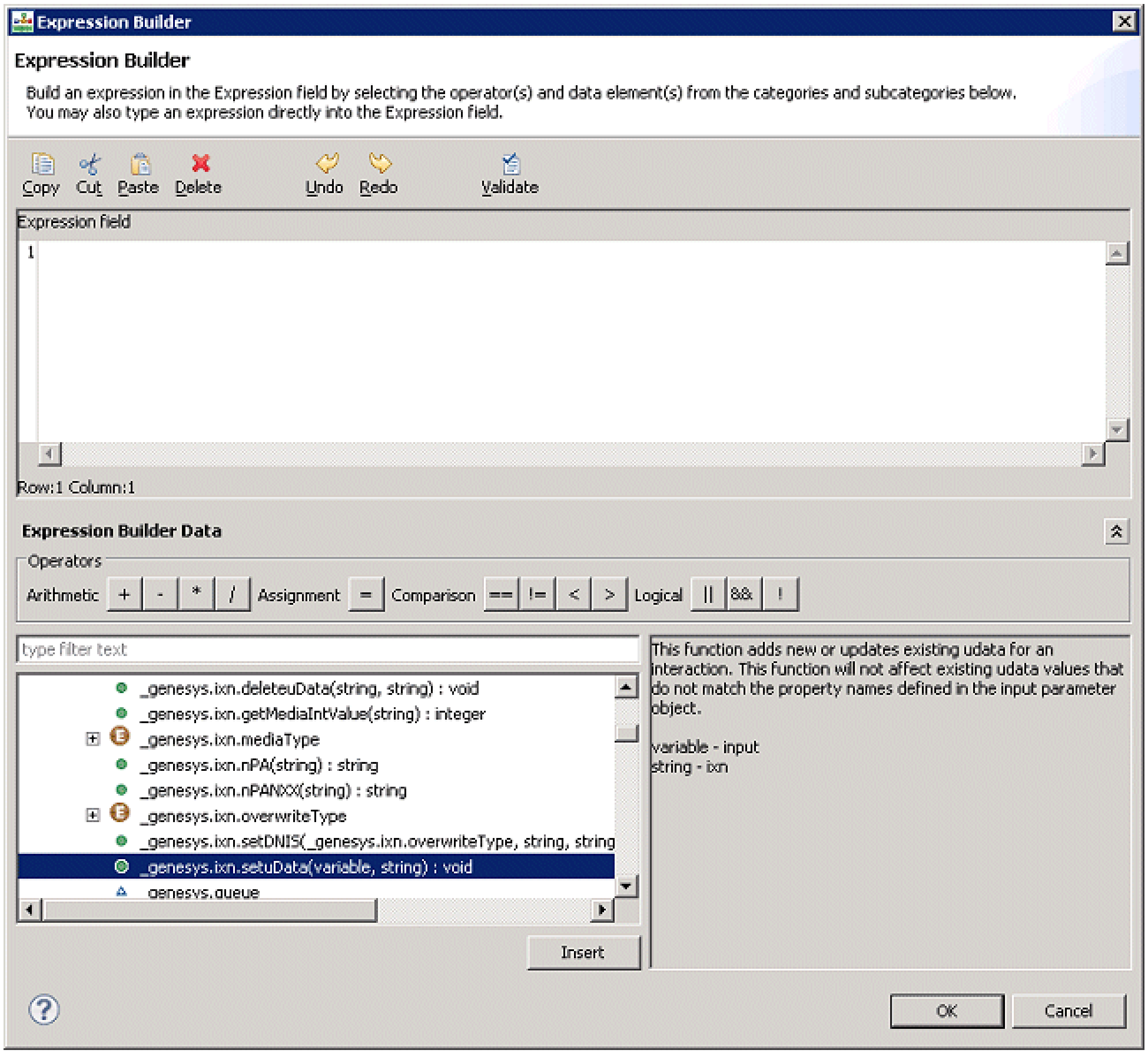
User Data

To work with User Data in Composer, you can use:

- The Interaction Data block for voice applications.
- The User Data property of the External Service block if you wish to pass User Data to an external service (routing and voice).
- The Entry block to access User Data (routing and voice).

For routing applications, you can use:

- The User Data block.
- The Create Email block, which lets you pick up standard response text from User Data.
- The Create SMS block, which lets you pick up message text from User Data.
- The Identify Contact block, which provides an option to update the interaction's User Data with the parameters returned by Universal Contact Server (Contact attribute data).
- The Create Interaction block, which lets you create a new interaction record in UCS database based on User Data.
- The ECMAScript block. The Script property lets you use Universal Routing Server User Data functions. Open Expression Builder. Select URS Functions and then _genesys.ixn.deleteuData (to add a User Data property or delete all properties) or _genesys.ixn.setuData (to add new or update existing User Data).


Expression Builder
Expression Builder
Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.
Copy
Cut
Paste
Delete
Undo
Redo
Validate
Expression field
Row:1 Column:1
Expression Builder Data
Operators
Arithmetic
+
-
*
/
Assignment
=
Comparison
==
!=
<
>
Logical
||
&&
!
type filter text
_genesys.ixn.deleteuData(string, string) : void
_genesys.ixn.getMediaIntValue(string) : integer
_genesys.ixn.mediaType
_genesys.ixn.nPA(string) : string
_genesys.ixn.nPANXX(string) : string
_genesys.ixn.overwriteType
_genesys.ixn.setDNIS(_genesys.ixn.overwriteType, string, string
_genesys.ixn.setuData(variable, string) : void
_genesys.queue
This function adds new or updates existing udata for an interaction. This function will not affect existing udata values that do not match the property names defined in the input parameter object.
variable - input
string - ixn
Insert
OK
Cancel

Hints

- A specific variable ‘xyz’ can be accessed directly; for example: _genesys.ixn.interactions[0].udata.xyz
- To write to User Data, use the setuData() function in an ECMAScript snippet. Usage is similar to the example below.

```
var input = new Object();

input.xyz = InputValue1; // Specify a value for the key ‘xyz’.

input[‘my-key-nname’] = ‘value’; // Use this notation if the key or property name has
a hyphen in it. Note that‘my-key-nname’has hyphens.
```

```
_genesys.ixn.setuData(input);
```

- Reading User Data is easier using the Assign block than with the ECMAScript block.

Mandatory Data for UCS Blocks

When working with the Update Contact and Render Message blocks (which map to Universal Contact Server services), certain properties must exist in the interaction User Data.

For the Update Contact block, ContactId must exist.

For the Render Message block, ContactId (if some contact-related Field Codes (as described in the *eServices 8.1 User's Guide*) are used in the text to render). Also InteractionId (if some interaction-related Field Codes are used in the text to render)and OwnerEmployeeId (if some agent-related Field Codes are used in the text to render).

As is the case with IRD, these properties are not set in the blocks themselves. Instead, the properties are assumed to be put in the interaction's User Data by some other block earlier in the workflow, such as the Identify Contact block or Create Interaction block with the Update User Data property set to true. In case no other block does this, the User Data block may be used for this purpose.

If these properties are not available, an explicit UCS error message (missing parameter) shows in the Orchestration Server log.

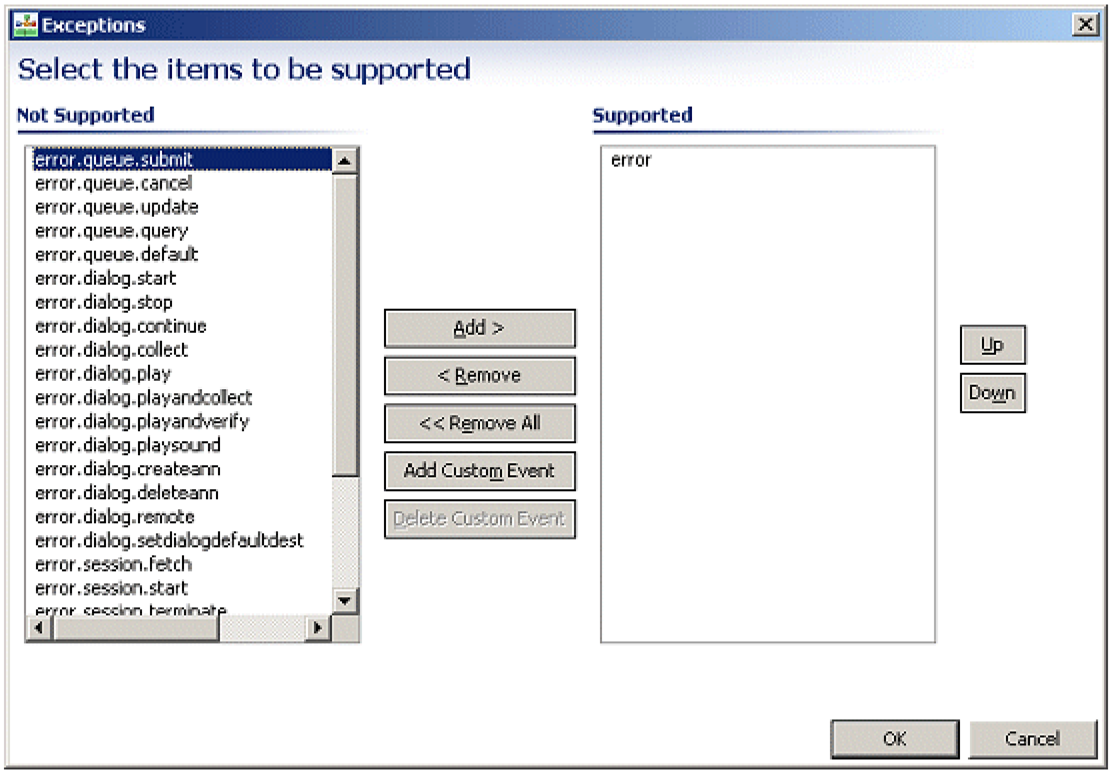
Callflow User Data

Also see the following blocks used for callflows:

- Interaction Data
- Route Request

Custom Events

The Exceptions dialog box, which opens from the Exceptions property, contains an Add Custom Event button. The figure below shows the Exceptions dialog box for the Entry block.



Not Supported:

error.queue.submit
error.queue.cancel
error.queue.update
error.queue.query
error.queue.default
error.dialog.start
error.dialog.stop
error.dialog.continue
error.dialog.collect
error.dialog.play
error.dialog.playandcollect
error.dialog.playandverify
error.dialog.playsound
error.dialog.createann
error.dialog.deleteann
error.dialog.remote
error.dialog.setdialogdefaultdest
error.session.fetch
error.session.start
error.session.terminate

Supported:

error

Adding a Custom Event

To add a custom event:

1. In the Exceptions dialog box, click **Add Custom Event**.
2. In the resulting dialog box, name the event and click **OK**. The event name appears in the **Supported** column.
3. When through in the dialog box, click **OK**.
4. Once a custom event is added to the list of exceptions in a block, you will see an exception port for this event (or exception) on the block, which you can now connect to another block to handle that special condition.

Expression Builder

Use for both voice callflows and routing workflows to build expressions for branching and conditional routing decisions. You create expressions in Expression Builder; you can assign them to variables using the Assign block. You can also build ECMAScript expressions that use the Genesys Functional Modules in Expression Builder). Also see Skill Expression Builder used for routing applications.

Opening Expression Builder

Expression Builder opens from the various blocks, which include but are not limited to:

- Assign--Assign Data Property
- Branching--Conditions Property
- ECMAScript (for workflows)--Script Property
- Entry--Variables Property
- Log--Logging Details Property
- Looping--Exit Expression Property

Data Categories Accessed

Expression Builder gives access to the following categories of data, which can be used in expressions:

- Project Variables
- Workflow Variables (if accessed from a workflow)
- Callflow Variables (if accessed from a callflow)
- Workflow Functions (if accessed from a workflow)
- Callflow Functions (if accessed from a callflow)
- JavaScript (Array, Boolean, Date, Math, Number, String functions)
- Orchestration Server Functions
- Context Services
- Configuration Server

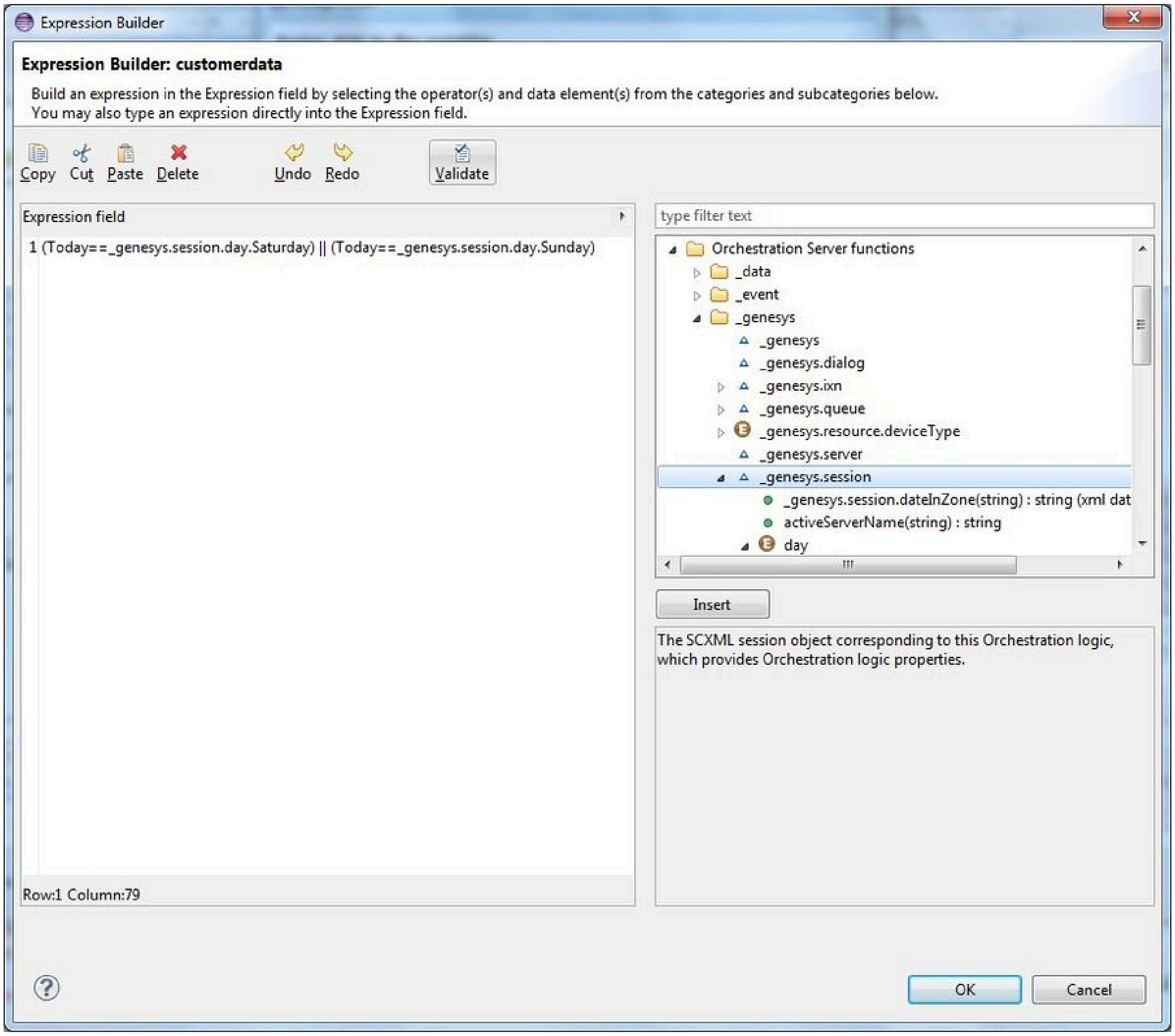
Note: Depending on the calling context (IPD, workflow editing, callflow editing), some of the above categories may be hidden.

Expression Builder Toolbars

This section discusses the editing toolbar the top of Expression Builder as well as the Operators toolbar.

Editing Toolbar

Expression Builder has an editing toolbar with buttons representing copy, cut, paste, delete, undo, redo, and validate . After you entering an expression and click the button to validate, syntax messages appear under the Expression Builder title. In the figure below, the syntax message is: No syntax error was found.



Expression Builder

Expression Builder: customerdata

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

```
(Today==_genesys.session.day.Saturday) || (Today==_genesys.session.day.Sunday)
```

Row:1 Column:79

type filter text
Orchestration Server functions
_data
_event
_genesys
_genesys
_genesys.dialog
_genesys.ixn
_genesys.queue
_genesys.resource.deviceType
_genesys.server
_genesys.session
_genesys.session.dateInZone(string) : string (xml dat
activeServerName(string) : string
day
Insert

The SCXML session object corresponding to this Orchestration logic, which provides Orchestration logic properties.

Operators

Expressions can consist of comparisons joined by AND (&) and OR (|), which have the same priority. URS uses integer arithmetic in its calculations, such as for expression evaluation. For this reason, you must always create expressions based on integer arithmetic, not float. When an expression contains more than one logical comparison, the logical operation to the left has precedence over the operation to the right. Use parentheses to overrule this precedence. When the order of logical operations is not explicitly specified by parentheses, the operation to the left has precedence over the operation to the right. For example: `Portuguese > 5 | Africa = 7 & SpTours >3` is equivalent to `(Portuguese > 5 | Africa = 7) & SpTours > 3` Each comparison consists of two data values that are compared against each other. The table below shows the operators used in logical expressions.

Symbol	Meaning	Example
==	use for comparison (e.g. x==y to compare is x equal to y)	see example screen shot
=	equal to done for assignment (e.g. x=y to assign x equal to y)	Day = Monday
!=	not equal to	Day != Sunday
&	Single & is an AND operators which manipulates internal bits of integers	
&&	&& and || are used for comparisons to build boolean (true/false) expressions	
||	see &&	see example screen shot
>	greater than	Time > 9:00
>=	greater than or equal to	Day >= Monday
<	less than	Time < 16:00
<=	less than or equal to	Day <= Friday

Mathematical Symbols Allowed

The allowed mathematical symbols in expressions are restricted to those symbols for addition, subtraction, division, and multiplication (+, -, /, *). These symbols work with numeric values only, so any participating String argument is automatically converted to a number.

Examples of Expressions

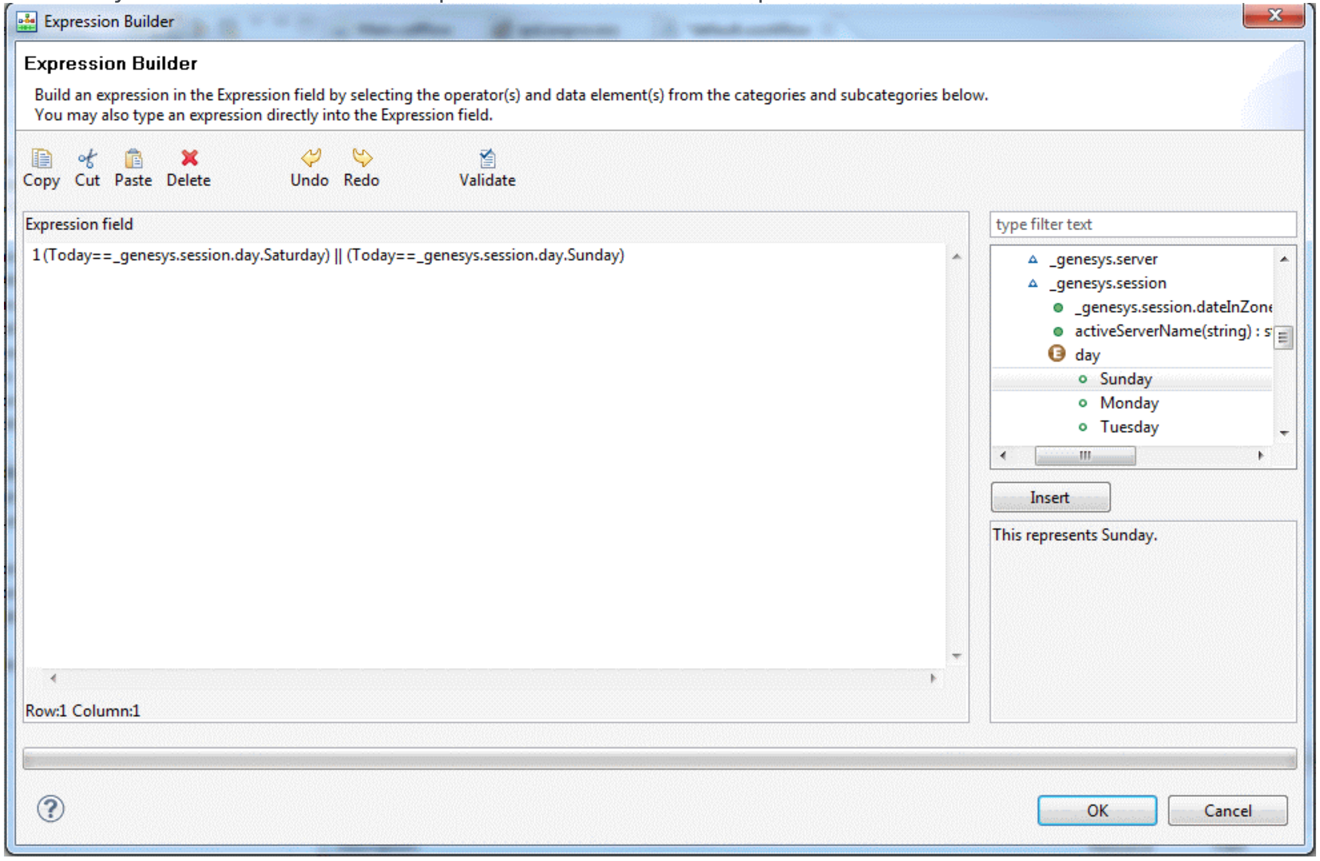
The table below shows example expressions.

Example	Interpretation
Day=Saturday|Day=Sunday	If the day is Saturday or Sunday
Time<=5:00 |Time>=18:00	If the time is less than or equal to 5:00 (5:00 AM) or

	greater than or equal to 18:00 (6:00 PM); in other words, if the time is between 6:00 PM and 5:00 AM.
Day>=Monday&Day<=Friday & ANI!=8004154732	If the day is a weekday and the ANI is not 8004154732
SData[1123@SF.A,StatTimeInReadyState]>120	If agent 1123 has been in Ready state for more than 120 seconds
UData[Acct]>=9000	If the value associated with the key Acct in the Interaction data structure is equal to or greater than 9000

Creating Expressions

Assume you wish to create the expression shown in the Expression field below.



Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below. You may also type an expression directly into the Expression field.

```
(Today==_genesys.session.day.Saturday) || (Today==_genesys.session.day.Sunday)
```

This represents Sunday.

In the above figure, the expression `(Today ==_genesys.session.day.Saturday) || (Today == _genesys.session.day.Sunday)` was created as follows:

- In the left text field under the toolbar, type an open parenthesis (().
- If you already defined the "Today" variable, select it under **Workflow variables** or **Project variables**. Otherwise, if not yet defined, type "Today" (no quotes) in the text after the open parenthesis.
- Type "==" (no quotes) in the text.

- In the right text box, expand:

`Orchestration Server Functions _genesys genesys.session.day`

- Double-click **Saturday**.
- In the left text field, type a close parenthesis ())
- Click the **|| Operators** button. Repeat the above steps except, at the end before the close parenthesis, select _genesys.session.day.Sunday.

Note: You may also use the search field on the right side to filter items that include **Saturday**

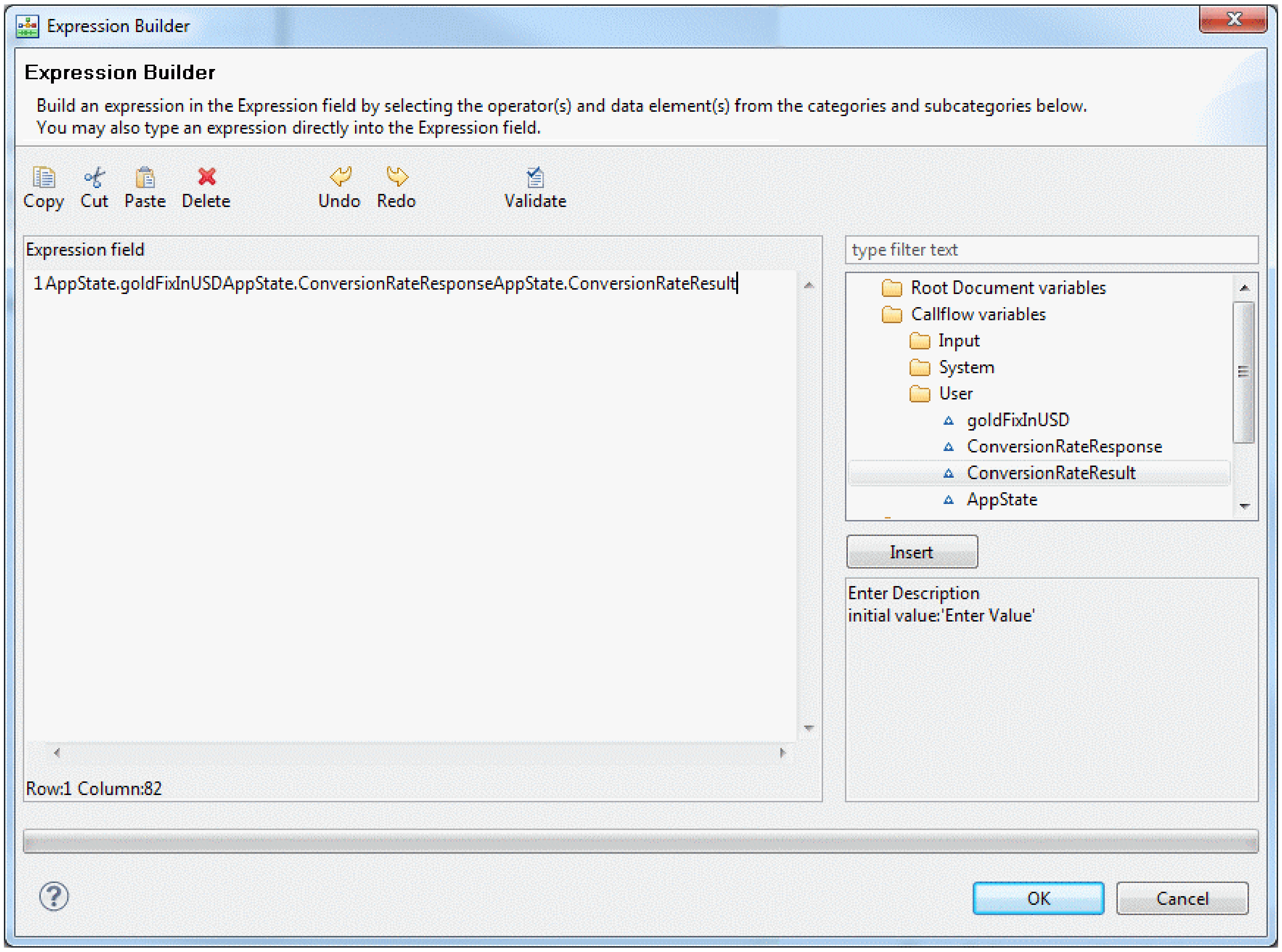
Usage of Variables in Expressions

Note: The steps for using variables in Expression Builder varies slightly depends on whether you are creating a voice callflow or a routing workflow.

- If creating a voice callflow, the right selection area contains Callflow Variables.
- If creating a routing workflow, the selection area contains Workflow Variables.

Using Variables in a Callflow

Assume you wish to use variables in a callflow to create the following: `AppState.goldFixInUSDAppState.ConversionRateResponseAppState.ConversionRateResult` The figure below shows the entry in Expression Builder.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

Expression field

```
1 AppState.goldFixInUSDAppState.ConversionRateResponseAppState.ConversionRateResult
```

Row:1 Column:82

- Root Document variables
- Callflow variables
  - Input
  - System
  - User
    - goldFixInUSD
    - ConversionRateResponse
    - ConversionRateResult
    - AppState

Enter Description
initial value:'Enter Value'

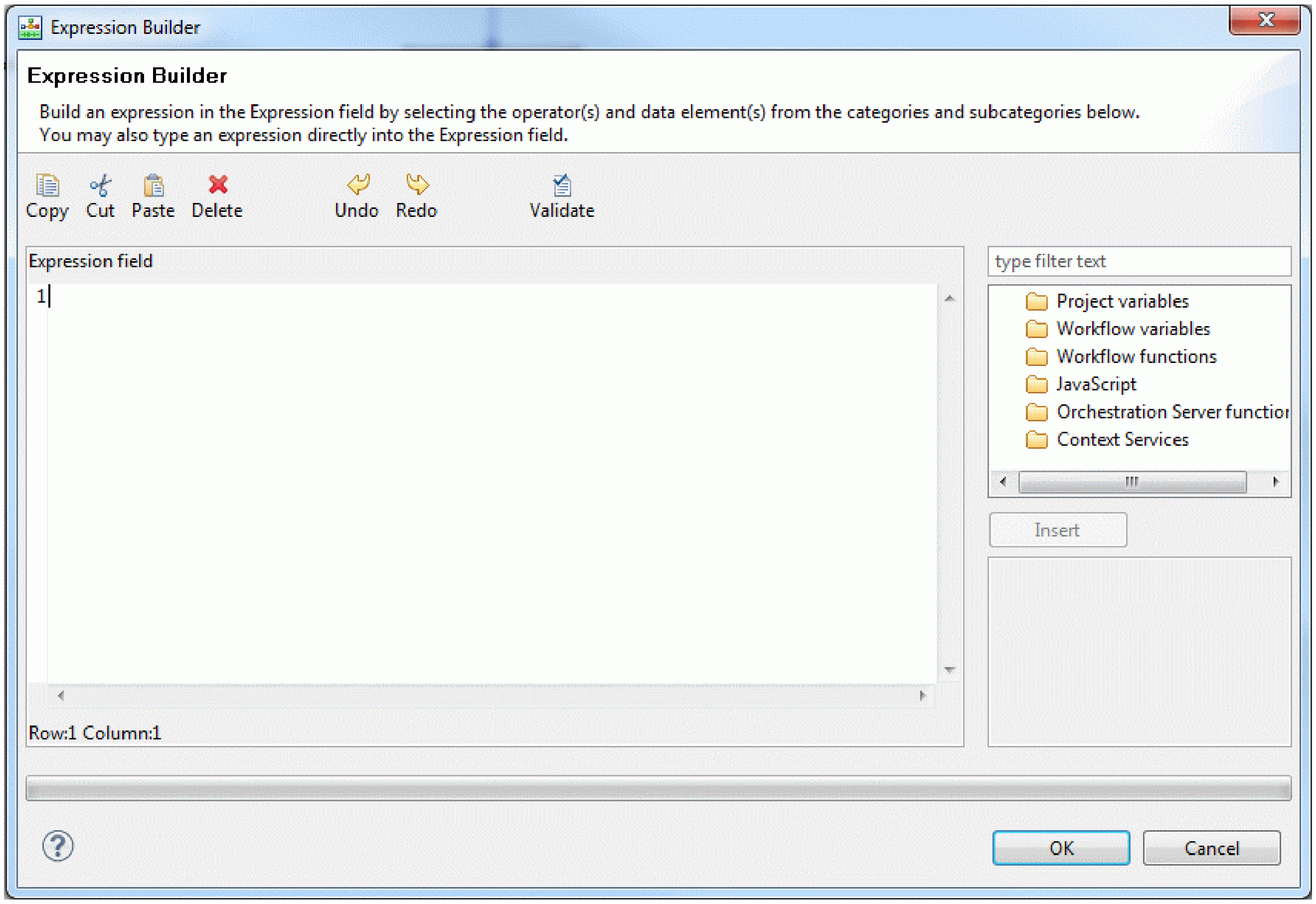
Assume you have already defined these variables in the Entry object. The above entry could then be created as follows:

1. The right selection area lists **Callflow Variables**, **Java Script**, and **Context Services**.
2. Expand **Callflow Variables**.
3. Expand **User** to view variables defined in the Entry block.
4. To place the variables in the Expression field at the top of Expression Builder, double-click **goldFixInUSD**,**ConversionRateResponse**, and **ConversionRateResult**. The AppState VoiceXML Data Model Object will be appended automatically to variables used inside Expression Builder. The code inside the Expression field will be directly substituted within the generated VoiceXML code.
5. Continue creating the expression.

Expression Builder Data Categories

Expression Builder accesses various categories of data Including Orchestration Server and Universal

Routing Server functions. The folders shown on the right depend on whether you are working with a callflow or workflow. The figure below shows Expression Builder Data Categories when working with a workflow.


Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below. You may also type an expression directly into the Expression field.

Project variables

Use Project variables when you need to share information across different workflows.

Workflow variables

Use Workflow variables when you need to share information across different blocks in the same workflow.

Workflow functions

Use the Workflow functions category when creating routing workflows. Selecting a function displays a description. See the Orchestration Server wiki for information on functions available for use in Composer when building routing workflows. **Notes**: Functions getCallType and getIxnMediaType can be used to identify the call type and/or media type for the purpose of segmenting incoming interactions.

- `getCallType(ixnID)`--This function gets the call type `_genesys.ixn.interactions[].voice.type` for the specified interaction. If the ixnID is not specified, it will return the calltype for the current interaction.
- `getIxnMediaType(ixnID)`--This function gets the correct media type ENUM from `_genesys.FMName.MediaType` for the specified ixnID. If ixnID is not specified, the current interaction id will be used. If the interaction's media type cannot be determined or the specified ixnID does not exist, the function will return undefined.

Use Case: A call arrives on a routing pointing, initiating a routing workflow. The workflow checks for the call type. If the call type is outbound, then the call is immediately moved to the front of the queue and routed to an available agent. If the call type is inbound, the call is assigned a priority and routed based on the desired service level. For information on the other functions, see eServices Blocks.

Callflow functions

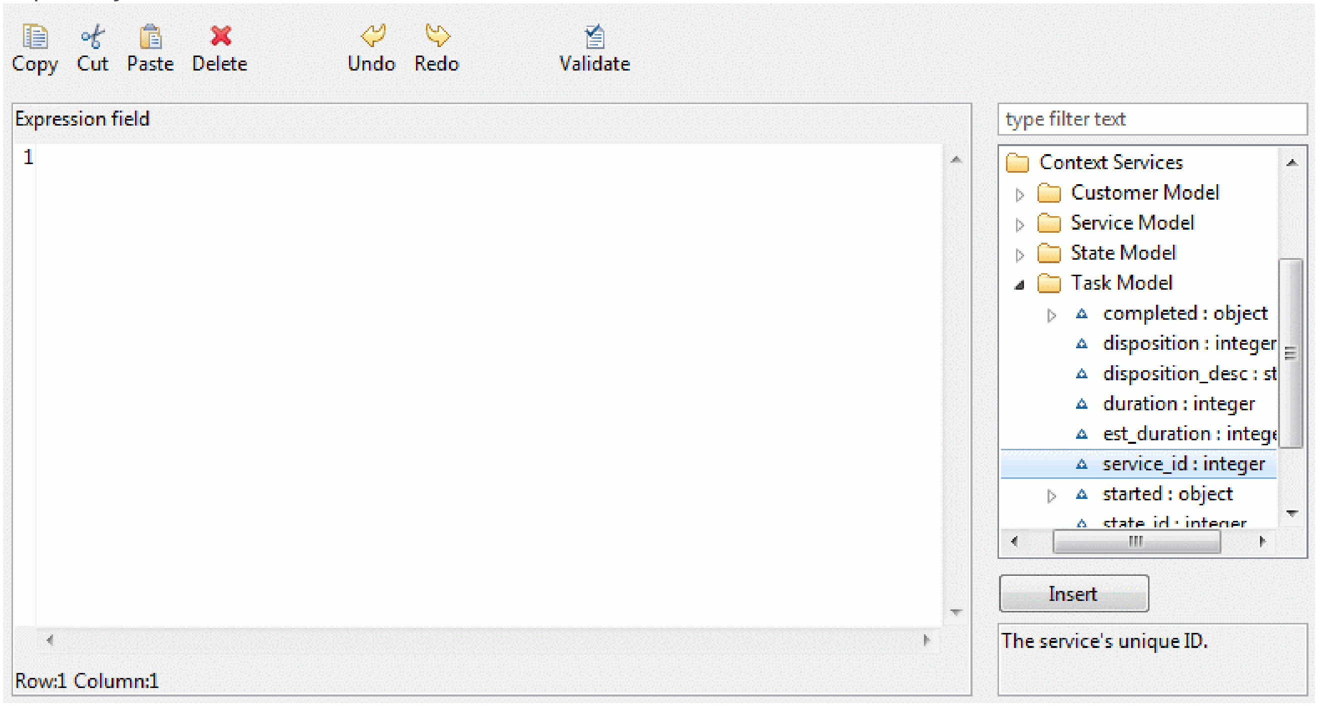
Note: Function `getSIPHeaderValue(headername)` returns the SIP header value associated with the given SIP headername. You may wish to use this function with the Assign block.

JavaScript

Use JavaScript to access those functions categorized as follows: Array, Boolean, Date, Math, Number, String.

Context Services

Use Context Services when creating expressions that use attributes associated with this optional capability of Universal Contact Server Database.



The service's unique ID.

Configuration Server

This category is displayed for workflows when the Expression Builder is called from the ECMAScript and Branching blocks. If connected to Configuration Server, Composer can fetch standard responses and category codes.

Standard Response Library

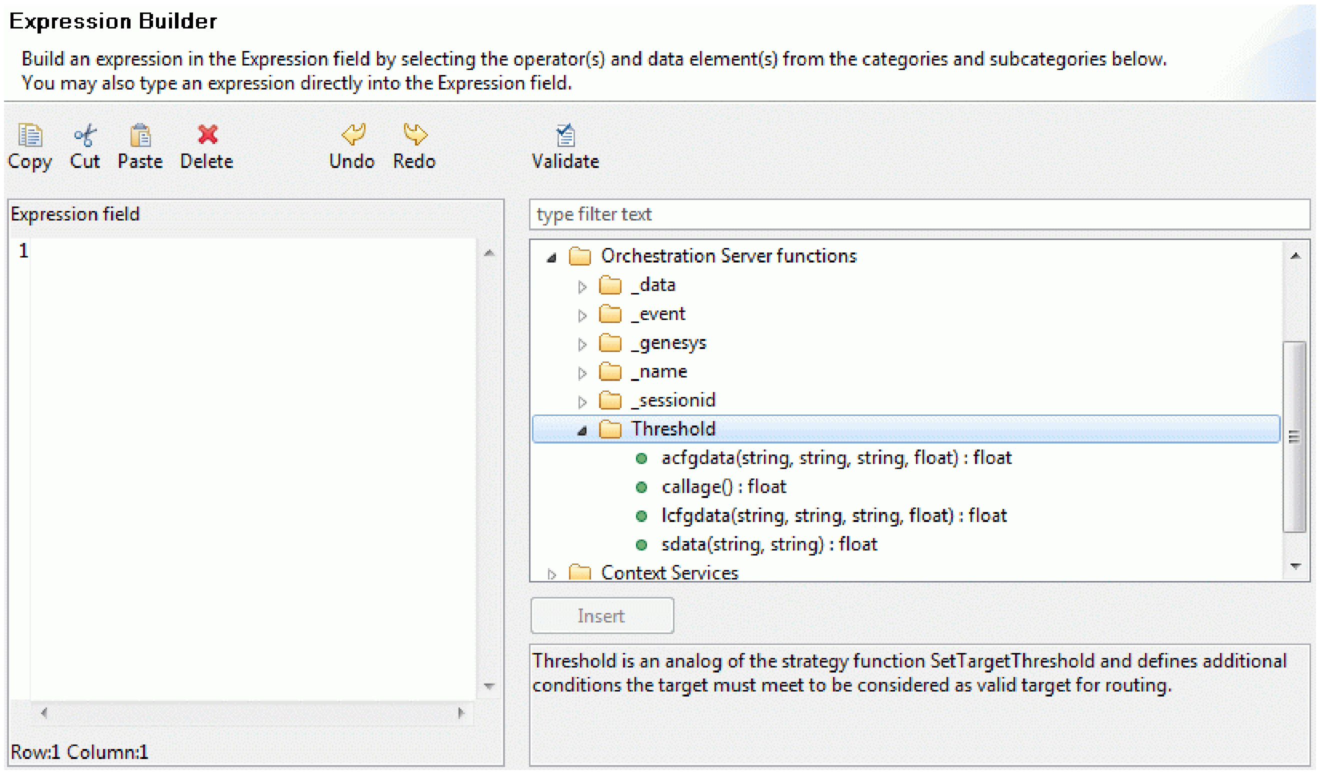
This category allows you to access pre-written responses for customers that have been defined in Knowledge Manager (as described in the *eServices 8.1 User's Guide*).

Orchestration Server and URS Functions

Composer's Expression Builder provides access to many Orchestration Server and Universal Routing Server (URS) functions. For more information see Using URS Functions.

Threshold Functions

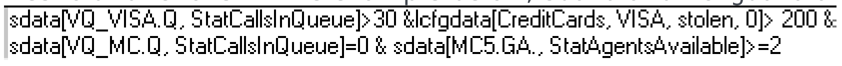
Universal Routing Server's threshold functions can be used for conditional routing. For example, the threshold functions can be used in the Target block for a type of conditional routing called "share agents by service level agreement routing." This type of routing enables a business user that manages multiple business lines to define the triggering conditions and constraints that allow agents to be shared among business lines. By constructing a single threshold expression, you can define the triggering conditions for borrowing agents from other business lines as well as the conditions that apply to the lending business line.
Threshold is an analog of the URS strategy function `SetTargetThreshold` as defined in *Universal Routing 8.1 Routing Application Configuration Guide*. It defines additional conditions the target must meet to be considered as valid target for routing. The threshold functions are available in Composer's Expression Builder:


Expression Builder
Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.
Copy
Cut
Paste
Delete
Undo
Redo
Validate
Expression field
type filter text
Orchestration Server functions
_data
_event
_genesys
_name
_sessionid
Threshold
acfgdata(string, string, string, float) : float
callage() : float
lcfgdata(string, string, string, float) : float
sdata(string, string) : float
Context Services
Insert
Row:1 Column:1

Threshold is an analog of the strategy function SetTargetThreshold and defines additional conditions the target must meet to be considered as valid target for routing.

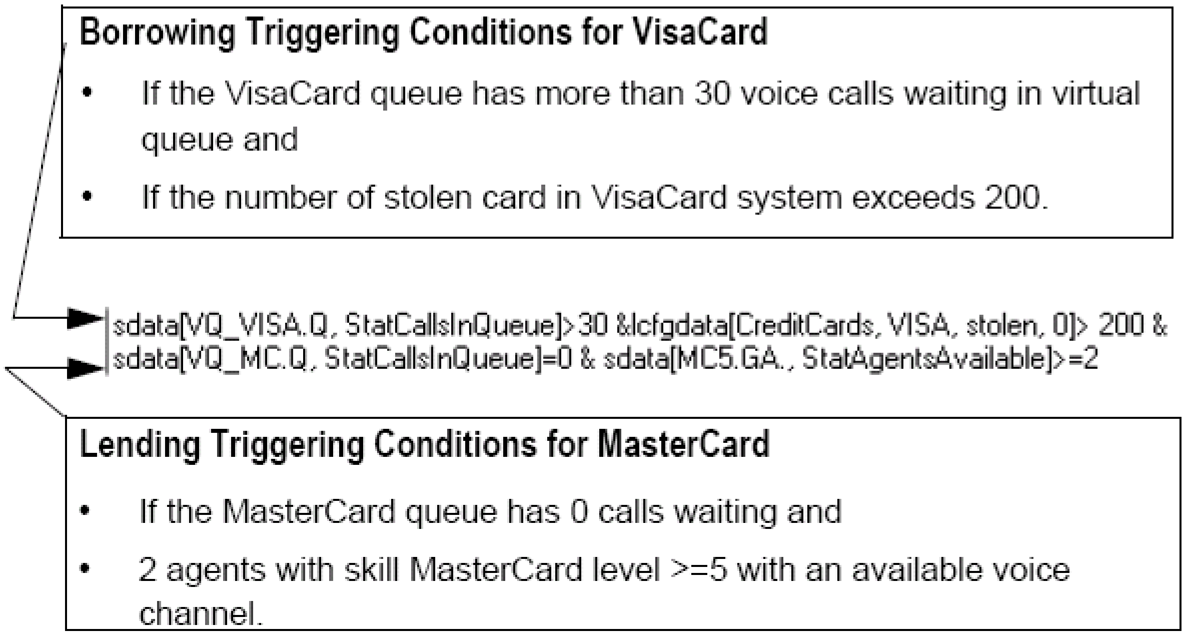
- `acfgdata(Application name, folder, property, default value)`. Use this function to affect routing conditions based on external data stored in properties of Configuration Database Application objects (ApplicationConFigDATA). Returns a numeric value for a specified Application option. If an Application has no such option then the default value is returned. Return value type: FLOAT. Example: sdata(Group2.GA, StatAgentsAvailable)>acfdata(URS, default, MinNumOfRdyAgents, 2)
- `callage`. Use this function to return the age of an interaction in seconds. Use for time-based routing conditions, such as a call that can only be routed if it waits more then 60 seconds. Return value type: FLOAT.
- `lcfgdata(list name, item, attribute, default value)`. Use this function to affect routing conditions based on external data stored in List objects. Returns a numeric value for a specified attribute of a List object's item If a List object has no such item or attribute, the default value is returned. Works like acfgdata, but uses a List object (`ListConFigDATA`) instead of an Application. Return value type: FLOAT.
- `sdata(target, statistic)`. Use this function to affect routing conditions based on statistics. Specify targets and statistics just like for the `SData[]` function described in the *Universal Routing 8.1 Reference Manual*. You can use the URS predefined statistics (see Statistics Manager and Builder), such as: PositionInQueue, CallsWaiting, and InVQWaitTime. Return value type: FLOAT. Example: sdata(Group2.GA, StatAgentsAvailable)>2

Example Threshold Expression

A threshold expression is text string very similar to the regular expressions used for branching, but uses the threshold functions. In the example below, `sdata` and `lcfgdata` are the predefined threshold functions. 

```
sdata[VQ_VISA.Q, StatCallsInQueue]>30 &lcfgdata[CreditCards, VISA, stolen, 0]> 200 &
sdata[VQ_MC.Q, StatCallsInQueue]=0 & sdata[MC5.GA., StatAgentsAvailable]>=2
```

In this example, both the borrowing and lending conditions are defined in a single threshold expression:



Borrowing Triggering Conditions for VisaCard

- If the VisaCard queue has more than 30 voice calls waiting in virtual queue and
- If the number of stolen card in VisaCard system exceeds 200.

```
sdata[VQ_VISA.Q, StatCallsInQueue]>30 &lcfgdata[CreditCards, VISA, stolen, 0]> 200 &
sdata[VQ_MC.Q, StatCallsInQueue]=0 & sdata[MC5.GA., StatAgentsAvailable]>=2
```

Lending Triggering Conditions for MasterCard

- If the MasterCard queue has 0 calls waiting and
- 2 agents with skill MasterCard level >=5 with an available voice channel.

For detailed information on using the threshold functions, see *Universal Routing 8.1 Routing Application Configuration Guide* and threshold attribute in Orchestration Developers Guide.

GetMediaTypeName Function

This function returns the name of media type associated with interaction as defined in the Configuration Database. Located in Expression Builder as follows: `_genesys.ixn.mediaType` Expression Builder lists the Media type enumerators supported by the URS platform. The main difference between this function and IRD's `getMediaTypeName` function is that

- The IRD function takes a parameter for (the current interaction media type) and returns a String name of the media type

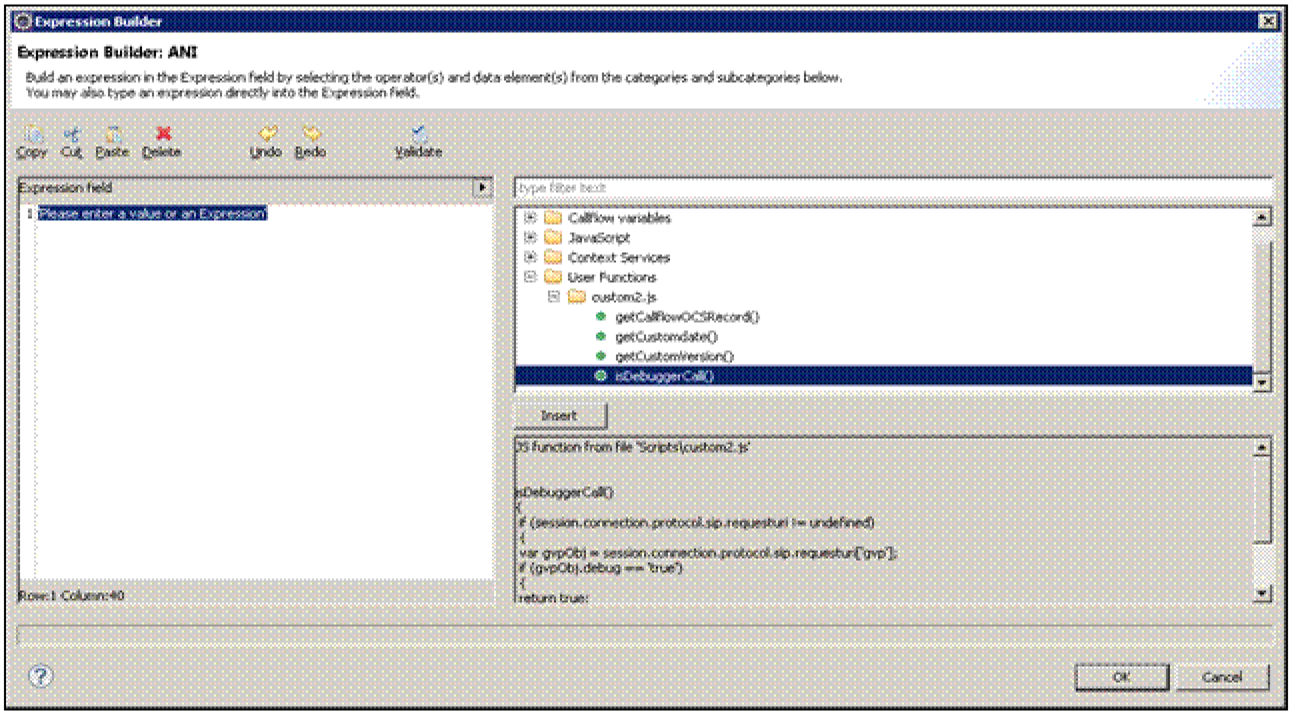
Whereas:

- Composer's function `getIxnMediaType(ixnID)` takes a parameter of any interaction and returns a ENUM type `_genesys.FMName.MediaType` of the media type. Returns the media type of the given interaction (ixnID), otherwise the current interaction.

User defined JavaScript Functions

Expression Builder shows the user defined JS methods under 'User Functions' category to list user defined JS methods for easy access.

- For Callflow diargams, JavaScript files added in the Scripts property of the Entry BlockScripts are considered.
- For Workflow diargams, JavaScript files added inside the include/user folder are considered.



Expression Builder: ANI

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

```
1 Please enter a value or an Expression
```

Row:1 Column:40

```
JS function from file 'Scripts\custom2.js'


isDebuggerCall()
{
if (session.connection.protocol.sip.requesturi != undefined)
{
var gvpObj = session.connection.protocol.sip.requesturi['gvp'];
if (gvpObj.debug == 'true')
{
return true;
```

Skill Expression Builder

You can route interactions to the most appropriately skilled agent using a skill expression or a statistical expression. Skill Expression Builder lets you create both types of expressions, which produce a result of true or false.

Also see: Variables and Literals

Using Skill Expression Builder

Open Skill Expression Builder from the Targets property in the Target block as follows:

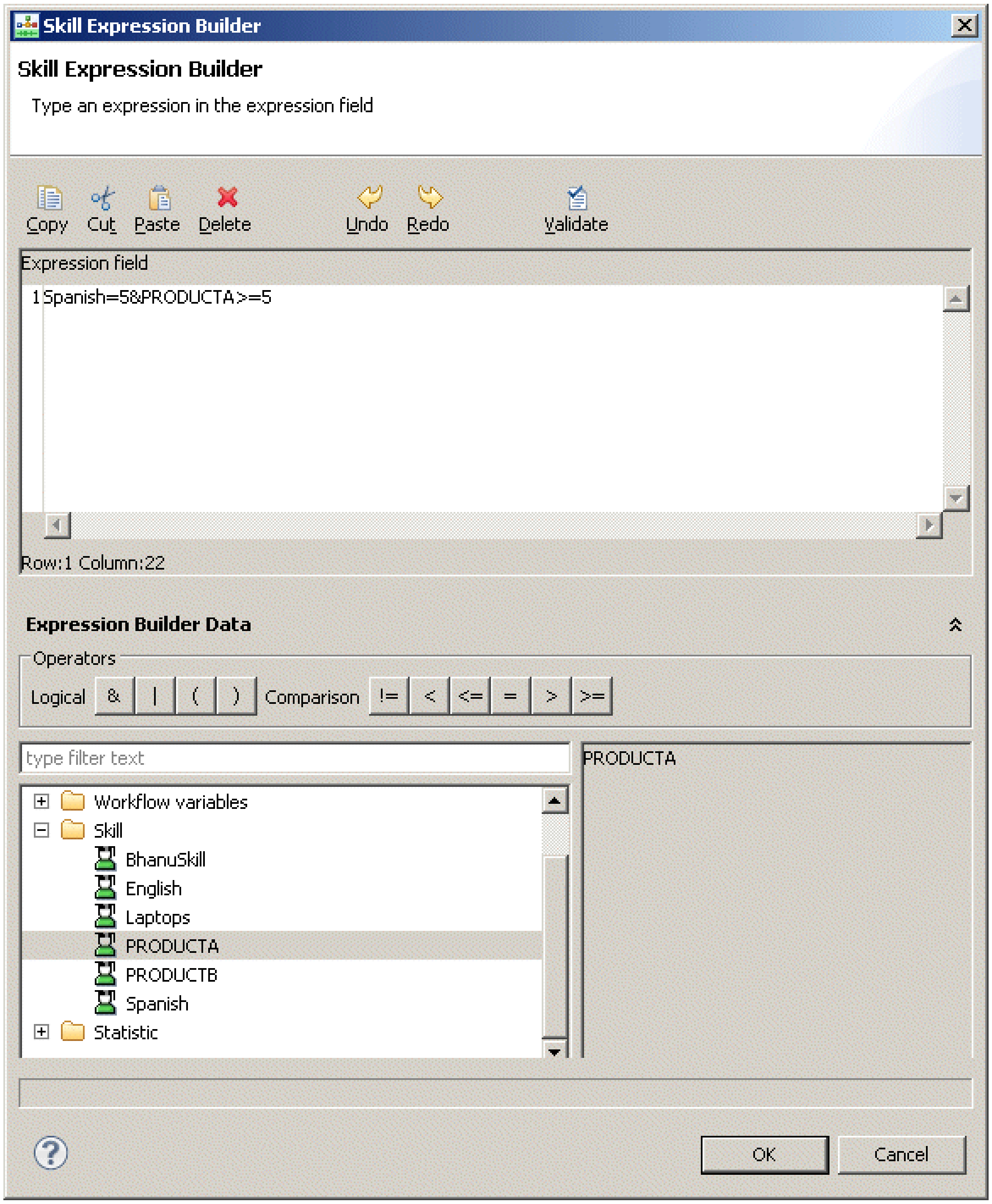
1. If you have not already done so, Server connect to Configuration Server. Otherwise, when selecting a Target of type Skill, the Skill Expression Builder will not be available.
2. Set the Validate Skill Expressions preference.
3. Opposite the Targets property, click under Value to display the button.
4. Click the button. The Targets dialog box opens.
5. Click Add in the Targets dialog box.
6. Click under Type to display a down arrow.
7. Click the down arrow and select the Skill target type.
8. Click under the Name field to display the button.
9. Click the button to bring up the Skill Expression Builder. Skill and Statistic appear in the lower text area.
   - **Skill**–The name a skill defined in the Configuration Database. Skill names are limited to alphanumeric characters and underscores, cannot begin with a digit, and cannot exceed 126 characters.
   - **Statistic**–The name a statistic/metric defined in the Configuration Database. The statistic name in a skill expression can be any agent statistic used in URS function SData, which returns the current value of the statistic for a given target. For example, you may wish to have URS return the number of interactions waiting, so that if a target is not available, the caller will hear the IVR announce the number of interactions ahead of him.

The statistic must be written in the format: $(statisticname). For example: $(StatAgentLoggedIn)=1

Using Skill

1. Expand **Skill**. Skill objects in your Configuration Database are listed for selection.

2. Select a Skill.
3. Create the skill expression using a combination of the Comparison symbols, values, variables, and the & | (, and) logic operators. An example is shown below.


Skill Expression Builder

Skill Expression Builder

Type an expression in the expression field

Expression field

```
Spanish=5&PRODUCTA>=5
```

Row:1 Column:22

Expression Builder Data

Operators

Logical & | ( ) Comparison != < <= = > >=

type filter text

- Workflow variables
- Skill
  - BhanuSkill
  - English
  - Laptops
  - PRODUCTA
  - PRODUCTB
  - Spanish
- Statistic

PRODUCTA

4. Click the button to validate on the toolbar. For example, the expression

```
Spanish > 5 & ProductA >= 5
```

was created as follows:

- Expand **Skill**. Double-click **Spanish**. Click the **>** symbol to insert. Type the number **5** followed by a space. Click the button for the **&** logic operator. Type a space. Double-click **ProductA**. Click the **>=** symbol. Type the number **5.**
- Click**OK.**

Using Statistic

1. Expand **Statistic**. URS predefined statistics appear for selection.
2. Select a statistic.
3. Click a comparison symbol.
4. Create the expression using a combination of the comparison symbols, values, and the & | (, and) logic operators. Example:

```
$(RStatCallsInQueue) = 3 & $(RStatCost) = 2 | $(PositionInQueue)) >=6.
```

Note: Use of the RStatCost statistic requires that you have cost-based routing implemented at your site. For a description of each statistic, consult the Universal Routing 8.1 Reference Manual.

5. Click the button to validate on the toolbar.
6. Click **OK** when through creating the expression to return to the Targets dialog box.

Comparison Symbols

The table below describes the comparison symbols used to evaluate a skill condition.

Symbol	Interpretation
!=	Differs depending on the Data type: Skill–not equal to the indicated level value. Statistic–not equal to the indicated statistic value.
<	Differs depending on the Data type: Skill–less than the indicated level value. Note: depending on how you use this operator, it may result in including agents that do not have the skill at all (skill name = 0). For example, with English < 8, the queue functional module includes all agents with the English skill less than 8, and also agents with no English skill at all. Statistic–less than the indicated statistic value
<=	Differs depending on the Data type:Skill–less than

	or equal to the indicated level value. Statistic–less than or equal to the indicated statistic value
=	Differs depending on the Data type: Skill–equal to the indicated level value. Statistic–equal to the indicated statistic value.
>	Differs depending on the data type: Skill–greater than the indicated level value. Statistic–greater than the indicated statistic value
>=	Differs depending on the Data type: Skill–greater than or equal to the indicated level value. Statistic–greater than or equal to the indicated statistic value.

This is a value of the same data type as the Data name element. The value must already evaluate to an integer. Float numbers are not supported. There are different limitations depending on the data type:

- **Skill value**–This value represents the level of skill. For example; an agent could have an English skill level greater than 3 (English > 3). An agent can be excluded from a skill by setting that agent's skill level for that skill to zero in the configuration layer (English=0).
- **Statistic**–This value represents the value of the statistic/metric. For example; an agent could be in Ready state longer than 20 seconds ($(StatTimeInReadyState) > 20).

Logic Operators

Use the logic operators to evaluate multiple conditional expressions together. The following logic operators are supported:

- **AND** (&)
- **OR** (|).

The AND and OR logic operators have the same priority. For example:

```
English >3 & $(StatAgentLoggedIn)=1
```

Variables and Literals

Starting with 8.1:

- Composer supports variables in skill expressions (they appear in the Skill Expression builder tree).
- You must enclose literal expressions in single quotes.

Background: Previously, Composer automatically added single quotes around the expression entered by the user. Now that variables are supported, Composer must distinguish literal strings and variables. As a result, you must enclose literal strings in single quotes.

List Objects Manager

A List object contains strings of any nature (for example, DNIS or ANI strings), which can be used in strategies. The strings can be as simple as 800 numbers or as complex as routing conditions.

For example, you may use a List object to create lists of toll-free numbers. Rather than reference each individual 800 number in a strategy, you can logically group numbers together and name the group. Then, when you need to add new numbers or edit numbers, you do not need to edit the strategy, but only the List object properties.

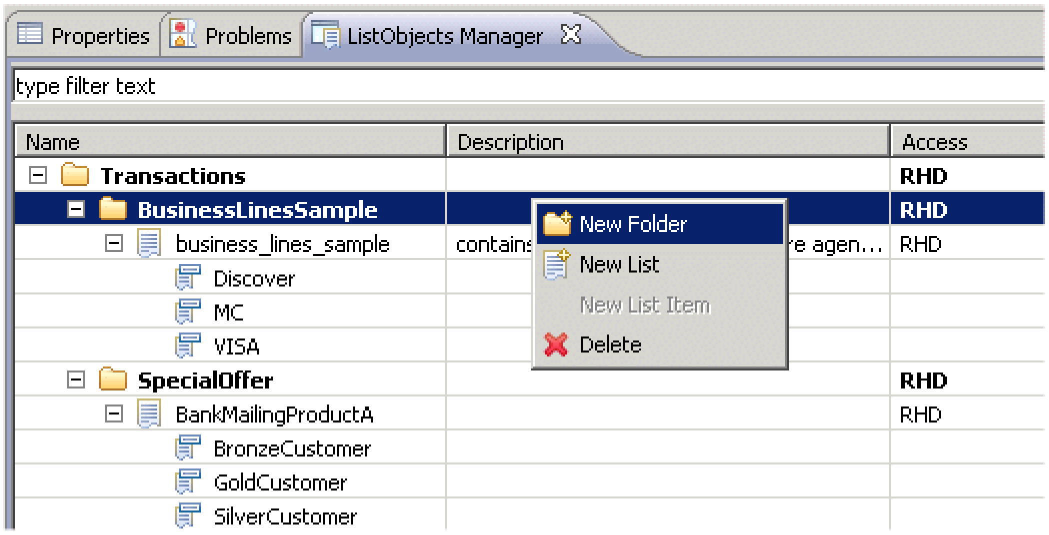
Providing key-value pairs for a List element enables you to store information in the Configuration Server Database and then retrieve it from the strategy.

You can specify a List Object variable when specifying targets in the Target selection block.

Creating List Objects

Composer supplies a List Objects Manager view to create List objects. To bring up the view and create a List object:

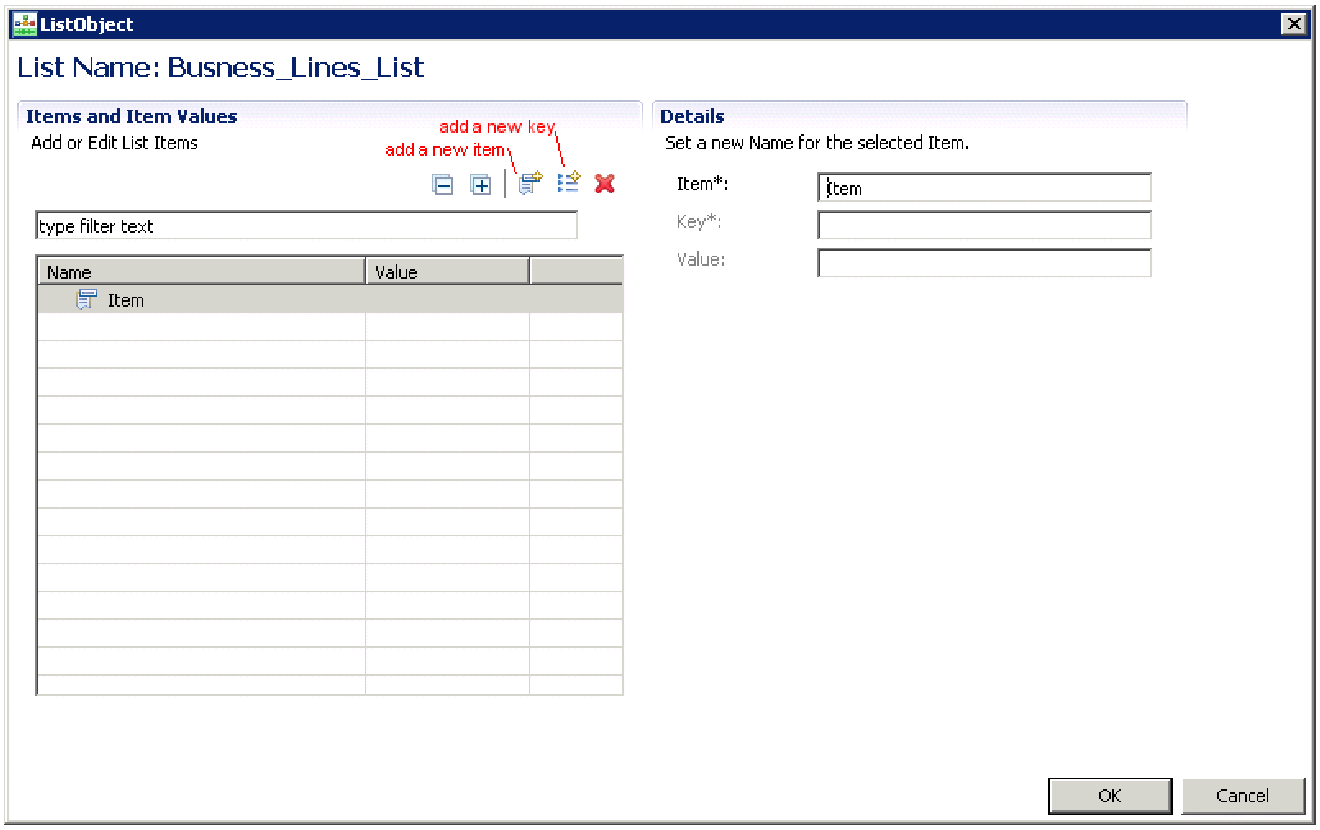
1. If you have not already done so, connect to Configuration Server.
2. From the Window menu, select **Show View** > **List Objects Manager**. You can customize the Show View menu to show List Objects Manager: **Window** > **Customize Perspective**.
3. Right-click the Transaction folder, select **New Folder**, and name the folder.



Name	Description	Access
Transactions		**RHD**
BusinessLinesSample		**RHD**
business_lines_sample	contains ... e agen...	RHD
Discover		
MC		
VISA		
SpecialOffer		**RHD**
BankMailingProductA		RHD
BronzeCustomer		
GoldCustomer		
SilverCustomer		

1. Right-click the new folder and select **New List**.
2. In the Create New List Object dialog box, name and describe the List object and click **OK**.

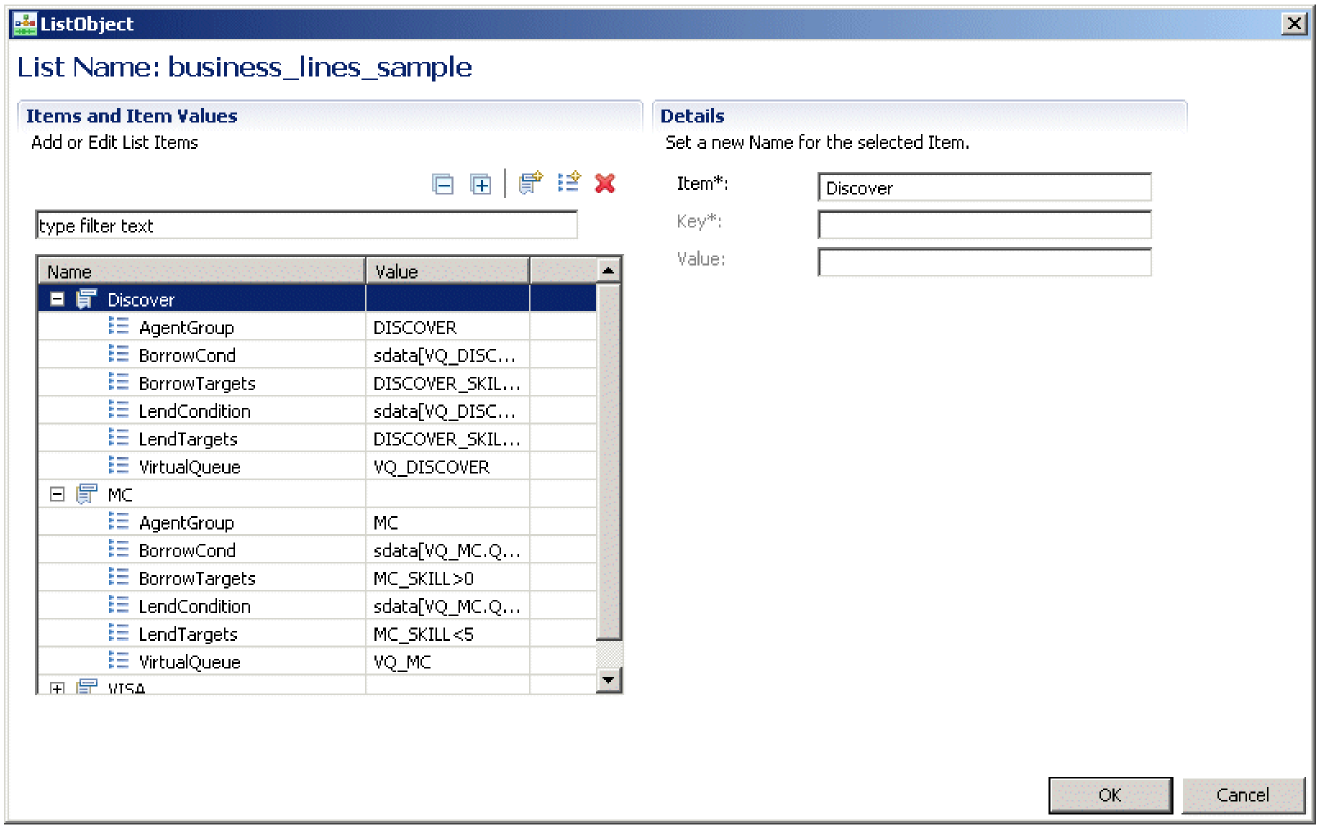
3. Right-click the name of the List object and select **New List Item**. The dialog box for defining a List item opens.



1. On the toolbar, click the button to add a new item.
2. Under **Details**, enter a name in the Item field and click **OK**.
3. Continue adding rows in the List object in this fashion.
4. When through adding rows, in the List Object Managers tab, double-click a row.
5. When the row is selected in the List Object dialog box, click the button to add a new key.
6. Under **Details**, enter the **Key** and **Value** fields.
7. Click **OK**.
8. Continue adding key-values to rows in this fashion.

Sample

Besides individual items, parts of expressions (or an entire expression) can be stored outside of a strategy inside a List object. The sample below is a List object that contains routing information that URS can use to decide when to borrow and lend agents among business lines.



Item*:	Discover
Key*:	
Value:	

Name	Value
Discover	
AgentGroup	DISCOVER
BorrowCond	sdata[VQ_DISC...
BorrowTargets	DISCOVER_SKIL...
LendCondition	sdata[VQ_DISC...
LendTargets	DISCOVER_SKIL...
VirtualQueue	VQ_DISCOVER
MC	
AgentGroup	MC
BorrowCond	sdata[VQ_MC.Q...
BorrowTargets	MC_SKILL>0
LendCondition	sdata[VQ_MC.Q...
LendTargets	MC_SKILL<5
VirtualQueue	VQ_MC
VISA	

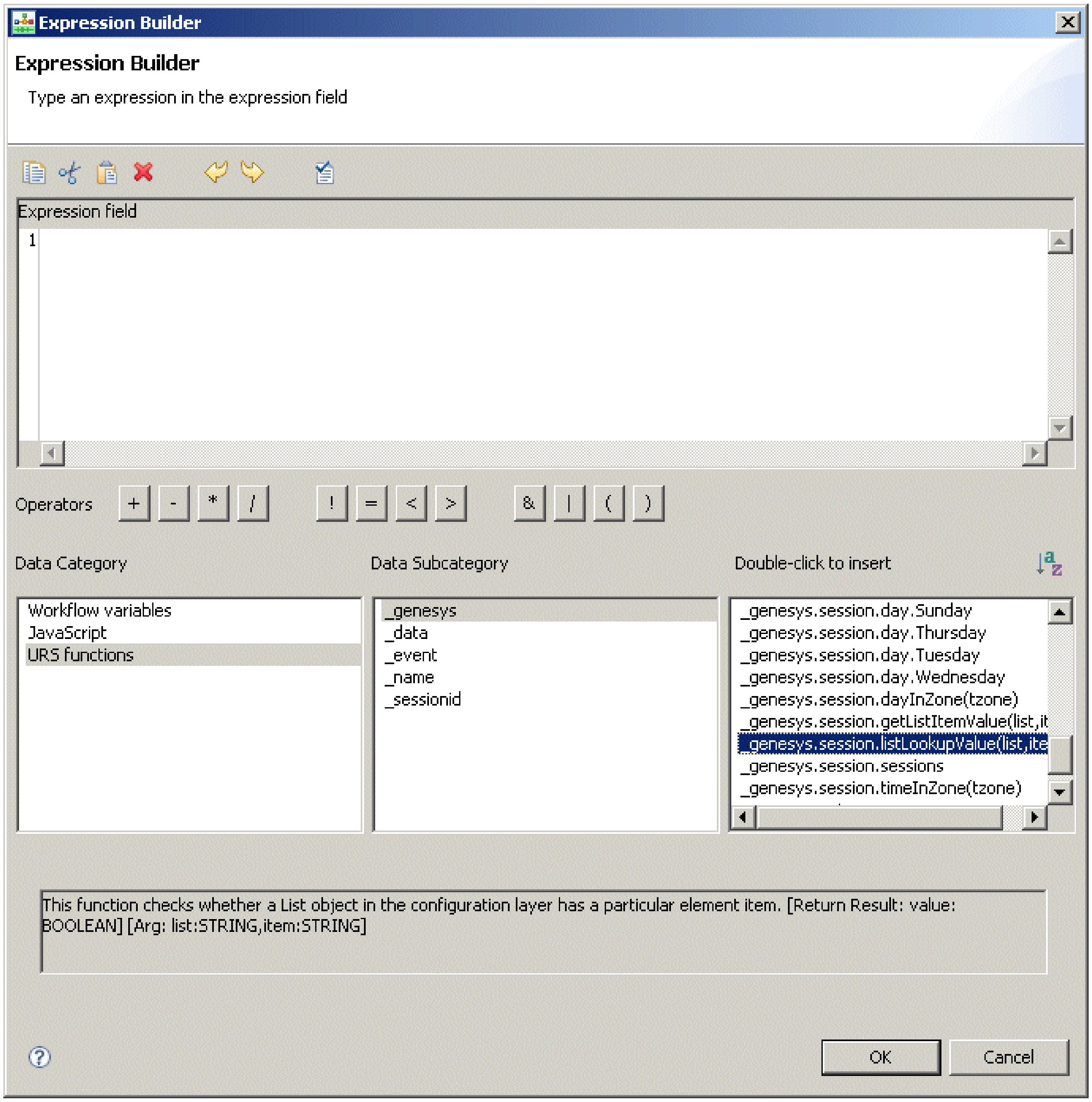
In this sample:

- An IVR has identified customers as wanting information on the MC, VISA, or DISCOVERY business line.
- The requested business line information has been passed to the calling strategy.
- The strategy has segmented interactions to take different paths based on the requested business line.
- If all agents serving the requested business line are busy, URS can use information in the List object to borrow agents from other business lines.

Expression Builder Functions for List Objects

In Expression Builder, two URS functions can be used to access List Objects:

- `_genesys.session.listLookupValue`
- `.genesys.session.getListItemValue`


Expression Builder
Expression Builder
Type an expression in the expression field
Expression field
Operators
+
-
*
/
!
=
<
>
&
|
(
)
Data Category
Data Subcategory
Double-click to insert
Workflow variables
JavaScript
URS functions
_genesys
_data
_event
_name
_sessionid
_genesys.session.day.Sunday
_genesys.session.day.Thursday
_genesys.session.day.Tuesday
_genesys.session.day.Wednesday
_genesys.session.dayInZone(tzone)
_genesys.session.getListItemValue(list,it
_genesys.session.listLookupValue(list,ite
_genesys.session.sessions
_genesys.session.timeInZone(tzone)
OK
Cancel

This function checks whether a List object in the configuration layer has a particular element item. [Return Result: value: BOOLEAN] [Arg: list:STRING,item:STRING]

Refer to *Universal Routing SCXML API Reference* help file for details on these functions (Help > Contents).

Common Blocks

This section describes the blocks that can be used for both Orchestration routing and GVP voice applications. Composer provides the following Common blocks:

- **Context Services Blocks**. If the Context Services capability is enabled at your site, you can use the following Context Services blocks to create callflows/workflows that extract customer data elements from the UCS Database and apply this knowledge when creating routing or voice applications.
- **Assign Block**. Use to assign a computed value/expression or an entered value to a variable.
- **Branching Block**. Use as a decision point in a callflow or workflow. It enables you to specify multiple application routes based on a branching condition. Depending on which condition is satisfied, the call follows the corresponding application route.
- **Log Block**. Use to record information about an application. For example, you can log caller-recorded input collected while an application is running or error messages.
- **Looping Block**. Use to iterate over a sequence of blocks multiple times.
- **Server-Side Blocks**. These 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.

Context Services Common blocks

If the Context Services capability is enabled at your site, you can use the following Context Services blocks to create callflows/workflows that extract customer data elements from the UCS Database and apply this knowledge when creating routing or voice applications.

- **Associate Service Block** to associate an anonymous service record with a customer whose profile exists in the database used for Context Services.
- **Complete Service Block** to mark an active service as completed in the Universal Contact Server Database used for Context Services. You can also use this block to pass additional state-related data to the database. For example, when completing a service state, such as "Callback pending," the disposition could be "unsuccessful - no answer."
- **Complete State Block** to mark the completion of a specified state in the context of a service in the database used for Context Services.
- **Create Customer Block** to create a callflow/workflow that includes the capability to create a customer profile through Context Services.
- **Complete Task Block** to mark the application as completing a specified task within a service/state.
- **Enter State Block** to mark the entry of the application into a specified state in the context of a service.
- **Identify Customer Block** to identify a customer in the database based on search criteria, which can be customer profile core data or customer profile extension data. If the customer is found, then Context Services can provide data that can be used to personalize offer or to resume/modify a service in process.
- **Query Customer Block** to look up a customer's core profile and profile extension attributes. Use the CustomerID to specify which customer you want to return data for. Composer stores the returned results in an application variable.
- **Query Services Block** to query the Universal Contact Server Database for a list of services associated with a particular Customer ID or, in case of unassociated services, the Contact Key. Composer stores the result in an application variable. You can query for active services, completed services, both active and completed services.
- **Query States Block** to query the database used for Context Services for active and completed states data for a specified service. You can also query for other types of service states such as user-defined service states.
- **Query Tasks Block** to query the Universal Contact Server Database used for Context Services for active and completed tasks within a state for a specified service.
- **Start Service Block** to communicate the creation or start of a service in the UCS Database. The service may or may not be immediately associated with a customer. For example, an application, such as a routing workflow, may not know the customer's identity when the service is started so the service may be started anonymously. Once the customer is known and identified, the workflow may associate the anonymous service with the customer.
- **Start Task Block** to mark the application as entering a specified task within a service/state.
- **Update Customer Block** to update the customer profile in the database used for Context Services. You can update customer profile core data or customer profile extension data.

Associate Service Block

Use the Associate Service block to associate an anonymous service record with a customer whose profile exists in the database used for Context Services.

Note: This method can be called more than once in a callflow/workflow. However, subsequent invocations override earlier associations. Therefore, if a service, that is associated with a customer, is again associated with a different customer, the earlier association will be replaced and the service will only be associated with the customer specified in the latest call.

The Associate Service block has the following properties:

Note! The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Customer ID Property

Click the down arrow under Value and select a variable that contains the Customer Identifier for the anonymous service.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Type Property Common Properties Context Services.

Estimated Duration Property

Use this property to specify the estimated service duration (in seconds).

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Estimated dialog box.

3. Select **Literal** or **Variable** from the Type dropdown menu.
    - If you select Literal, enter the estimated service duration in seconds.
    - If you select Variable, select the name of the variable.
4. Click **OK** to close the dialog box.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

Service Type Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Complete Service Block

Use this block in a callflow/workflow to mark an active service as completed in the Universal Contact Server Database used for Context Services.

You can also use this block to pass additional state-related data to the database. For example, when completing a service state, such as "Callback pending," the disposition could be "unsuccessful - no answer."

The Complete Service block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Disposition Code Property

Find this property's details under Common Properties Context Services.

Disposition Description Property

Find this property's details under Common Properties Context Services.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Complete State Block

Use this block in a callflow/workflow to mark an active service as completed in the Universal Contact Server Database used for Context Services.

You can also use this block to pass additional state-related data to the database. For example, when completing a service state, such as "Callback pending," the disposition could be "unsuccessful - no answer."

The Complete Service block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Disposition Code Property

Find this property's details under Common Properties Context Services.

Disposition Description Property

Find this property's details under Common Properties Context Services.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Create Customer Block

Use to create a callflow/workflow that includes the capability to create a customer profile through Context Services. Example use cases:

- The application queried Context Services, which did not have a record for this customer.
- The application wants to create customer preference data (last agent used, language preference, preferred agent, contact media preference and ordering, contact address information, etc.) to optimize and personalize the any future processing associated with the customer.

In order to use this block in a callflow, you must have Media Control Platform (MCP) 8.1.300.76+ installed. The Create Customer block has the following properties. The behavior of some properties can vary depending on whether you are in online or offline mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Profile Data Property

Use this property to enter key-value pairs corresponding to the customer profile.

1. Click under Value to display the button.
2. Click the button to open the Configure Profile Data dialog box.
3. Click Add to open the Add Extension dialog box.
4. Click the down arrow, select **core** or an extension, and click **OK**. The Configure Profile Data dialog box adds **Name** and **Value** fields and a second **Ad**d button.
5. Click the second **Add** on the right to open the Add Attribute dialog box.

- If you selected core, select a customer profile predefined value for Attribute. Opposite **Type**, select literal if you wish to enter the value now or variable if the value is contained in a variable. Opposite Value, enter the value or select a variable and click **OK**.

- If you selected a customer profile extension, select a user-defined extension for Attribute. Opposite Type, select literal if you wish to enter the value now or variable if the value is contained in a variable. Opposite **Valu**e, enter the value or select a variable and click **OK**.

The **Name** and **Value** fields in the Configure Profile Data dialog box reflect your entries.

1. Click **Add** again to continue entering customer attributes in this fashion.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Customer ID Property

Click the down arrow under Value and select a variable that contains the Customer Identifier for the anonymous service.

Complete Task Block

Use this block in a callflow/workflow to mark the application as completing a specified task within a service/state. The Complete Task block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Disposition Code Property

Find this property's details under Common Properties Context Services.

Disposition Description Property

Find this property's details under Common Properties Context Services.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

State ID Property

Find this property's details under Common Properties Context Services.

Task ID Property

Click the down arrow under Value and select a variable that contains the ID of the task.

Enter State Block

Use the Enter State block to mark the entry of the application into a specified state in the context of a service.

Service State Definition

Throughout their lifecycle, business services to customers proceed through a series of well-defined states (see list below). In order to personalize and properly orchestrate a service to a customer, Universal Contact Server must record when each of these predefined states is entered. For example, a company might have the following service states:

- Customer identification
- Service identification
- Assign service agent
- Waiting for service agent
- Offering another service while waiting for agent
- Offering callback
- Callback pending
- Delivering service
- Waiting for customer input
- Offering another service while delivering service

To implement these states you could create a State Business Attribute and map the above State.types to it. The list of Business Attribute values will then be available from the State Type property of this block. Note: The exact sequence of states depends entirely on the way in which the customer service application (such as an IVR or Agent Desktop application) is written.   The Enter State block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

State ID Property

Find this property's details under Common Properties Context Services.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Estimated Duration Property

Find this property's details under Common Properties Context Services.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

Previous State ID Property

Use this property to specify the ID of the state that came before this one.

1. Click under Value to display the button.
2. Click the button to open the Previous State ID dialog box.
3. Select **Literal** or **Variable** from the **Type** dropdown menu.

- If you select **Literal**, enter the Previous State ID associated with the service.
- If you select **Variable**, select a variable that contains this information.

State Type Property

Use this property to filter for specific service state types.

1. Click under Value to display the button.
2. Click the button to open the State Type Selection dialog box.
3. Select one of the following from the **Type** dropdown menu:

- Context Services. Select the State Type identifier for **Value**. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a State Type DB ID.
- **Literal**. Enter the State Type ID.
- Variable. Select the variable that contains the State Type ID.

Identify Customer Block

Use this block to identify a customer in the database based on search criteria, which can be Profile Core customer profile data or customer extension data. If the customer is found, then Context Services can provide data that can be used to personalize offer or to resume/modify a service in process.

Note: Prior to using this block, set Context Services Preferences. For detailed information on how Universal Contact Server identifies customers, see the Context Services User's Guide. For an example of how to use this block, see the Getting and Using E-mail Addresses topic.

The Identify Customer block has the following properties. The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Get Attributes Property

Use this property to control whether only matching Customer IDs are returned (No) or whether all profile attributes are returned (Yes).

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Get Attribute dialog box.
3. Opposite **Type**, click the down arrow and select one of the following:

- **Variable**. Then click the **Value** down arrow and select the name of the variable.
- **Literal**. Then for **Value**, select **Yes** or **No**.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

You can also define custom events.

Suppress Customer Not Found Exception Property

- If set to true, no exception will be raised when no customer is found. The Customer Data array will be empty. See the Customer Data property below.

- If set to false (default), the error.com.genesyslab.composer.customernotfound exception is raised when no matching customer is found.

Customer Attributes Property

Use this property to specify a list of attributes which will be used to search for the customer. To specify customer attributes:

1. Click under Value to display the button.
2. Click the button to open the Customer Attributes dialog box.
3. Click **Add** in the dialog box to open the Customer Attribute dialog box.
4. Opposite **Extension**, click the down arrow and select either **Core** (for customer profile core data) or a customer profile extension.

    - If you select **Core**, select a core attribute from the **Attribute** dropdown menu. For example, you might select Core and then CustomerSegment.
    - If you select an extension, select the extension attribute name from the **Attribute** dropdown menu.

Note: In offline mode, for both core and extension data, there is an additional field ‘’Attribute Type” where you must choose between string, Boolean, integer, long, double, date, datetime, or currency depending on the customer profile attribute definition.

5. Click the down arrow opposite Type and select **Literal** or **Variable**.

    - If you select **Literal**, then for **Value**, enter the value of the attribute. For example, if you are looking for a customer having LastName=Rosen, you would key in Rosen.

- If you select **Variable**, select the variable under **Value**.

6. Click OK to close the Customer Attribute dialog box. The Customer Attributes dialog box reflects your entries. You can also use this dialog box to edit and remove entries.

Identification Key Property

Use this property to specify the name of the key to be used for lookups. If specified, speeds the lookup. The key can be contained in a variable.

1. Click under **Value** to display the button.
2. Click the button to open the Identification Key dialog box.
3. Opposite **Type**, click the down arrow and select one of the following:
    - **Context Services**. Then click the *Value* down arrow and select the key. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select an Identification Key name for Value.
    - **Variable**. Select the name of the variable.
    - **Literal**. Enter the name of the key.

Customer Count Property

Click the down arrow under Value to select a variable whose value is the number of customer records returned by Universal Contact Server. This feature is for your convenience. It also serves the purpose of retaining the original number of records returned in case the returned data is modified through other blocks.

Customer Data Property

Click the down arrow under Value to select a variable whose value will be the JSON array containing data returned.

If no matching customers are found, an empty array is returned and an exception will be thrown unless the Suppress Customer Not Found Exception property is set to true.

Variables Mapping Property

Variables Mapping Property Use this property to map the JSON data returned by this block to variables. See the Variables Mapping topic for details.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Query Customer Block

Use this block to look up a customer's core profile and profile extension attributes. Use the CustomerID to specify which customer you want to return data for. Composer stores the returned results in an application Property variable. For an example of how to use this block, see the Getting and Using E-mail Addresses topic.

The Query Customer block has the following properties. The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

You can also define custom events.

Note: The error.com.genesyslab.composer.customernotfound exception is raised if no customer with the specified customer ID is found.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Customer ID Property

Click the down arrow under Value and select a variable to specify the Customer Identifier.

Include Extensions Property

Select a variable or from the list of extensions to specify which customer profile extension attributes are returned as part of the query operation.

1. Click under **Value** to display the button.
2. Click the button to open the Extensions dialog box.
3. Click **Add** to open the Extension dialog box.
4. Opposite **Type**, click the down arrow and select one of the following:
    - **Context Services**. Select an extension attribute already defined in the database for Value. Note: Composer supports multi-valued extensions starting with Universal Contact Server 8.0.2.
    - **Variable**. Select the variable that contains the extension.
    - **Literal**. Enter the name of the extension attribute.
5. Click **OK**. The Extensions dialog box lists the extension attribute or the name of the selected variable. You can also use this dialog box to edit and remove extensions.

Result Property

Click the down arrow and select a variable whose value contains the JSON data returned by the Context Services web service. These results will then be available in other blocks in the application for further processing.

Any post processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It already supports writing simple or complex expressions to extract values out of JSON strings and arrays.

Variables Mapping Property

Find this property's details under Mapping Common Properties Context Services.

Query Services Block

Use this block to query the Universal Contact Server Database for a list of services associated with a particular Customer ID or, in case of unassociated services, the Contact Key. Composer stores the result in an application variable. You can query for:

- Active services
- Completed services
- Both active and completed services

Use Case

Service/state history is primarily meant to support service personalization and resumption. For example, a given application is using a state to record the fact that the system is waiting for the customer to fax in a signed authorization to complete a transaction. When the customer calls into the IVR to verify some recent activities, the IVR application queries the service state history and is informed that a service is waiting on a fax to arrive. The Query Services block has the following properties: Note! The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Service Elements Property

Use this property to indicate whether information on completed service elements/tasks should be included in the returned results.

1. Click under **Value** to display the ... button.
2. Click the ... button to open a dialog box.

3. Check one or more of the following:
    - **Active Services**
    - **Completed Services**
    - **Active Tasks**
    - **Completed Tasks**

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Service Data Property

Click the down arrow and select a variable to contain the output data for matching services. The output of the Query Service block will be in JSON format. You will need to use the Assign block and EMCAScript in Expression Builder to parse the JSON. Below is an example of a small ECMAScript that will take the output of the Query Service block, which has been put into an application variable "QueryServices," and retrieves the service_id for the customer, if it any active services exist. It puts that service_id value into a workflow application variable called "serviceid". `if (QueryServices.length > 0) { serviceid = QueryServices[0].service_id; } else { serviceid = 'new service'; }` **Note**: You can also do the same things with Composer's Variables Mapping feature. It is easier (and it saves an ECMAScript block) to simply use the Variables Mapping property to map "service_id" to the application variable "serviceid".

Variables Mapping Property

Use this property to map the JSON data returned by this block to variables. See the Variables Mapping topic for details.

Identifier Property

Use this property to identify the customer. Choose the Customer ID (for associated services) or the Contact Key (for unassociated services).

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the Identifier dialog box.
3. Select one of the following buttons:

- **Customer Identifier** (for associated services).
- **Contact Key** (for anonymous services)

1. Click the down arrow opposite **Type** and select the source:

- **Literal**. Then enter the attribute, such as CustomerID.
- **Variable**. Then select the variable that contains the Contact Key or Customer ID.

Service Status Property

Use this property to choose the Composer method to call.

1. Click under **Value** to display the ⋯ button.

2. Click the button to open the Service Status Selection dialog box.
3. Opposite Type, select one of the following:

    - **Variable**. Then click the **Valu**e down arrow and select a variable with a value of "Completed", "Active", or "All".
    - **Literal**. Then select one of the following:**Completed'**, *Active*, '**All**

Service Type Property

Find this property's details under Common Properties Context Services.

Service Completed After Property

Use this property to filter for services completed on or after the given date/time.

1. Click under **Value** to display the button.
2. Click the button to open the Service Completed After dialog box.
3. Click the down arrow opposite **Type** and select one of the following:

    - **Literal**. Click arrows to select the Date and Time or manually enter.
    - **Variable**. Select the variable whose value must be formatted with the ISO 8601 UTC pattern YYYY-MM-DDTHH:mm:ss.SSSZ (for example: 2010-03-15T11:33:48.000Z).

Also see the Time Zone Preferences topic.

Service Completed Before Property

Use this property to filter for services completed prior to the given date/time. If using a variable for this property, the variable value must follow the ISO 8601 UTC pattern: [YYYY]-[MM]-[DD]T[HH]:[mm]:[ss].[SSS]Z.

1. Click under **Value** to display the button.
2. Click the button to open the Service Completed Before dialog box.
3. Click the down arrow opposite **Type** and select one of the following:

    - **Literal**. Click arrows to select the Date and Time or manually enter.
    - **Variable**. Select the variable whose value must be formatted with the ISO 8601 UTC pattern YYYY-MM-DDTHH:mm:ss.SSSZ (for example: 2010-03-15T11:33:48.000Z).

Also see the Time Zone Preferences topic.

Service Started After Property

Use this property to filter for services started on or after the given date/time.

1. Click under **Value** to display the button.
2. Click the button to open the Service Started After dialog box.
3. Click the down arrow opposite **Type** and select one of the following:
   - **Literal**. Click arrows to select the Date and Time or manually enter.
   - **Variable**. Select the variable whose value must be formatted with the ISO 8601 UTC pattern YYYY-MM-DDTHH:mm:ss.SSSZ (for example: 2010-03-15T11:33:48.000Z).

Also see the Time Zone Preferences topic.

Service Started Before Property

Use this property to filter for services started prior to the given date/time.

1. Click under Value to display the button.
2. Click the button to open the Service Started Before dialog box.
3. Click the down arrow opposite **Type** and select one of the following:
   - **Literal**. Click arrows to select the Date and Time or manually enter.
   - **Variable**. Select the variable whose value must be formatted with the ISO 8601 UTC pattern YYYY-MM-DDTHH:mm:ss.SSSZ (for example: 2010-03-15T11:33:48.000Z).

Query States Block

Use this block to query the database used for Context Services for active and completed states data for a specified service. You can also query for other types of service states such as user-defined service states. The Query States block has the following properties. The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

State Elements Property

Use this property to indicate whether information on completed states and/or active tasks for this service state should be included in the returned results.

1. Click under **Value** to display the button.
2. Click the button to open a dialog box.
3. Check one or more of the following:
    - **Active Tasks**
    - **Completed Tasks**

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

States Data Property

Click the down arrow under **Value** and select a variable to contain the matched state information.

Variables Mapping Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

State Status Property

This property controls whether Active, Completed, or All States are returned.

1. Click under **Value** to display the button.
2. Click the button to open the State Status dialog box.
3. Opposite **Type**, select **Variable** or **Literal**.
    - If you select **Variable**, click the **Value** down arrow and select a variable that contains the name of the method to call.
    - If you select **Literal**, click the **Value** down arrow and select one of the following: **Completed**, **Active**, or **All**.

State Types Property

Use this property to filter for other service state types, such as user-defined service states.

1. Click under Value to display the button.
2. Click the button to open the State Types dialog box.
3. Click **Add** to open the Add dialog box.
4. Opposite **Type**, click the down arrow and select one of the following:
    - **Context Services**. Select a State Types identifier for Value. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a State Types DB ID. If no Business Attribute is mapped in the UCS configuration, then UCS accepts any integer value which could represent a state type defined in an external client-specific database.
    - **Literal**. Then enter a preconfigured state type from the Configuration Database.
    - **Variable**. Then select the variable that contains the state type.
5. Click **OK**. The State Types dialog box reflects your entry. You can also use this dialog box to edit and remove state types.

Query Tasks Block

Use this block to query the Universal Contact Server Database used for Context Services for active and completed tasks within a state for a specified service.

The Query Tasks block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Task Data

Click the down arrow and select a variable to hold the output data for matching tasks.

Variables Mapping Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

State ID Property

Click the down arrow under Value and select a variable that contains the ID of the newly entered/created state.

Task Status Property

This property controls whether Active, Completed, or All tasks are returned.

1. Click under Value to display the button.
2. Click the button to open the Task Status dialog box.
3. Opposite **Type**, you can:
    - Select **Variable** and select a variable that contains the name of the method to call.
    - Select **Literal** and select one of the following: **Completed**, **Active**, or **All**

Task Type Property

Use this property to filter for specific task types.

1. Click under Value to display the button.
2. Click the button to open the Task Types dialog box.
3. Click **Add** to open the Add dialog box.
4. Opposite Type, click the down arrow and select one of the following:
    - **Context Services**. Select a Task Type identifier for Value. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a Task Types DB ID. If no Business Attribute is mapped in the UCS configuration, then UCS accepts any integer value which could represent a task type defined in an external client-specific

database.

- **Literal**. Then enter a pre-configured task type from the Configuration Database.
- **Variable**. Then select the variable that contains the task type.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Start Service Block

Use this block to communicate the creation or start of a service in the Universal Contact Server (UCS) Database. The service may or may not be immediately associated with a customer. For example, an application, such as a routing workflow, may not know the customer's identity when the service is started so the service may be started anonymously. Once the customer is known and identified, the workflow may associate the anonymous service with the customer.

Service Definition

Data residing in the UCS Database includes service data. A service may be seen as a communication or series of communications between a customer and an enterprise, and possibly also between various parts of the enterprise. A service has a temporal beginning and end. It may span multiple interactions and include interactions of various media types (voice, e-mail, and so on).

The scope of a given service is completely defined by your enterprise and the way its customer service applications are written (for example, an IVR or Agent application).

States and Tasks

As described in the [{Context Services User's Guide]], services are composed of any number of states, and states in turn can be composed of any number of tasks. Services, states, and tasks are each different types of Service Elements, which exist over an application-defined lifecycle, and have business context attached to them in the form of a disposition. Within the database, data for Service Elements is constructed based on a sequence of one or more Service Events received from an application, such as a routing workflow.

The Start Service block has the following properties:

Note! The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

You can also define custom events.

Service ID Property

Find this property's details under Common Properties Context Services.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Estimated Duration Property

Use this property to specify the estimated service duration (in seconds).

1. Click under Value to display the button.
2. Click the button to open the Estimated dialog box.
3. Select **Literal** or **Variable** from the Type dropdown menu.
    - If you select **Literal**, enter the estimated service duration in seconds.
    - If you select **Variable**, select the name of the variable.

Identifier Property

Use this property to identify the customer. Choose the Customer ID (for associated services) or the Contact Key (for unassociated services).

1. Click under Value to display the button.
2. Click the button to open the Identifier dialog box.
3. Click one of the following buttons:
    - **Customer Identifier** (for associated services)
    - **Contact Key** (for anonymous services)
4. Click the down arrow opposite **Type** and select the source: **Literal** or **Variable**.
    - If you select **Literal**, enter an attribute for Value, such as CustomerID.
    - If you select **Variable**, select the variable for Value that contains either the Contact Key or the Customer ID.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Service Type Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Start Task Block

Use this block to mark the application as entering a specified task within a service/state.   The Start Task block has the following properties. The behavior of some properties will vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Extensions Property

Find this property's details under Common Properties Context Services.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for strategy blocks. You can also define custom events.

Task ID Property

Click the down arrow under Value and select a variable that contains the ID of the task.

Application ID Property

Find this property's details under Common Properties Context Services.

Application Type Property

Find this property's details under Common Properties Context Services.

Estimated Duration Property

Find this property's details under Common Properties Context Services.

Media Type Property

Find this property's details under Common Properties Context Services.

Resource ID Property

Find this property's details under Common Properties Context Services.

Resource Type Property

Find this property's details under Common Properties Context Services.

Use Server Timestamp Property

Find this property's details under Common Properties Context Services.

Service ID Property

Find this property's details under Common Properties Context Services.

State ID Property

Click the down arrow under Value and select a variable that contains the ID of the state.

Task Type Property

Use this property to filter for specific task types.

1. Click under Value to display the … button.
2. Click the … button to open the Task Types dialog box.
3. Click **Add** to open the Add dialog box.
4. Opposite **Type**, click the down arrow and select one of the following:

    - **Context Services**. Select a Task Type identifier for Value. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a Task Types DB ID. If no Business Attribute is mapped in the UCS configuration, then UCS accepts any integer value which could represent a task type defined in an external client-specific database.
    - **Literal**. Then enter a pre-configured task type from the Configuration Database.
    - **Variable**. Then select the variable that contains the task type.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Update Customer Block

Use this block to update the customer profile in the database used for Context Services. You can update customer profile Profile Core core data or customer profile extension data. Composer supports multi-valued extensions starting with Universal Contact Server 8.0.2. For more information on Context Services attributes and extensions, see the *Context Services User's Guide*.

Note: Place this block after the Query Customer block after you place the results of the customer query in a variable. This applies if you want to update some individual customer attributes and keep the other attributes at their original values. Otherwise, to remove all older attribute values, you do not need to use a Query Customer block and the Profile Data Variable property can remain not set.

Note! You may wish to configure Context Services Preferences. The behavior of some properties can vary depending on whether you are in offline or online mode.

The Update Customer Profile block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Profile Data Variable

Click the down arrow under Value and select the variable that contains the customer profile data for the update operation. A Query Customer block can be used to initialize this variable.

Profile Data Property

Use this property to specify the core or extension customer profile data.

1. Click under **Value** to display the ⋯ button.
2. Click the ⋯ button to open the Profile Data dialog box. The first time this dialog box appears, an **Add** button appears on the left side only.

3. Click **Add** to open the Profile Attribute dialog box.
4. Click the down arrow and select either **Core** (for customer profile core data) or a customer profile extension.
5. Click **OK** to close the Profile Extension dialog box. The Profile Data dialog box adds a second column with a second **Add** button.
6. With Core or the customer profile extension still highlighted in the first column, click the **Add** button in the second column. The Customer Attribute dialog box opens. Do one of the following:
    - Opposite **Attribute**, select the attribute (core or extension depending on what you previously selected). In offline mode, there is an additional field ‘’Attribute Type” in this dialog where you must choose between **string**, **Boolean**, **integer**, **long**, **double**, **date**, **datetime**, or **currency** depending on the customer profile attribute definition.
    - Opposite **Type**, select **Literal** or **Variable**.
    - Opposite **Value**, enter the literal or select the variable.
    - Click **OK** to close the dialog box. The Profile Data dialog box reflects your entry.
7. Repeat these steps if you wish to update another attribute.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for strategy blocks.

You can also define custom events.

Customer ID Property

Click the down arrow under Value and select a variable to specify the Customer Identifier.

Operation Property

Use this property to select the block’s operation. Click the down arrow and select one of the following:

- **Update** to update an existing customer profile.
- **Insert Extension** to insert extensions records to an existing customer profile.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Using Context Services Blocks

This section describes how to use the Context Services blocks.

Context Services and Composer

Using Context Services

Context Services refers to an optional capability of Universal Contact Server and its Universal Contact Server (UCS) Database, a repository of customer-related Definition service, and interaction-centric data (current and historical) from Genesys and third party sources. You can use the Context Services blocks for:

1. **Service personalization**. You can create callflows/workflows that alter the customer experience based on information known about the customer.
2. **Offer personalization**. When managing conversations, routing workflows can use the results of previous offers made to the customer to decide whether a new offer should be presented.
3. **Service resumption**. Workflows can leverage service state/task information to continue a customer service that was not completed in an earlier interaction.
4. **Enhanced reporting**.

A Context Services Project template is included with this release.

You can access Context Services attributes through Expression Builder.

Customer Profile Core

As described in the *Context Services User's Guide*, the customer profile contains a core set of customer characteristics available "out-of-the-box" from Context Services. Use the Query Customer block to request (and assign to variables) the following types of customer profile information:

- **Last agent used**. Allows an application to route an interaction to the agent that last processed the last interaction for this customer.
- **Language preference**. Allows an application to use the customer’s preferred language when communicating with them. This can be used in a voice Self-Service dialog, in e-mail responses, or in routing to an agent that speaks the language.
- **Preferred agent**. Allows an application to route to the customer's preferred agent if available.
- **Contact media preference**. Allows the application to use the customer preferred media when sending notifications or initiating any outbound contact with the customer. For example, use e-mail first, home phone next, commonly used web pages, and so on.

Customer Extension Data

You can extend customer profile core data with specific types of data that your business wants to include (Create Profile Extension message). For example, assume your business introduces an automated newsletter. You then might wish to add a profile extension to record customer preferences for receiving the newsletter, along with the preferred e-mail format (text or html).

Use the Update Customer Block block to update customer profile extension data.

In order for extension data to be available for selection in Composer, the data must already be defined for UCS Context Services using its HTTP interface.

Note: Composer supports multi-valued extensions starting with Universal Contact Server 8.0.2.

Identification Keys

One of the core features of the Context Services API is the ability to identify customers based on one or more attributes of the customer, known as Identification Keys. Each identification key consists of one or more attributes of the core customer profile, or of any defined extension. An attribute must be specified as an Identification Key to be usable in customer identification.

Authentication Support

Composer supports username/password authentication for VXML and SCXML-generated applications.

For Orchestration applications, username and password values are configured in the EnhancedRouting ScriptConfiguration Server object: (`ApplicationParms/context_management_services_password` and `ApplicationParms/context_management_services_username`).

For design-time access to Context Services, you can specify these values in Context Services Preferences. Composer can then use them to authenticate with Context Services when it connects to Context Services for retrieving profile objects, extensions, etc. Composer updates values specified in these preferences when diagrams are published to Configuration Server.

For voice applications, username and password values are configured in the Voice Platform IVRProfile Configuration Server object (`gvp.service-parameters/voicexml.cms_username` and `gvp.service-parameters voicexml.cms_password`).

Also see Business Rules Block Runtime Configuration.

Common Properties Context Services

The following properties are common to multiple Context Services blocks. Their descriptions are placed here to minimize duplication of content. The behavior of some properties can vary depending on whether you are in offline or online mode.

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Application ID Property

Use this property to assign a unique ID (for example, a Genesys DB ID) for the Application issuing the anonymous service event (for example, a GVP VoiceXML Application, an Orchestration SCXML Application, and so on).

1. Click under Value to display the button.
2. Click the button to open the Application ID dialog box.
3. From the Type dropdown menu, select one of the following:

- **Literal**. For Value, enter the Application ID.
- **Variable**. For Value, select the name of the variable that contains the ID.

Application Type Property

Use this property to assign a unique ID associated with the type or class of application issuing the completed service event. May be used to group related applications, potentially across resource types.

1. Under Value to display the button.

2. Click the button to open the Application Type dialog box.
3. Select one of the following:

   - **Context Services**. Then select an Application Type idenitifer. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select an Application Type DB ID for Value.
   - **Variable**. Select the name of the variable that contains the Application Type identifier.
   - **Literal**. Enter the Application Type.

Disposition Code Property

Use this property to assign a unique ID for the business disposition assigned to the given service/state. Typically this will be a Configuration Manager Disposition Code Business Attribute. For more information on disposition, see the Context Services User's Guide, . To set this property:

1. Click under Value to display the button.
2. Click the button to open the Disposition Code dialog box.
3. Select one of the following:

   - **Literal**. Enter the Disposition Code for Value.
   - **Variable**. Select the name of the variable that contains the Disposition code.
   - **Context Services**. Select a Disposition code. If Context Servicesattributes have been mapped to Configuration Server Business Attributes, you can select a Disposition code DB ID.

Disposition Description Property

Use this optional property to enter text providing additional context on the business disposition. Limited to 64 characters.

1. Click under Value to display the button.
2. Click the button to open the Disposition Description dialog box.
3. Select one of the following:

   - **Literal**. Then enter the text under Value.
   - **Variable**. Then select the variable under Value.

Extensions Property

Use this property to select customer profile extension attributes to use as part of the search). Note: Composer supports multi-valued extensions starting with Universal Contact Server 8.0.2.

1. Click under Value to display the ⋯ button.
2. Click the ⋯ button to open the Extensions dialog box.
3. Click Add in the Extensions dialog box to open the Add Extension dialog box.
4. Opposite Type, click the down arrow and select one of the following:
    - **Context Services**. For Value, select an extension already defined in the Universal Contact Server Database. If you are not connected to Universal Contact Server, the Value field does not list extensions.
    - **Variable**. For Value, select an extension contained in a variable.
    - **Literal**. For Value, enter the name of the extension attribute.
5. Click OK. The Extensions dialog box lists the extension attribute or the name of the selected variable. You can also use this dialog box to edit and remove extensions.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for strategy blocks. You can also define custom events.

Condition Property

Find this property's details under Common Properties.

Logging Details Property

Find this property's details under Common Properties.

Log Level Property

Find this property's details under Common Properties.

Enable Status Property

Find this property's details under Common Properties.

Estimated Duration Property

Use this property to specify the estimated service duration (in seconds).

1. Click under Value to display the button.
2. Click the button to open the Estimated dialog box.
3. Select from the Type dropdown menu.
    - If you select**Literal**, enter the estimated service duration in seconds.
    - If you select **Variable**, select the name of the variable.

Media Type Property

Used to specify a particular Media Type for the service, which can be a Configuration Server Business Attribute (such as for the Application Type property).

Resource ID Property

Use this property to assign a unique ID for the specific resource-providing service. This might be the Genesys DB ID of a specific GVP or orchestration platform, or the DB ID of a given agent, depending on the context.

1. Click under Value to display the button.
2. Click the button to open the Resource ID dialog box.
3. Select from the Type dropdown menu.
    - If you select **Literal**, enter the unique ID for the resource providing service.
    - If you select **Variable**, select a variable that contains this information.

Resource Type Property

Use this property to assign a unique ID associated with the type or class of resource providing service (for example, GVP, Agent Desktop, Orchestration).

1. Click under Value to display the button.
2. Click the button to open the Resource Type dialog box.
3. Select one of the following from the Type dropdown menu:
   - **Context Services**. Select a Resource Type identifier. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a Resource Type DB ID for Value.
   - **Literal**. Enter the unique ID associated with the type or class of resource.
   - **Variable**. Select a variable that contains this information.

Service ID Property

Click the down arrow under Value and select a variable that contains the ID of the anonymous service.

Service Type Property

Use this property to assign a Service Type code (Business Attribute), which describes what type of service a customer is requesting at a particular moment in time. For example, an IVR system may have the customer select 1 for Loan, 2 for Investment or 3 for Information. Loan, Investment, and Information are all Service Types.

1. Click under Value to display the button.
2. Click the button to open the Service Type dialog box.
3. Click the down arrow opposite Type and one of the following:
   - **Context Services**. Select a Service Type code. If Context Services attributes have been mapped to Configuration Server Business Attributes, you can select a Service Type DB ID for Value.
   - **Literal.** Enter a Service Type code.
   - **Variable**. Select the variable that contains the Service Type code.

State ID Property

Use this property to specify the identifier for the completed state

1. Click under Value to display the button.
2. Click the button to open the State ID dialog box.

3. Select one of the following:
    - **Literal**. Then enter the identifier for the service state under Value.
    - **Variable.** Then select the variable under Value that contains the state identifier.

Use Server Timestamp Property

Use this property to determine if Universal Contact Server should assign the time at which the service was associated with the customer.

- If **True**, the UTC time at which the service event was associated is assigned by the server.
- If **False**, the UTC time is assigned by script embedded in the SCXML application.

1. Click under Value to display the button.
2. Click the button to open the User Server Timestamp dialog box.
3. Select one of the following:
    - **Literal**. Then for Value, select True or False.
    - **Variable**. Then for Value, select a variable that contains true or false.

Variables Mapping Property

Use this property to map the JSON data returned by this block to variables. See the Variables Mapping topic for details.

Online and Offline Modes

Composer support two modes during design time when working with Context Services:

- Online mode (connected to Universal Contact Server)
- Offline mode (not connected to Universal Contact Server)

You specify online or offline mode in Context Services Preferences by checking the box opposite Connect to the Universal Contact Server when designing diagrams and completing the associated fields.

Online Mode

In online mode, you are connected to the Context Services server, which is Universal Contact Server (UCS). In this case, based on Context Services Preferences, Composer accesses a specified server instance during design time, queries for information, and populates dropdown lists and other interface items appropriately. Composer opens a new connection to the server each time some data needs to be fetched. The connection is not kept alive during calls (as is the connection Configuration Server) so no connection status is displayed.

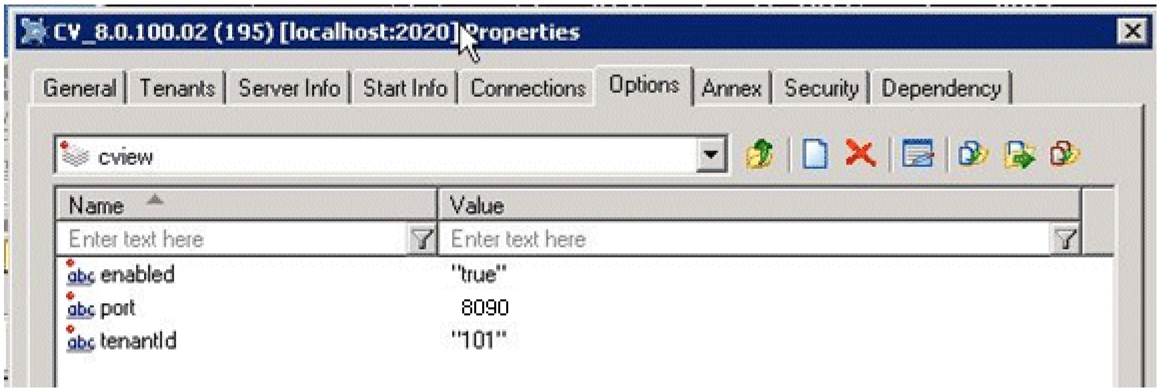
Offline Mode

An offline mode is also supported. In this mode, Composer does not contact the Context Services server during design time. Any dialogs or properties that query the server in online mode revert to an open interface in offline mode and do not show dropdown lists containing database objects. In this case, you must key in literal values or you may select variables if the particular property or dialog supports working with application variables. Note: Both these modes apply to Composer during application design. At runtime, the SCXML/VXML application will contact an instance of UCS.

Setting Context Services Preferences

When working with Context Services blocks, you may wish to use online mode. In this mode, Composer fetches data from Universal Contact Server during design phase to help you configure the blocks. For example, Composer can fetch customer profile attribute names, extension attribute names, and so on. You can enable/disable this behavior in the Context Services Preferences page. If the Context Services capability is enabled at your site, set preferences as follows:

1. Go to **Window** > **Preferences** > **Composer** > **Context Services**.
2. Check the following box to specify online or offline mode when connecting to Context Services: **Connect to the Universal Contact Server when designing diagrams**. This enables the fields below.
3. Under **Universal Contact Server**, enter the server host name in your Configuration Database, which is the name (or IP address) of the Universal Contact Server. Also see the Runtime Configuration topic.
4. Enter the **Server Port number** for Universal Contact Server. **Note:** For the port number, open the Universal Contact Server Application object in your Configuration Database, go to Options tab, select the cview section, and the port option. Example settings are shown below.



Name	Value
enabled	"true"
port	8090
tenantId	"101"

5. Enter the **Base URL** for the Context Services server (UCS).

The GVP Debugger passes all host, port, and base URL parameters to the VXML platform. It uses the parameters to make an url made of: `[http:// http://]<host-parameter>:<port-parameter>[/<base-url>`.

6. Under **Security Settings**, **Use secure connection**, select **never** or **TLS if Transport Layer Security** is implemented as described in the *Genesys 8.1 Security Deployment Guide*. Also see *Debugging Transport Layer Security*.
7. Select **Use Authentication** to require a user name and password when connecting to Universal Contact Server. If selected, enter the User and Password fields.
8. On the Context Services Preferences page, click the **Test Connection** button. Clicking should cause connection successful to appear. If not, check that Universal Contact Server is running and that the entered host/port values are correct. Other sources of error could be:

    - Base URL parameter value is incorrect

- UCS version is not 8.1 or higher

Note: Composer can successfully communicate with UCS at design stage whatever the UCS mode is (production or maintenance). However, UCS needs to be in production mode at runtime stage (when running Context Services SCXML or VXML applications, even when using GVP Debugger).

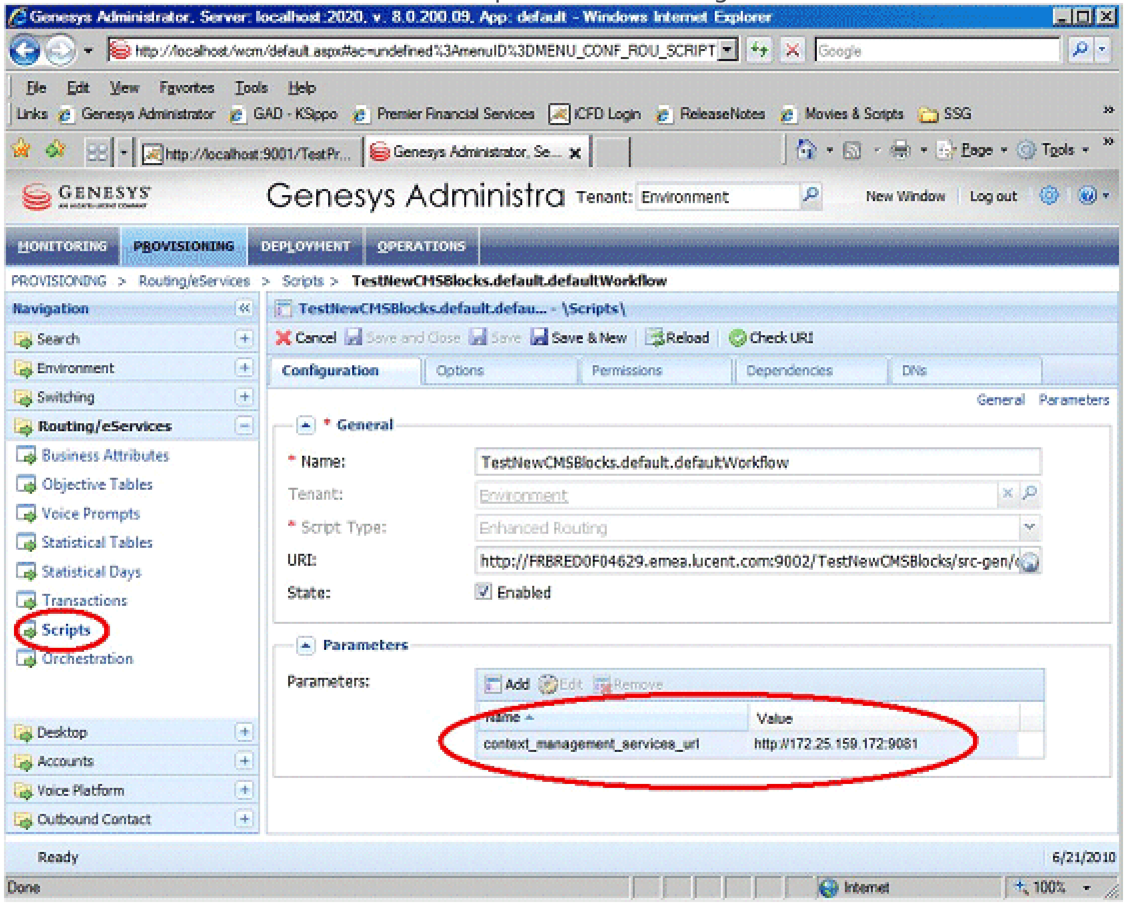
9. Under **Context Services object Validation,** select one of the following:
    - **No validation**
    - **Validate if connected**
    - **Validate**

Runtime Configuration

This topic discusses both Workflow Diagram and Callflow Diagram runtime configuration.

Workflow Diagram Runtime Configuration

When you publish an interaction process diagram, Composer creates an `EnhancedRouting Script` object in Configuration Server. This Script object has a `context_management_services_url` parameter, which is initialized with the UCS server parameters configured in Context Services



General

Name: TestNewCMSBlocks.default.defaultWorkflow
Tenant: Environment
Script Type: Enhanced Routing
URI: http://FRBRED0F04629.emea.lucent.com:9002/TestNewCMSBlocks/src-gen/
State: ☑ Enabled

Parameters

Name	Value
context_management_services_url	http://172.25.159.172:9081

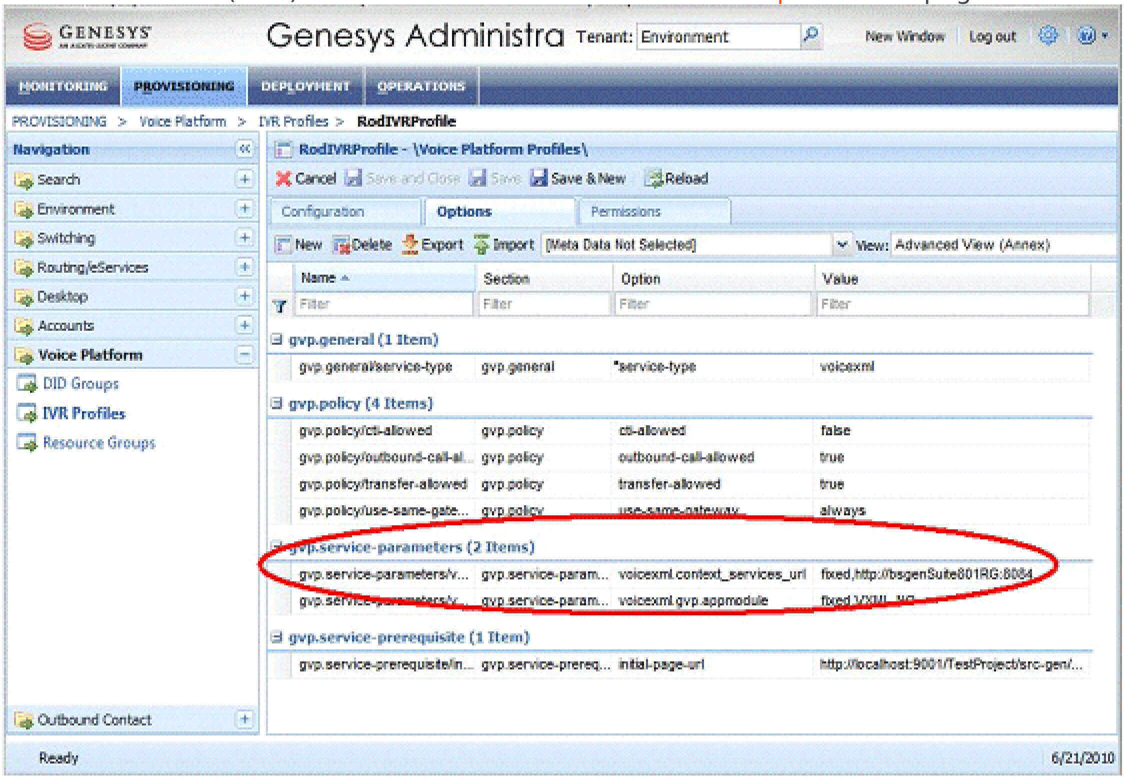
Preferences.
The Script objects are automatically created by Composer and the url is also automatically set if the Project is deployed within Composer (in embedded TOMCAT)

Manual Configuration

If using a DN to trigger the interaction process SCXML application, in the Annex of the DN, you must manually add the following property: `Orchestration/application=script:`*<Name of the Script object as defined above>*

Callflow Diagram Runtime Configuration

Update the IVR Profile to define a `context_services_url` parameter whose value points to the Context Services (UCS) URL defined in the Context Services preference page.



Name	Section	Option	Value
gvp.general (1 Item)			
gvp.general/service-type	gvp.general	*service-type	voicexml
gvp.policy (4 Items)			
gvp.policy/cti-allowed	gvp.policy	cti-allowed	false
gvp.policy/outbound-call-al...	gvp.policy	outbound-call-allowed	true
gvp.policy/transfer-allowed	gvp.policy	transfer-allowed	true
gvp.policy/use-same-gate...	gvp.policy	use-same-gateway	always
gvp.service-parameters (2 Items)			
gvp.service-parameters/v...	gvp.service-param...	voicexml.context_services_url	fixed,http://bsgenSuite801RG:8084
gvp.service-parameters/v...	gvp.service-param...	voicexml.gvp.appmodule	fixed,VXML_NG
gvp.service-prerequisite (1 Item)			
gvp.service-prerequisite/in...	gvp.service-prereq...	initial-page-url	http://localhost:9001/TestProject/src-gen/...

Running a Callflow

This section describes the configuration required to run a callflow from a Play Application workflow block. Configure the `context_services_url` parameter in Resource Manager's default IVR Profile, which Resource Manager passes on to the VXML application.

1. In the Sip Switch/DN/VOIP Services/MSML_Service DN (if the msml-support option is true in Sip Server) or in the standard VoipService DN (if the msml-support option is false in Sip Server): change the option contact from `sip:host_MCP:port_MCP` to `sip:host _RM:port_RM`.
2. In the Tenant object, designate a default profile for Resource Manager: `gvp.general` section, option `default-application=`<*name of some IVRProfile object under that tenant*>; for instance, Default Application.
3. In the IVR Profile/Default Application specified above, in the Annex, add the section `gvp.service-parameters`.
4. In the `gvp.service-parameters` section, add the option `msml.context_services_url= fixed,http://demosrv8:9080` (here, host:port of Context Management Server, which is the Server port that you specified in Context Services Preferences).
5. In the `gvp.service-parameters` section, add the option `voicexml.context_services_url=`

`fixed,http://demosrv8:9080` (here, host:port of Context Management Server, which is the Server port that you specified in Context Services Preferences).

Context Services Exception Events

Below are some common exceptions for Context Services blocks:

Exception	Number	App Last Error Description
error.com.genesyslab.composer.badfetch	400	Bad Request
error.com.genesyslab.composer.notauthorized	401	Not Authorized
error.com.genesyslab.composer.forbidden	403	Forbidden + specific error message from the server
error.com.genesyslab.composer.notfound	404	Not Found
error.com.genesyslab.composer.servererror	500	Internal Server Error + specific error message from the server
error.com.genesyslab.composer.badgateway	502	Bad Gateway
error.com.genesyslab.composer.serviceunavailable	503	Service Unavailable

Outbound Common Blocks

The Outbound blocks support Genesys Outbound Contact, an automated product for creating, modifying, running, and reporting on outbound campaigns for proactive customer contact. Outbound Contact Solution (OCS) provides automated dialing and call-progress detection, so that an Agent is required only when a customer is connected. Composer supplies the following Outbound blocks:

Block Name	Block Description
Add Record	Automates building of Calling Lists by adding a new record to a specified Calling List.
Cancel Record	Cancels a customer record in a calling list.
Do Not Call	Adds a contact record, such as a phone number or an e-mail address, to a specified Do Not Call List and marks the corresponding record as Do Not Call.
Record Processed	Marks a record as requiring no further handling.
Reschedule Record	Reschedules a customer interaction from the specified Calling List.
Update Record	Updates a Calling List record that you specify via a RecordHandle parameter.

OCS Variables

The Outbound blocks use OCS variables for SCXML applications and OCS variables for VXML applications, present in the Entry blocks of their respective diagrams. These variables are prefixed by "OCS_" and are added to the Entry by default.

Using the Outbound Blocks

Outbound blocks are specifically designed to be used in callflows/workflows that are configured to work with Outbound records, the essential element of which is communication between Universal Routing Server (URS) and Interaction Server, and between Interaction Server and OCS. For additional information, see:

- *Outbound Contact 8.1 Deployment Guide*
- *Outbound Contact 8.1 Reference Manual*

Add Record Block

Use this block to automate building of Calling Lists by adding a new record to a specified Calling List. For example, you can use the Add Record block to automatically develop a Calling List, such as one to follow up on inbound calls that were abandoned during traffic peaks. You can then configure a routing workflow to detect abandoned calls and add records to the Calling List with the parameters of the incoming interactions. The Calling List can then be used by an outbound campaign that dials out to these customers during off-peak hours and has the Agent apologize and follow up. Also see:

- OCS Application Variables
- Using the Outbound Blocks

This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Contact Info Property

Click the down arrow and select the variable that contains the contact telephone number (home, work, cell), FAX number, or e-mail address.

Contact Info Type Property

Click the down arrow and select a Contact Information Type: **No Contact Type**, **Home Phone**, **Direct Business Phone**, **Business with Extension**, **Mobile**, **Vacation Phone**, **Pager**, **Modem**, **Voice Mail**, **Pin Pager**, **Email Address**, **Instant Messaging**.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Record Status Property

Click the down arrow and select a record status, such as **Ready**, **Retrieved**, **Updated**, **Stale**, **Cancelled**, **Agent Error**, **Missed Callback**.

Record Type Property

Click down arrow and select the Type of record, such as **No Record Type**, **Unknown**, **General**, **Campaign Rescheduled**, **Personal Rescheduled**, **Personal Callback**, **Campaign CallBack**, **No Call**.

Call Time Property

This property specifies the time when record was called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the call date and time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the call timetime.
4. Click **OK** to close the dialog box.

Call Time From Property

This property specifies the time frame when a record can be called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time From dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the time.
4. Click **OK** to close the dialog box.

Call Time Until Property

This property specifies the time frame when a record can be called.

1. Click under **Value** to display the button.
2. Click the button to open the Call Time Until dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the time.

4. Click **OK** to close the dialog box.

Scheduled Date and Time Property

This property specifies the date/time at which scheduled call should be dialed.

1. Click under **Value** to display the … button.
2. Click the … button to open the Scheduled Data and Time dialog box.
3. From the **Type** dropdown, do one of the following:
    - Select **Literal** from the dropdown menu and then specify the date and time.
    - Select **Variable** from the dropdown menu and then select the variable that contains the date and time.
4. Click **OK** to close the dialog box.

Time Zone Property

This property specifies the name of a Time Zone, associated with the customer record and configured in Configuration Server.

1. Click under **Value** to display the … button.
2. Click the … button to open the Time Zone dialog box.
3. From the **Type** dropdown, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Time Zone from the **Value** field.
    - Select **Literal** from the dropdown menu and then enter the Time Zone in the **Value** field.
    - Select **Variable** from the dropdown menu and then select the variable that contains the Time Zone from the **Value** field.
4. Click **OK** to close the dialog box.

Attempts Property

Click the down arrow and select the variable that specifies the maximum number of attempts to dial the record in the Calling List during one Campaign.

Calling List Property

This property specifies the name of a Calling List, which is configured in Configuration Server.

1. Click under **Value** to display the button.
2. Click the button to open the Calling List dialog box.
3. From the **Type** dropdown, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Calling List from the **Value** field.
    - Select **Literal** from the dropdown menu and then enter the Calling List in the **Value** field.
    - Select **Variable** from the dropdown menu and then select the variable that contains the Calling List from the **Value** field.
4. Click **OK** to close the dialog box.

Call Result Property

Click the down arrow and result code as defined in a Configuration Manager Enumeration table, such as Abandoned, Agent Callback Error, All Agents Busy, Answer, Answering Machine Detected, Bridged, Busy, Call Drop Error, and so on.

Campaign Property

This property specifies the name of an Outbound Campaign associated with the Calling List, which is configured in Configuration Server.

1. Click under **Value** to display the button.
2. Click the button to open the Campaign dialog box.
3. From the **Type** dropdown, do one of the following:
    - If you are connected to Configuration Server, select **Configuration Server** from the dropdown menu. Select the Campaign from the **Value** field.
    - Select **Literal** from the dropdown menu and then enter the Campaign in the **Value** field.
    - Select **Variable** from the dropdown menu and then select the variable that contains the Campaign from the **Value** field.
4. Click **OK** to close the dialog box.

Chain ID Property

Click the down arrow and select the variable that contains a unique chain identifier (optional). If missing, it is assumed that a record forms a new chain.

Chain N Property

Click the down arrow and select the variable that contains a unique number in a chain (optional). If missing, the next available number is assigned.

OC Server Property

This property identifies the Outbound Contact Server that will interact with the block. You can specify a different OCS application for a specific block. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key-value pairs for user data attached to the interaction.

1. Click under **Value** to display the button.
2. Click the button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
    - Select the **Value is numeric box if applicable.**
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Cancel Record Block

Use this block to cancel a customer record in a calling list. You can identify the customer record to cancel by using the Record Handle, Contact Info, or Customer ID property (one of these must be specified). If you specify more than one of these properties, the identifiers are prioritized as follow: Record Handle (highest), Contact Info, Customer ID (lowest). This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Contact Info Property

Select the variable that contains contact information, such as telephone number (home, work, cell), FAX number, or e-mail address. This parameter can be used for Inbound calls to reference the customer record when Record Handle is not available.

Customer ID Property

Select the variable that identifies the customer when a user-defined field is present in the calling list as described in the *Outbound Contact 8.1 Deployment Guide*. You can use for Inbound calls to reference the customer record when Record Handle is not available.

OC Server Property

This property identifies the Outbound Contact Server processing this Calling List. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

Record Handle Property

Select the variable that identifies the customer using the Record ID assigned by Outbound Contact Server if available. Either Record Handle, Contact Info or Customer ID must be specified.

Tenant Property

Select the variable that identifies the tenant associated with the Calling List.

Update Record Chain Property

Select False to indicate if only the customer record should be cancelled. Select True if all records chained to the customer record should be canceled.

Do Not Call Block

Use this block to add a contact record, such as a phone number or an e-mail address, to a specified Do Not Call List and marks the corresponding record as Do Not Call. **Note:** Do not use the Do Not Call and Record Processed blocks to finalize Outbound record processing. You cannot use other Outbound blocks to process records with the same Record Handle after using Processed or Do Not Call in a workflow. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Contact Info Property

Select the variable that contains contact information, such as telephone number (home, work, cell), FAX number, or e-mail address. This parameter can be used for Inbound calls to reference the customer record when Record Handle is not available.

Customer ID Property

Select the variable that identifies the customer when a user-defined field is present in the Calling List as described in the *Outbound Contact 8.1 Deployment Guide*. You can use for Inbound calls to reference the customer record when Record Handle is not available.

OC Server Property

This property identifies the Outbound Contact Server processing this Calling List. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

Record Handle Property

Select the variable that identifies the customer using the Record Handle if available. Either Record Handle, Contact Info or Customer ID must be specified.

Tenant Property

Select the variable that identifies the tenant associated with the Calling List.

Update Record Chain Property

Select False to indicate if only the customer record should be cancelled. Select True if all records chained to the customer record should be canceled.

Record Processed Block

Use the Record Processed block to mark a record as requiring no further handling. When an Agent finishes processing a Calling List record, Genesys Desktop sends a RecordProcessed event to indicate that the record is processed and Outbound Contact Server updates the record accordingly. Use the Record Processed block in a workflow to have URS request (through Interaction Server) that Outbound Contact Server finish processing a record created as a result of a customer inquiry. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. **Note:** Do not use the Do Not Call and Record Processed blocks to finalize Outbound record processing. You cannot use other Outbound blocks to process records with the same Record Handle after using Processed or Do Not Call in workflow. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server that will interact with the block. You can specify a different OCS application for a specific block. By default, the OCS_URI application variable is used. If the datasource is Config Server, Composer will read the OCS host, listening port and connection protocol from config server. If the datasource is Literal/Variable, the format should be `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key-value pairs for user data attached to the interaction.

1. Click under **Value** to display the … button.
2. Click the … button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
    - Select the **Value is numeric box if applicable.**
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Reschedule Record Block

Use this block to Reschedule a customer interaction from the specified Calling List. A record is typically rescheduled during a call when a customer requests a callback at a certain time. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server (OCS) application that the block will interact with. It allows you to specify a different OCS application for a specific block. By default, the OCS_Record_URI application variable is used.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.
    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Outbound Contact Server object from the **Value** field.
    - You can also select **Literal** and enter the name of the server in the **Value** field.
    - You can also select **Variable** and select the variable containing the name from the **Value** field.

If the datasource is Configuration Server, Composer reads the OCS host, listening port, and connection protocol from Configuration Server. If the datasource is Literal/Variable, use the format `[http|https]://<host>:<port>`.

Scheduled Date and Time Property

Specify the date/time at which scheduled call should be dialed.

1. Click under **Value** to display the button.
2. Click the button to open the Scheduled Date and Time dialog box.
3. The next step depends on whether you are connected to Configuration Server.
4. Do one of the following.
    - Select **Literal** and select the date and time from the **Value** field.

- Select **Variable** and select the name of the variable containing the date and time.
- Select **Delay** and select an amount of time to delay from the **Value** field.

Update Record Block

Use this block to update a Calling List record that you specify via a RecordHandle parameter. For example, in Predictive dialing mode, this request can be used to overwrite the call result detected by call progress detection when needed. Or you can overwrite an answer call result with the wrong party call result. **Note:** When this block is executed in a workflow, it results in an External Service Request (through Interaction Server) to Outbound Contact Server. Since the request goes through Interaction Server, you must have the Genesys Multimedia product installed and an Open Media component to handle External Service processing. For additional information on using this block, including returned results and fault codes, consult the *Universal Routing 8.1 Reference Manual* and the section on updating call results and custom fields in the *Outbound Contact 8.1 Reference Manual*. This block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OC Server Property

This property identifies the Outbound Contact Server (OCS) application that the block will interact with. It allows you to specify a different OCS application for a specific block. By default, the OCS_Record_URI application variable is used.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. The next step depends on whether you are connected to Configuration Server.

    - If you are connected, select **Configuration Server** from the **Type** dropdown menu. Select the name of the Outbound Contact Server object from the **Value** field.
    - You can also select **Literal** and enter the name of the server in the **Value** field.
    - You can also select **Variable** and select the variable containing the name from the **Value** field.

If the datasource is Configuration Server, Composer reads the OCS host, listening port, and connection protocol from Configuration Server. If the datasource is Literal/Variable, use the format `[http|https]://<host>:<port>`.

User Data Property

Use this property to specify key value pairs for user data attached to the interaction.

1. Click under **Value** to display the button.
2. Click the button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.
5. Opposite **Value**, click the down arrow and select either literal or variable.

- If you select **Literal**, enter the name of the key in the second field.
- If you select **Variable**, select the name of the variable from the second field.
- Select the **Value is numeric box if applicable.**

6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Server-Side Common Blocks

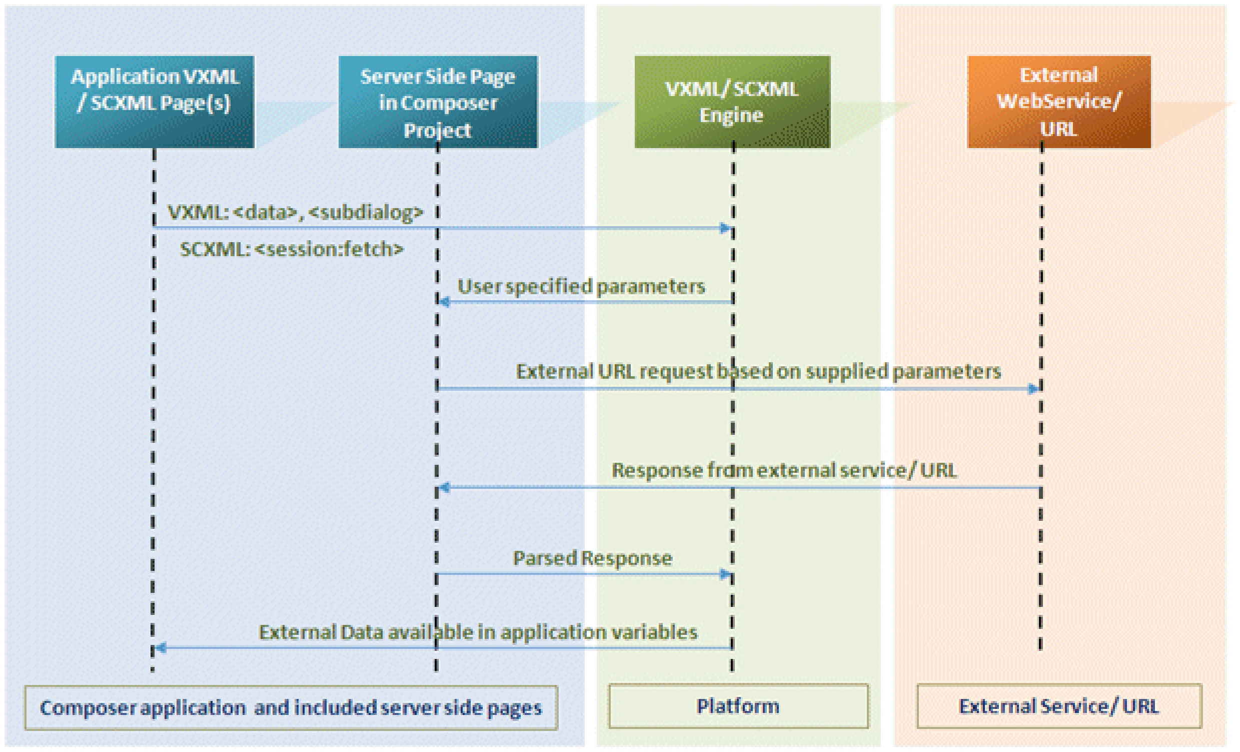
Both routing and voice applications use the Server-Side blocks.

- **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.
- **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)
- **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.
- **TLib Block** (route only). Use this block in workflows and sub-workflows that will use <session:fetch> method="tlib". The block exposes properties to form a TLib request to set agent status not ready equivalent to TAgentSetNotReady. It also sets srcexpr and <content> element to make it possible to form generic TLib requests.

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.

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	
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 the Web Request blocks. What can I use?	Backend block	Try starting with the Backend block since the implementation is 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.

Backend Common Block

The Backend block is used for both routing and voice applications. Use to invoke custom backend Java Server Pages (JSP). You have the option to pass back all the application session state data to the backend logic page on the server. Data being returned will be sent back as a JSON string. Other features:

- Provides a mechanism for creating new backend logic JSP. The added JSP file will have a basic template code already filled out. As the application developer, you will only need to implement a performLogic function. The VXML/SCXML to return back control will be auto-generated in the template.
- User-written custom backend logic pages are stored in the Java Composer Project's `src` folder. Composer provides standard include files for Backend logic blocks in the Java Composer Project's `include` folder.

Note: If any custom backend logic pages use libraries, place the libraries in the Java Composer Project's `WEB-INF/lib` directory. This directory typically contains JAR files that contain Java class files (and associated resources) required for the application. Note: The Tomcat application server should be restarted after changing any JAR files in this folder.

- Composer includes a CHEAT SHEET for creating a Backend logic application as well.

The Backend block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for voice blocks or Common Properties for Workflow Blocks.

Uri Property

The Uri property specifies the http:// page to invoke. To set a URL destination for the Uri property:

1. Select the Uri row in the block's property table.
2. In the Value field, click the … button to open the Uri dialog box.
3. Select a file from the available projects.

Encoding Type Property

The Encoding Type property (used for callflows only) indicates the media encoding type of the submitted document. GVP 8.1 supports two encoding types:

- application/x-www-form-urlencoded
- multipart/form-data

To select a value for the Encoding Type property:

1. Select the **Encoding Type** row in the block's property table.
2. In the **Value** field, select **application/x-www-form-urlencoded** or **multipart/form-data** from the drop-down list.

Parameters Property

Note: Parameters cannot be entered until the Uri property is specified. Use the Parameters property to specify parameters to pass to the invoked backend JSP. To specify parameters:

1. Click the Parameters row in the block's property table.
2. Click the … button to open the Parameter Settings dialog box.

Add Button Use the Add button to enter parameter details.

1. Click **Add** to add an entry to Backend Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the Backend application.
Out	Output parameters are variables that the Backend application returns and will be reassigned back to the current callflow.

InOut	InOut parameters are parameters that act as both input and output.

4. In the **Expression** drop-down list, select from among the listed variables, type your own expression, or click the button to use Skill Expression Builder.
5. In the **Description** field, type a description for this parameter.
6. Click **Add** again to enter another parameter, or click OK to finish.

Delete Button To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

Pass State Property

Note: This property is used for callflows only. The Pass State property Indicates whether or not to pass the application state to the backend. The application state includes all the variables shown in the Entry block as well as all variables containing returned values from user Input blocks. You can find Instructions on how to access these backend variables in Creating a Backend JSP File and Creating a Backend ASP.NET File. The Parameters property can also be used to pass specific parameters into the backend and, for efficiency reasons, should be considered first. There is also a Cheat Sheet, Creating a Backend Logic Block (**Help** > **Cheat Sheets** > **Composer** > **Building Voice Applications**). To select a value for the Pass State property:

1. Select the **Pass State** row in the block's property table.
2. In the **Value** field, select **true** or **false** from the drop-down list.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for

Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Business Rule Common Block

Business Rules

Composer interfaces with the Genesys Rules Engine, which is part of the Genesys Rules System. A business rule is an external piece of business logic, which can be customized, and then invoked by Genesys applications. Here is an example business rule for a bank: `IF product = 'mortgage' and loanAmount >=200000 THEN TTSMsg = 'You must have a credit score of 300 or great to qualify for this loan.'` To simplify rule creation, the Genesys Rules System uses Rule Templates. These are initially created by developers and IT professionals. A Composer-compatible plug-in is available for developing business Rule Templates. This plug-in is provided as part of the Genesys Rules System. For information on installing the plugin, refer to the *Genesys Rule System 8.1 Deployment Guide*. See Chapter 2, Installation. Once validated and deployed, Rule Templates are available for customization in the Genesys Rules Authoring Tool GUI. Business analysts then use the templates to create related sets of business rules called Rule Packages. Packaging rules together allows the business analyst to define which rules will support a particular application. You can use Composer's Business Rule block to request the Genesys Rules Engine to execute a Rule Package in a routing workflow or voice callflow and write the results back to a variable. A business rule preference specifies the Genesys Rules Authoring Tool server to work with. Find information on using the business rules GUI in the following documents:

- *Genesys Rules System 8.1 Deployment Guide*
- *Genesys Rules System 8.1 Rules Authoring Tool Help*
- *Genesys Rules System 8.1 Rules Development Tool Help*

Note: In the Genesys 8.1 release, the Genesys Rules System is packaged only with the intelligent Workload Distribution product and the Conversation Manager product.

Business_Rules_Preferences

The preferences entered here are used in the Business Rule block, Business Rule Package property. To set Business Rules Preferences:

1. Select **Window** > **Preferences** > **Composer** > **Business Rules**.
2. Configure the connection to the Genesys Rules Authoring Tool (GRAT) server by entering the following fields:
    - **GRAT Server**. Enter the address of the Application server hosting the GRAT Server. When using the Business Rule Package property in the Business Rule block, Composer will connect to this server to query information about packages and rules. Example only: http://ca-to-lennon:8080.
    - **Server Path**. Enter the name of the web application deployed as the GRAT. For example, if you have the GRAT running at http://ca-to-lennon:8080/genesys-rules-authoring, then the

GRAT server is http://ca-to-lennon:8080 and the Server Path is /genesys-rules-authoring.

- **Tenant**. To obtain a list of Rule Packages, Composer will query the GRAT server using an HTTP request to http://{server-address:port}/tenant/packages. Enter the name of the tenant as defined in the Configuration Database.
- **Username**. Enter the username defined in the Configuration Database for logging into the GRAT server.
- **Password**. Enter the password defined in the Configuration Database for logging into the GRAT server.
- **Genesys Rules Engine (Optional). GRE URL**. Enter the URL for the GVP Debugger to use when starting a call. The GRE URL will be passed to the VXML application in the SIP URL. If set, this value will be passed to the voice or routing application and will override the value set in the Rules Engine URL property of the Business Rule Block (see that section below).

Business Rule Templates

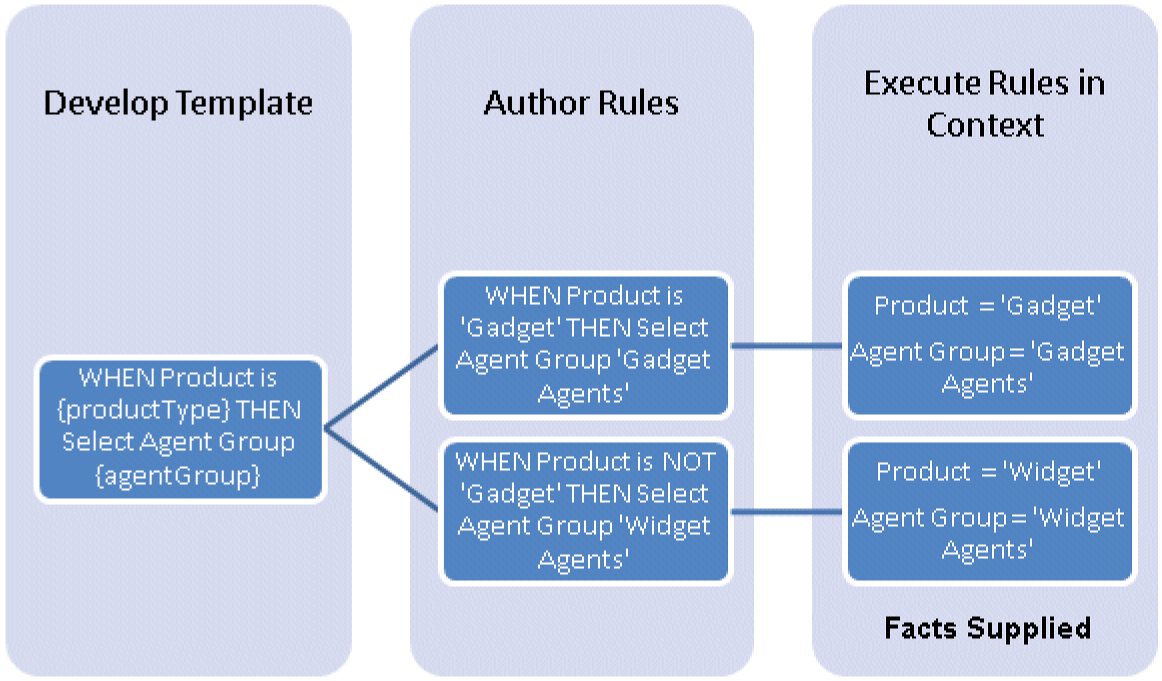
This functionality is enabled by an Eclipse plug-in that can be installed within Composer or in a standalone Eclipse environment.

- To install the plugin, refer to the *Genesys Rule System 8.1 Deployment Guide*. See Chapter 2, Installation.

The plug-in enables developers to create Rule Templates. Rule Templates consist of rule parameters, conditions, actions, and functions. When a Rule Template is published to the Rules System repository, it is made available to be added to Rule Packages. Rule Packages are the deployable objects, which are used to expose rule conditions and actions to business users for creating rules through the Genesys Rules Authoring tool. A brief summary of Rule Templates is presented below. For detailed information, see the *Genesys Rules System 8.1 Rules Development Tool Help*. Once you install the plugin, this help system is available within Composer by selecting **Help** > **Contents**.

Genesys Rules System Architecture

A logical view of the Genesys Rules System architecture is shown below.


Develop Template
Author Rules
Execute Rules in Context
WHEN Product is {productType} THEN Select Agent Group {agentGroup}
WHEN Product is 'Gadget' THEN Select Agent Group 'Gadget Agents'
WHEN Product is NOT 'Gadget' THEN Select Agent Group 'Widget Agents'
Product = 'Gadget'
Agent Group = 'Gadget Agents'
Product = 'Widget'
Agent Group = 'Widget Agents'
Facts Supplied

- The first category reflects Rule Template creation, which can be done in Composer if the set of plugins is installed.
- The second category reflects rule creation by business analysts in the Genesys Rules Authoring Tool.
- The third category reflects rule evaluation by the Genesys Rules Engine using the Business Rule block once the Facts are known.

Type of Rules

The Genesys Rules System supports both basic and decision table rules.

Basic Rule

A basic or linear business rule is of this form: WHEN {condition} THEN {action} In other words, when the condition is true, the action will occur. This is a rule template. When a business analyst uses the Genesys Rules Authoring Tool to customized a template with valid values, this creates a business rule. The following rules are all valid instances:

- `WHEN Product = 'Gadget' THEN Select Agent Group 'Gadget Agents'`
- `WHEN Product = 'Widget' THEN Select Agent Group 'Widget Agents'`
- `WHEN Customer Segment = 'Gold' THEN Assign Credit Limit '200000'`

This form of rule is preferred for simple actions, such as assigning a value to return back to the application.

Decision Table Rule

A business rule can also take form of a decision table. For example, assume in a particular scenario that there are 3 customer levels: Gold, Silver, and Bronze. For each of these levels, we wish to make an offer to customers based on a qualifying purchase they may have made. Gold customers automatically qualify for a Premium Offer. Silver customers need to have spent $1000 or more to qualify for that offer, otherwise they get the Special Offer. Finally, Bronze customers need to have spent $5000 or more for the Premium Offer; or $2000 or more for the Special Offer; otherwise they are informed of the offers available if they make the qualifying purchase level. **Note:** The Genesys Rules Engine cannot execute Rule Templates.

Business Rule Block

Once the Rule Package (created from Rule Templates) that you want to work with are deployed to the Genesys Rules Engine, you can use the Business Rule block on the Server Side palette to create voice and routing applications that use business rules. Use this block to have Composer query the Genesys Rules Authoring Tool (GRAT) for deployed packages. 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.   **Note:** This last step (evaluation) happens as part of a VXML or SCXML application that Composer developer creates, not as part of Composer. A business rule preference specifies the Genesys Rules Engine to work with. **Runtime Parameters** The following parameters (defined in Preferences) are used at runtime, when the VXML and SCXML application queries the GRAT to execute the rule.

- `grat_username` -- a user login for accessing the GRAT server
- `grat_password` -- the password for the above login
- `grat_server` -- a URL for the GRAT server, for example: http://hostname:8080/genesys-rules-authoring
- `grat_tenant` -- the tenant associated with the login, e.g. Environment

The Business Rule block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Business Rule Package Property

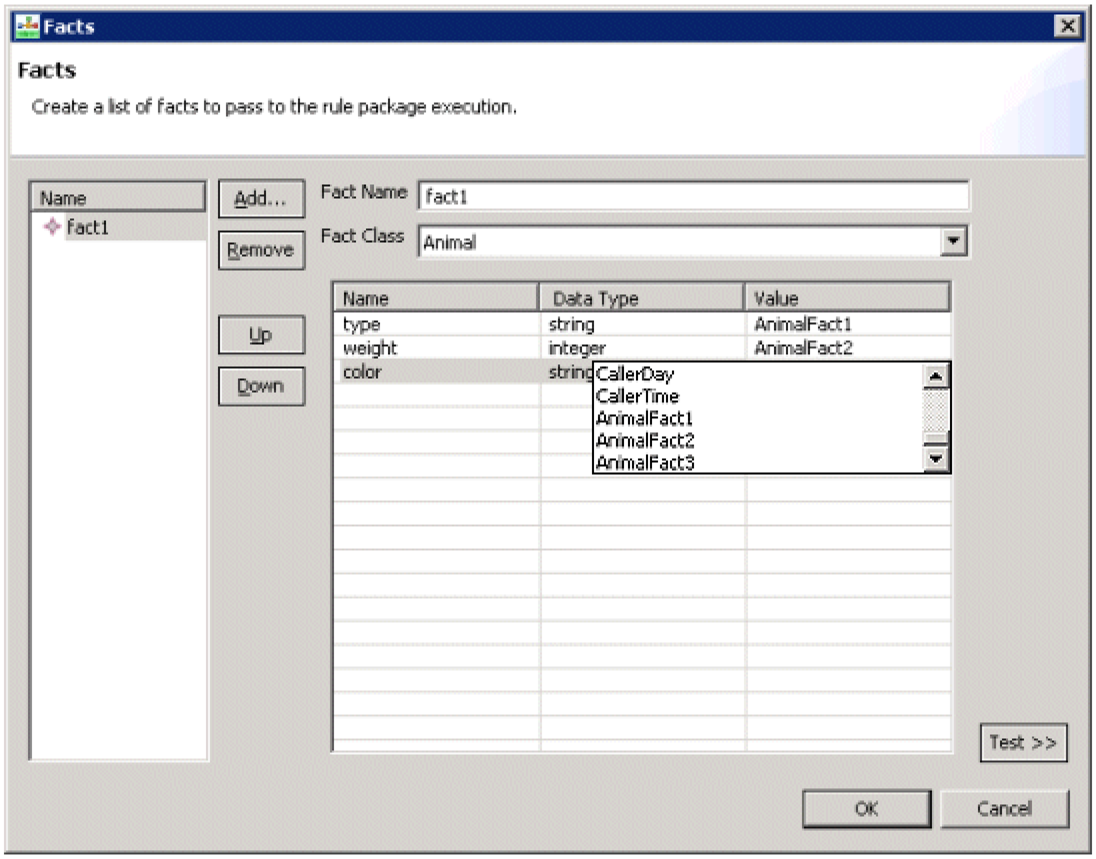
Use to select the Rule Package (collection of related rules) you would like to execute. Packaging rules together allows the business analyst to define which rules will support a particular application. Before using this property, you must set Business Rules Preferences.

1. Click the button to request Composer to connect to the Genesys Rules Authoring Tool Server using the information specified in Business Rule Preferences. After a successful connection, the Business Rule Package dialog appears.
2. Select a Rule Package and click **OK**. The dialog closes and the name of the Rule Package appears under Value.

Facts Property

Use this property to execute the logic contained in the selected Rule Package by supplying input parameters called *Facts*. To specify Facts:

1. Click the **Facts** row in the block's property table.
2. Click the button to open the Facts dialog box.
3. Click **Add**. The dialog box adds additional fields consisting of the Facts to use when executing the Rule Package. You then click the down arrow and select a value or a variable that contains the value for each Fact. An example dialog box is shown below.


Facts

Create a list of facts to pass to the rule package execution.

Name: fact1

Fact Name: fact1

Fact Class: Animal

Name	Data Type	Value
type	string	AnimalFact1
weight	integer	AnimalFact2
color	string	

Facts.gif

4. Enter the **Fact Name** field.
5. Click the down arrow and select an entry for the **Fact Class** field.
6. Click the down arrow and select a value or a variable that contains the Fact value.
7. Click **Add** again to enter another Fact, or click **OK** to finish.

Delete Button To delete a Fact:

1. Select an entry from the list.
2. Click **Delete**.

Rules Engine URL

Select the variable containing the Genesys Rules Engine URL. Background: Starting with 8.1.2, Composer-generated applications no longer interact with the GRAT server at runtime. Previous requests to the GRAT Server were done to retrieve the URL of the GRE server to which a rules package is deployed. Instead, the runtime applications now use the Rules Engine URL property, which is passed into the application via the IVR Profile or an Enhanced Routing Script object. You can use this Rules Engine URL property to override any GRE URL configured in the IVR Profile or

EnhancedRouting Script object.

Exceptions Property

The Business Rule block supports the following exceptions. They correspond to the HTTP status codes returned by the Business Rule Server (BRS).

Exception Event Name	HTTP Return Code	BRS Error Code	Description
error.com.genesyslab.composer.badrequest	400	610	The received URI does not match the Engines REST specification.
error.com.genesyslab.composer.notfound	404	620	The package for the evaluation request received was not found.
error.badfetch.http			Any other HTTP error.
error.com.genesyslab.composer.notacceptable	406	602	The evaluation request received could not be converted to a valid knowledgebase-request message, or the evaluation request received could not be evaluated due to an exception.

Details of the exception can be obtained from the body of the response. The Composer application will log the description. The JSON body of the response will look like the following: `{ error:{ code:6xx, description:error message } }` Also see Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Interaction ID Property

Set to a meaningful value or keep the default value, which is the system variable InteractionId. Applicable to workflows only. Can be used for "interaction-less" processing for scenarios where the InteractionId variable is not automatically initialized, but instead must wait for an event. An example would be an SCXML application triggered by a Web Service that does not add an interaction. Background: Previous to 8.1.1, Composer did not expose an Interaction ID property. Instead, when ORS started processing an interaction, a generated SCXML application automatically initialized the system variable, InteractionId. This variable was then used internally by Routing and certain eServices blocks when interacting with ORS. With the introduction of support for Interaction-less processing, you can now define a specific event (IPD Wait For Event property) to initialize InteractionId, or not define an event at all. For scenarios with an interaction (IPD Diagram/Wait For Event=interaction.present for example), you may keep the default value for the Interaction ID property. The default value is the system variable InteractionId, which is initialized automatically in this case. For other scenarios (any scenario where the system variable InteractionId is not set), you may choose to:

1. Not use blocks that require an Interaction ID
2. And/or set the Interaction ID property to a meaningful value
3. And/or assign a meaningful value to the InteractionId system variable

Output Result Property

Use this property to save the results of the business rule execution to a variable. To select a variable:

1. Select the **Output Result** row in the block's property table.
2. In the Value field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

The format of returned data is JSON. Any post-processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It supports writing simple or complex expressions to extract values out of JSON strings and arrays. In a workflow, the Output Result can be attached to User Data. In the Specify Output Result Location dialog box, select

User Data or Variable. If User Data is selected, the specified name is used as a prefix of the keys that will be added to user data. For example, if you specify abc, then the User Data will look like:

```
  'abc_fact1'(list) '@class'      'com.genesyslab.animals.Animal'
                                  'color'       'red'
                                  'type'        1903
                                  'weight'      123                  'abc_fact2'(list)
'@class'     com.genesyslab.animals.Car'                                    'make'
       'mazda'
```

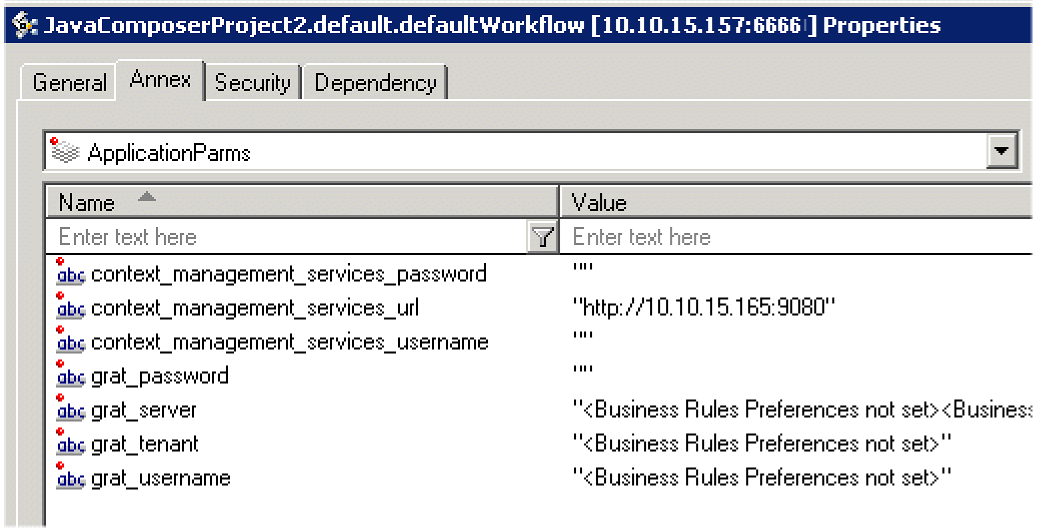
Note: The Output Result property takes effect only during application runtime. Its purpose is to take the output of the rule execution (at runtime) and store returned results back in the specified application variable so other parts of the application can access the data.

Business Rules Block Runtime Configuration

The table below shows the parameters that must be set up in Genesys Administrator in order for the Business Rules block to work.

	ERS Object Key Names	**IVRprofile Object Key Names**
GRS	grat_server	grat_server
	grat_tenant	grat_tenant
	grat_username	grat_username
	grat_password	grat_password

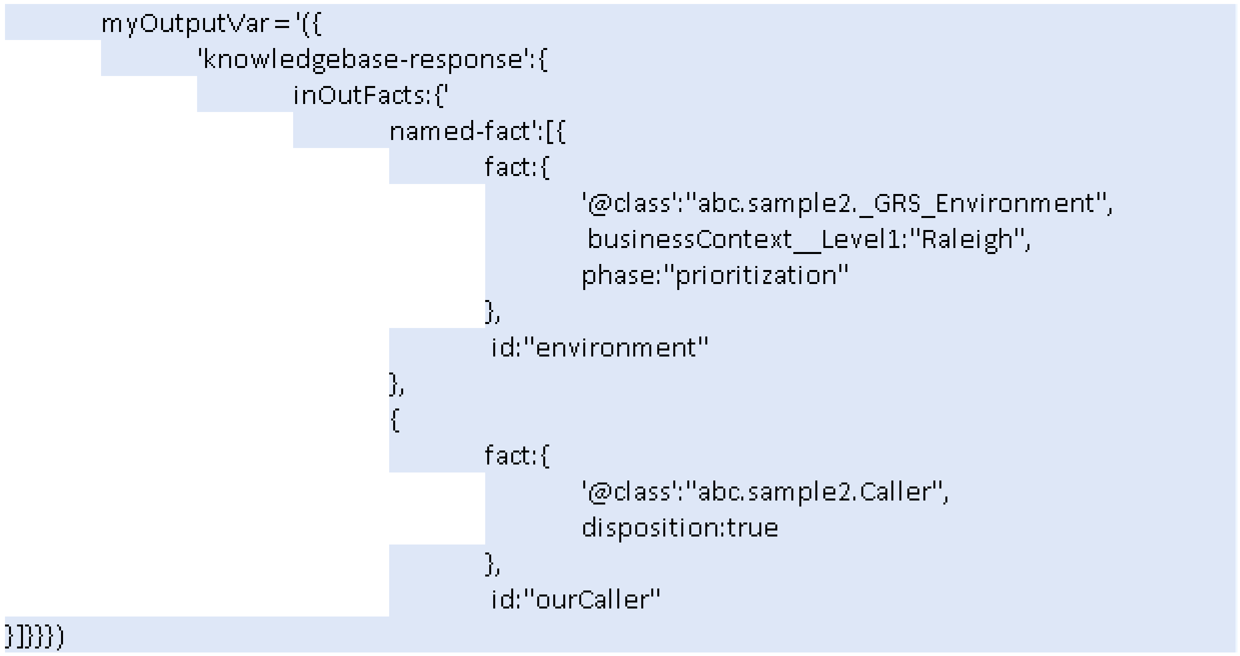
The figure below shows an example Enhanced Routing Script object created by Composer. It creates these parameters in the ApplicationParms section in the Annex, so you do not have to key in parameter names. Note: If you accidentally changes parameter names, these functions will not work.



Name	Value
context_management_services_password	""
context_management_services_url	"http://10.10.15.165:9080"
context_management_services_username	""
grat_password	""
grat_server	"<Business Rules Preferences not set><Business
grat_tenant	"<Business Rules Preferences not set>"
grat_username	"<Business Rules Preferences not set>"

Working With Returned Data

Below is an example on how to work with data returned by the Business Rules block. A sample of the output can look like the snippet below, which will be stored in the output variable myOutputVar.



```
        myOutputVar='({
                'knowledgebase-response':{
                        inOutFacts:{'
                                named-fact':[{
                                        fact:{
                                                '@class':"abc.sample2._GRS_Environment",
                                                 businessContext__Level1:"Raleigh",
                                                phase:"prioritization"
                                        },
                                         id:"environment"
                                },
                                {
                                        fact:{
                                                '@class':"abc.sample2.Caller",
                                                disposition:true
                                        },
                                         id:"ourCaller"
}]}}})
```

To extract the value of the disposition field, an expression like this can be used: `myDisposition = myOutputVar["knowledgebase-response"].inOutFacts["named-fact"][1].fact.disposition` This will return true.

DB Data Common Block

Important Note: When using the DB Data block, database errors (such as failure to connect to a database) could result in an invalid JSON message being returned to the workflow or callflow, which could cause an application to fail. In order to avoid this potential issue, Genesys recommends upgrading to Release 8.1.301.02 as it contains an update to the database access library.

The DB Data block is available for both routing and voice applications. 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 Using the Query Builder or Stored Procedure Helper. It can also use a SQL script file. **Note:** When using the DB Data block to connect to and query information from an Oracle database, some connections may remain in the TIME_WAIT state. If you encounter this situation, use connection pooling in order to avoid exhausting the number of allowed Oracle connections. This block acts as a data source for the DB Prompt and DB Input blocks (available only in callflows). An Entry block user variable can also be used to access the results of a Stored Procedure call specified in a DB Data block for both voice and routing applications. **Note:** The Looping block can work with the DB Data block. For example, you can use the Looping block to Iterate over a data set returned by the DB Data block to map values returned from a database query to application variables. Also see: Working with Database Blocks. The DB Data block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. **Note:** if you rename a DB Data block, its corresponding SQL statement file in the db folder will not be updated and will not be valid until you generate code again.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Connection Profile Property

The Connection Profile property allows you to select a previously-created database Database Connection Profiles that specifies database details for this DB Data block. If you have not created a connection profile, open the Connections Profile editor as follows:

1. Under **Value**, click the down arrow.
2. Select **Create New Profile Using Editor...**

Refer to the topic Database Connection Profiles for instructions. To select a connection profile for your database query:

1. Select the **Connection Profile** row in the block's property table.
2. Select the connection profile to use for this query.

Connection Properties Property

The Connection Properties property allows you to override the parameters in connection profile during runtime. The properties that can be overriden are Hostname, Password, Port, Database, Username and other Custom Parameters. Variable mapping can be configured in the dialog box provided for the property. To define the variable mapping for Connection Parameters:

1. Click the ⋯ button to open the Connection Properties variable mapping dialog.
2. Dialog displays the parameter name and value in connection profile. Select the system variable in the drop down combo against each property.
3. Click OK

Connection String Property

The Connection String Property allows you to define the value of Connection String that need to be used at Runtime. If this property is specified, the parameters from Connection Profile is ignored. To define this property enter either literal value or select system variable from the combo provided for the property.

Timeout Property

The Timeout property defines the length of time in seconds that the voice application will wait for query execution to complete. To provide a timeout value**:**

1. **Select the** Timeout row in the block's property table.
2. In the Value field, type a timeout value, in seconds.

The default value (20 seconds) of this property is used if not specified explicitly. Disable the timeout by setting to -1. If the query takes longer than this specified time to complete the `error.com.genesyslab.composer.dbtimeout` exception is thrown. In order to select a query type, the Connection Profile property must be set.

Query Type Property

To define a query type:

1. Select the **Operation Type** row in the block's property table.
2. Select one of the following:
    - **SQLQuery**
    - **SQLScriptFile**
    - **StoredProcedure**

Based on the value selected for **Operation Type**, the specified value is used and some properties are not used. Query Property The Query property opens the Query Builder in which you can visually build the database query. **Note:** The Query property and Query File property are mutually exclusive. If both are entered, then the Query File property takes precedence over the query defined in the Query property. To define a query:

1. Select the **Query** row in the block's property table.
2. Click the ▭ button to open the Query Builder.

Query File Property

The Query File property accepts a filename that points to a SQL file that the user has written. To provide a filename for a user-written SQL file:

1. Select the **Query File** row in the block's property table.
2. In the Value field, type the filename of the SQL file (the file is usually in the db folder of your project. If it is present in a different location, specify a relative path, such as `../myfolder/myquery.sql`.

Stored Procedure Property

The Stored Procedure property opens the Stored Procedure Helper in which you can visually build the database query. To define a stored procedure call:

1. Select the **Stored Procedure** row in the block's property table.
2. Click the ▭ button to open the Stored Procedure Helper.

Column Names Variable Property

The Column Names Variable property maps the list of column names in the result to the specified variable. The default is Use system default, in which case the system uses an internal variable which is named in the format below. Genesys recommends that you define a user variable for this purpose in the Entry block and specify it in the DBData block. For Callflow diagrams: AppState.<blockname>DBResultColumnsNames For Workflow diagrams: App_<blockname>['DBResultColumnsNames'] To select a variable:

1. Select the **Column Names Variable** row in the block's property table.
2. In the **Value** field, select the variable from the dropdown list.

Records Variable Property

The Records Variable property maps the records (data) in the result set to the specified variable. The default value is Use system default, in which case the system creates an internal variable which is named in the format below. However, Genesys recommends that you specify a user variable in the Entry block. For Callflow diagrams: AppState.<blockname>DBResult For Workflow diagrams: App_<blockname>['DBResult'] To select a variable:

1. Select the **Records Variable** row in the block's property table.
2. In the **Value** field, select the variable from the dropdown list.

Note: The following applies to all methods of getting database results (query builder, stored procedure helper, custom queries): Results are stored in a variable as a two-dimensional JSON array. This data can then be accessed via a Looping block or via scripting in the Assign or ECMAScript block. For example, if the database result set looks like this in tabular form:

Vegetables	Animals
lettuce	chicken
broccoli	lion

The JSON for the result will look like this: `{"db_result":[["lettuce", "chicken"], ["broccoli", "lion"]],"db_result_columns":["vegetables", "animals"]}`

Suppress Empty Result Set Exception Property

The Suppress Empty Result Set Exception property determines if the dbemptyresultset exception should be thrown if a query or a stored procedure execution results in an empty result set (number of records returned is zero). To provide a value:

1. Select the **Suppress Empty Result Set Exception** row in the block's property table.
2. Select **true** or false.

Exceptions Property

Find this property's details under Common Properties for voice blocks or Common Properties for Workflow Blocks. The Exceptions dialog box for the DB Data block has the following exception events:

- `error.com.genesyslab.composer.dbconnectionerror`
- `error.com.genesyslab.composer.dberror` (pre-selected in the Supported column)

- `error.com.genesyslab.composer.dbemptyresultset` (pre-selected in the Supported column)
- `error.com.genesyslab.composer.dbtimeout`

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

External Service Block

This block 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. Can be used for both voice and non-voice interactions. **Notes:**

- In order to use this object, the third party server/application must already be defined in the Configuration Database as a server of type Third Party Server or Third Party Application. Before completing the External Service block properties, you must already know the names of Services, Methods, and Signatures (requested input/output parameters) provided by the external service.
- The Composer External Service block does not automatically pass user data in the ESP call unlike the legacy IRD External Service object. Therefore, ESP methods that expect user data cannot be called using this block. Please refer to the ESP method/API documentation to determine if user data is required. To call an ESP API that requires user data, a hand coded SCXML page can be used and invoked using the SubRoutine block. Please refer to the <session:fetch> documentation in the Orchestration Server Developers Guide. See Action Elements under Session Interface for details on how to pass user data in ESP requests.

Use Case

A customer has a custom integration to a third party application (a workflow system), through the Open Media API. The workflow system uses Genesys to distribute work items at various times during the workflow. At some point in the IPD handling a work item, there is a need to update the workflow system and assign a new value to one of the attributes of the work item. The Genesys developer has the IPD call a routing strategy, which uses the External Service block to call a specific method exposed by the third party application. This allows the developer to update the value of the specific attribute of the work item. The External Service block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events.

Application Property

Use this property to select the name of the third party application to be contacted or the general application type to be contacted, which must be defined in the Configuration Database.

1. Click under **Value** to display the button.
2. Click the button to open the Application Selection dialog box.
3. Select the third party application to be contacted.
4. Click **OK**.

Method Name Property

Use this property to specify the Method defined by the third party server or application.

1. Click under **Value** to display the button.
2. Click the button to open the Method Name dialog box.
3. Opposite **Type**, select one of the following as the source for the name:
    - **Literal** to enter the method name manually in the Value field.
    - V**ariable** to select a variable for the method name in the Value field.
4. Click **OK** to close the dialog box.

Method Parameters Property

Use this property to specify the list of input parameters to be passed to the specified external service. Click the button to add a new entry:

1. Click under **Value** to display the button.
2. Click the button to open the Method Parameters dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite **Key**, leave **Literal** in the first field and enter the input parameter name in the second field.

5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
6. Click **OK** to close the Select Items dialog box. The Method Parameters dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the Method Parameters dialog box. .

Service Name Property

Use this property to specify the name of the Service defined by the third party server or application for the functionality requested.

1. Click under **Value** to display the button.
2. Click the button to open the Service Name dialog box.
3. Opposite **Type**, select one of the following as the source for the name:
    - **Literal** to enter the service name manually in the Value field.
    - **Variable** to select a variable for the service name in the Value field.
4. Click **OK** to close the dialog box.

Service Timeout Property

Use this property to specify the timeout in seconds (s) to be used for invoking this method. If not checked, URS uses the Reconnect Timeout entered for third party server or application in Configuration Server. In the case of a connection or service request failure, error codes are returned. The default is 30 seconds.

User Data Property

Use this property to specify the list of User Data parameters to be passed to the specified external service. Click the button to add a new entry:

1. Click under **Value** to display the button.
2. Click the button to open the User Data dialog box.
3. Click **Add** to open the Select Items dialog box.
4. Opposite Key, leave Literal in the first field and enter the input parameter name in the second field.

5. Opposite **Value**, click the down arrow and select either literal or variable.
    - If you select **Literal**, enter the name of the key in the second field.
    - If you select **Variable**, select the name of the variable from the second field.
6. Click **OK** to close the Select Items dialog box. The User Data dialog box shows your entry.
7. Continue adding parameters in this fashion.
8. Click **OK** when through in the User Data dialog box.

Result Property

Use this property to specify an application variable to store the results. These results will then be available in other blocks in the application for further processing. The format of returned data is JSON. Any post processing work to be done on returned results can be done in the existing Assign block which provides access to ECMAScript functions. It already supports writing simple or complex expressions to extract values out of JSON strings and arrays.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

OPM Common Block

The OPM block 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). The OPM block is available for both Callflows (VXML) and Workflows (SCXML).

The OPM block allows browsing through a JSON structure based on metadata retrieved from the GAX server in order to form a JSON expression. The OPM block generates code to evaluate the specified expression and assign results to the **App_OPM** (voice application) or **system.OPM** (routing application) application variable accessible via the Entry block.

GAX Server

GAX refers to a Genesys Administrator Extension (GAX) plug-in application used by Genesys web application EZPulse. EZPulse enables at-a-glance views of contact center real-time statistics in the GAX user interface. A button on the Composer main toolbar, Launch GAX Server Command, lets you launch the Genesys Administrator Extension used by the GAX Server. Composer uses the server host, port, username, and password entered on the GAX Server Preferences page to fetch audio resource management parameters or an audio resource IDs list. Before using this block set GAX Server Preferences.

Note: The OPM block in Composer 8.1.2 supports GAX 8.1.2.

Note: GVP 8.1.6 supports OPM parameters only with lowercase key names - Composer includes a warning to that effect. Please consult your GVP version's documentation for any changes to this behavior.

The OPM block has the following properties:

Name Property

Find this property's details under Property Common Properties for Workflow Blocks or Property Common Properties for Callflow Blocks

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Notes Property Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Property Common Properties for Workflow Blocks or Property Common Properties for Callflow Blocks

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Assign OPM Data Property

Use this property to assign OPM parameters from GAX to variables.

1. Click opposite **Assign OPM Data** under **Value.** This brings up the button.
2. Click the button to bring up the Assign OPM Data dialog box.
3. Click **Add**.
4. Select the variable or click the **Variables** button to add a new variable.
5. Enter a value for the variable or click the button where you can create an expression with Expression Builder.

TLib Block

Use this block in workflows and sub-workflows that will use <session:fetch> method="tlib". The block exposes properties to form a TLib request to set agent status not ready equivalent to TAgentSetNotReady. It also sets srcexpr and <content> element to make it possible to form generic TLib requests. The TLib block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks.

Condition Property

Find this property's details under Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Workflow Blocks.

Result Property

Response returned from the TLib function. You may use this property to assign the collected data to a user-defined variable for further processing.

1. Select the Output Result row in the block's property table.
2. In the Value field, click the down arrow and select a variable.

Application Property

The T-Server application object in Configuration Server that represents the third party server to be contacted. You can select Configuration Server, enter a literal or enter a value.

Content Property

Use this property to specify input parameters to be passed to the specified TLib function. You can enter as literals and enter property names and values or select a variable.

URL Property

Select the variable that contains the URI for the TLib function that is to be used.

Enable Status Property

Find this property's details under Common Properties for Workflow Blocks.

Web Request Common Block

The Web Request block is used for both routing and voice applications. Use to invoke any supported HTTP web request or REST-style web Service.

- It supports PUT, DELETE, GET and POST methods over HTTPS.
- It is based on common Web Services standards such as XML, SOAP and WSDL instead of proprietary standards that are currently being replaced.

REpresentational State Transfer (REST) is an XML-based protocol for invoking Web Services over HTTP. REST is a lighter version of SOAP, which has evolved into a more complex protocol. REST-style web services offer a less coupled paradigm whereby simpler requests and responses are used. As an example, a simple HTTP request follows the REST methodology. The Web Request block allows the user to query "RESTful" Web services. The supported return formats for the Web Request block are:

- plain text

Note: For workflows, the result will be returned in a JSON string with key name result, e.g., {"result":"This is a plain text result"}

- plain XML
- JSON string (See an issue pertaining to JSON objects in Troubleshooting.)

> **Important**
> Composer does not support fetching URLs using HTTPS in Web Request and Web Service blocks.

The Web Request block has the following properties:

Name Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks

Block Notes Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Workflow Blocks or Common Properties for Callflow Blocks You can also define custom events.

Request Method Property

This property Indicates the method for invoking the web request:

- **get**--Invoked using HTTP Get
- **post**--Invoked using HTTP Post. This option is valid only when the parameters are passed as a namelist (Use Namelist property is set to true). This is generally used when a large amount of data needs to be sent as an input value for a subdialog.
- **put**--Invoked using HTTP Put
- **delete**--Invoked using HTTP Delete

To select a value for the Request Method property:

1. Select the **Request Method** row in the block's property table.
2. In the **Value** field, select get, post, put, or delete from the drop-down list.

Uri Property

The Uri property specifies the http:// page to invoke. To set a URL destination for the Uri property:

1. Select the **Uri** row in the block's property table.
2. In the **Value** field, click the down arrow and select the variable that contains URL.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Authentication Type Property

The Authentication Type property specifies whether to use an anonymous or basic authentication for the web request. To assign a value to the Authentication Type property:

1. Select the Authentication Type row in the block's property table.
2. In the Value field, select anonymous (default) or basic from the drop-down list. With the anonymous type of access, no user name/password is passed to Web service for client authentication in order to get data. If you select the basic type of access, you must supply the Login Name and Password properties.

Encoding Type Property

The Encoding Type property (used for callflows only) indicates the media encoding type of the submitted document. GVP 8.1 supports two encoding types:

- application/x-www-form-urlencoded
- multipart/form-data

To select a value for the Encoding Type property:

1. Select the **Encoding Type** row in the block's property table.
2. In the **Value** field, select one of the following:
    - **application/x-www-form-urlencoded (default)**
    - **application/json**

Input Parameters Property

Use the Input Parameters property to specify a list of required Name/Value pairs to pass as parameters to the http:// page. To specify input parameters:

1. Click the Parameters row in the block's property table.
2. Click the ⋯ button to open the Parameter Settings dialog box.

Add Button Use the Add button to enter parameter details.

1. Click **Add** to add an entry to Web Request Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the web request.
Out	Output parameters are variables that the web request returns and will be reassigned back to the current callflow/workflow.
InOut	InOut parameters are parameters that act as both input and output.

1. In the Expression drop-down list, select from among the variables shown, type your own expression, or click the ⋯ button to use Skill Expression Builder.
2. In the Definition field, type a description for this parameter.
3. Click Add again to enter another parameter, or click OK to finish.

Delete Button To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

JSON Content Property

If the HTTP request to be invoked expects JSON content, this property can be used to specify that input. It expects a variable whose content will be sent to the API specified in the HTTP URI property of the block. Set the `Encoding Type` property of the block to `application/json`. In this case, the `Input Parameters` property will not be used.

The variable selected in this property should contain a JavaScript object. The object can be built from a JSON string, or using the ECMAScript block.

For example, if you would like to pass a JSON content to the HTTP URI, using a variable named "content", the variable can be initialized in the following ways:

- If you have a JSON string, you can use the Assign block to assign the following value to "content":

```
JSON.parse('{"abc": "def", "xyz": 3}')
```

- Alternately, you can build a JavaScript object using an ECMAScript block with code like the following:

```
var content = new Object(); content['abc'] = 'def'; content['xyz'] = 3;
```

In both cases, set the `JSON Content` property of the Web Request block to the variable named "content".

Timeout Property

Select the variable containing the number of seconds that the application will wait when fetching the result of the Web Service or the Web Request. If the requested resource does not respond in that time, then a timeout event will occur.

Custom HTTP Headers Property

Use this property to add Custom headers to be sent along with the HTTP request during the runtime execution of the Server Side block.

1. Click the row in the block's property table.
2. Click the … button to open the Custom HTTP Headers dialog box.
3. Click **Add** to open Configuration Custom HTTP Headers dialog box.
4. Select a Header type.
5. Select **Literal** or **Variable**.
6. Type the literal value or select the variable that contains the value.

Login Name Property

Used when Authentication type = basic. The Login Name property specifies the login name for the invoked web page. To provide a login name for the web request:

1. Select the **Login Name** row in the block's property table.
2. In the **Value** field, type a valid login name.

Password Property

Used when Authentication type = basic. The Password property specifies the password for the invoked web page. To provide a password for the web request:

1. Select the **Password** row in the block's property table.
2. In the **Value** field, type a valid password that corresponds to the login name above.

Result Property

The Result property is the variable used to get back a result from the web request. To select a variable:

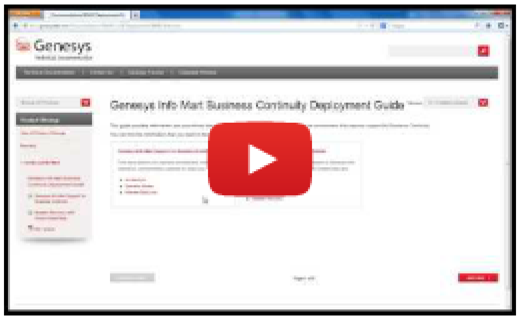
1. Select the **Result** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

Web Service Common Block

Video Tutorial

Below is a video tutorial on using the Web Service block.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```



SOAP-Compliant Web Services

This block can be used to invoke SOAP 1.1 compliant Web Services. It accepts and parses WSDL content for the WebService and collects input parameters based on this WSDL content.

- Uses common Web Services standards such as XML, SOAP and WSDL.
- You can pass parameters (as in subdialogs) and store the return values in variables.
- GET, POST and SOAP over HTTPS are supported.
- Supports SOAP 1.1 and therefore requires a WSDL file to describe endpoints and services. The Web Service block will not work without this WSDL file.
- WSDL-based Web Services are supported with certain limitations. The WSDL is parsed and you are provided the option to select the service name, bindings type, operations, service end point, and mode (GET / POST). The Input and Output parameter list is pulled by default from the WSDL.

Data returned by the Web Service is converted to JSON format and made available in the application. (See an issue pertaining to JSON objects in Troubleshooting.)

Important

SOAP 1.2 is not supported.

Additional Information

For additional information, see:

- Web Service Block and Signed SOAP Requests and Web Service SOAP Message Examples.
- WSDL_SOAP_XSD_WSSE_Support

Web Service Block Security

For Java and .NET Composer projects, the Web Service Block supports secured SOAP communication using XML Digital Signature with a Client Certificate for Java Composer Projects. XML Digital Signature authentication is in compliance with the Second Edition of the XML Signature Syntax and Processing Specification and the OASIS Web Services Security SOAP Messages Security Specification. The Authentication Type property below allows you to select various types of authentication.

Important

Composer does not support fetching URLs using HTTPS in Web Request and Web Service blocks.

Testing the Web Service Block

When working with either a callflow or workflow, the Web Service block provides menu option to test the configured SOAP Web Service using the Web Services Explorer. Right-click the Web Services block and select **Test with Web Services Explorer.** The Web Service block has the following properties:

Name Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Block Notes Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Exceptions Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks. You can also define custom events. The Web Service block Exceptions dialog box has the following pre-set exceptions:

- Callflows: `error.badfetch and error.com.genesyslab.composer.webservice.badFetch`
- Workflows: `error.session.fetch and error.com.genesyslab.composer.webservice.badFetch`

Service URL Property

The Service URL property specifies the WSDL URL of the Web Service to invoke. To set the Service URL:

1. Select the **Service URL** row in the block's property table.
2. In the **Value** field, type a valid URL.

When you provide the WSDL URL in the Service URL property, Composer will try to access the URL and parse it to populate the drop-down lists for the remaining properties:

- **Available Services**
- **Bindings**
- **Operations**
- **Service End Point**
- **Use Protocol**

Note: When ***upgrading older diagrams to 8.1.1 and higher, it is necessary to clear out the service URL***and specify it again. This is needed in newer versions to re-parse the WSDL obtained from the specified URL and not use the cached information stored in the diagram.

Available Services Property

When Composer accesses the Service URL, the available Web Services will populate the drop-down list of the Available Services property. To select an available service:

1. Click the **Available Services** row in the block's property table.
2. In the **Value** field, select an available Web Service from the drop-down list.

Bindings Property

When Composer accesses the Service URL, the available bindings will populate the drop-down list of the Bindings property. To select a binding:

1. Click the **Bindings** row in the block's property table.
2. In the **Value** field, select an available bindings setting from the drop-down list.

Operations Property

When Composer accesses the Service URL, the available operations will populate the drop-down list of the Operations property. To select an operation:

1. Click the **Operations** row in the block's property table.
2. In the**Value** field, select the desired operation from the drop-down list.

Service End Point Property

When Composer accesses the Service URL, the service end point options will populate the drop-down list of the Service End Point property. To select a service end point:

1. Click the **Service End Point row in the block's property table.**
2. In the **Value** field, select the service end point from the drop-down list.

Service End Point Variable Property

Property to parameterize the Service End Point in the Web Service Block. This property will overwrite the 'Service End Point' properties literal value.

Use Protocol Property

When Composer accesses the Service URL, the protocol options (SOAP and HTTP) will populate the drop-down list of the Use Protocol property. To select a protocol:

1. Click the **Use Protocol** row in the block's property table.
2. In the **Value** field, select SOAP or HTTP from the drop-down list.

Condition Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Logging Details Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Log Level Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Enable Status Property

Find this property's details under Common Properties for Callflow Blocks or Common Properties for Workflow Blocks.

Input Parameters Property

Note: The Web Service block won't work with IRD if the Web Service parameters are named double since URS considers it a reserved keyword. The same Web Service block will work fine in the voice application. After you have chosen the available service and operations which you want to invoke, along with bindings, service end point, and protocol, use the Input Parameters property to specify a list of required Name/Value pairs to pass as parameters to the Web Service URL. To specify input parameters:

1. Click the **Parameters** row in the block's property table.
2. Click the ... button to open the Parameter Settings dialog box.

Add Button

Use the **Add** button to enter parameter details.

1. Click **Add** to add an entry to Web Service Parameters.
2. In the **Parameter Name** field, accept the default name or change it.
3. From the **Parameter Type** drop-down list, select **In**, **Out**, or **InOut**:

In	Input parameters are variables submitted to the web service.
Out	Output parameters are variables that the web service returns and will be reassigned back to the current callflow/workflow.
InOut	InOut parameters are parameters that act as both input and output.

4. In the **Expression** drop-down list, select from among the variables shown, type your own expression, or click the ▭ button to use Skill Expression Builder.
5. In the **Definition** field, type a description for this parameter.
6. Click **Add** again to enter another parameter, or click OK to finish.

Delete Button

To delete a parameter:

1. Select an entry from the list.
2. Click **Delete**.

Timeout Property

Select the variable containing the number of seconds that the application will wait when fetching the result of the Web Service or the Web Request. If the requested resource does not respond in that time, then a timeout event will occur.

Custom HTTP Headers Property

Use this property to add Custom headers to be sent along with the HTTP request during the runtime execution of the Server Side block.

1. Click the row in the block's property table.
2. Click the ▭ button to open the Custom HTTP Headers dialog box.
3. Click **Add** to open Configure Custom HTTP Headers dialog box.

Note: The list of headers is a standard list defined by the HTTP protocol. You can optionally specify a list of headers. For each header, the name can be selected from the drop down list or keyed in. The value can be specified as literal values or as variable. There is no special format.

1. Select a **Header type**.
2. Select **Literal** or **Variable**.
3. Type the literal value or select the variable that contains the value.

Authentication Type Property

To assign a value to the Authentication Type property:

1. Select the **Authentication Type** row in the block's property table.
2. In the **Value** field, select from the following:

   - **Anonymous**--With the anonymous type of access, no user name/password is passed to Web service for client authentication in order to get data.
   - **HTTP Basic Authentication**--HTTP Protocol level Basic Authentication using Authorization header. If you select the basic type of access, you must supply the Login Name and Password properties.
   - **SOAP Message Level Basic Authentication**--SOAP Message level Basic Authentication for legacy Web Services using <BasicAuth> header.-- Rarely used but for compatibility.
   - **SOAP XML Signature Authentication**--SOAP Message level XML Digital Signature Authentication using Client Certificate.
   - **SOAP Signature with HTTP Basic Authentication**--SOAP Message Level XML Digital Signature Authentication using Client Certificate + HTTP Basic Authentication (for the Web Server level).

Important

Basic HTTP Authentication properties in the Web Service block are validated only during runtime in the server-side pages (ASPX/JSP). For design time, WSDL parsing authentication is not supported. You can copy the WSDL file to the **Include** folder within the required **Composer Project** folder and specify `include/<filename.wsdl>` in the **Service URL** property to parse the WSDL file and configure the block.

Login Name Property

The Login Name property specifies the login name for the invoked web page. To provide a login name for the web request:

1. Select the **Login Name** row in the block's property table.
2. In the **Value** field, type a valid login name.

Password Property

The Password property specifies the password for the invoked web page. To provide a password for the web request:

1. Select the **Password** row in the block's property table.
2. In the **Value** field, type a valid password that corresponds to the login name above.

Certificate Store Name Property

Use this property to specify the name of the Windows Certificate Store. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate Store Name** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Certificate Alias Property

Use this property to specify the Client Certificate Name. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate Alias** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Certificate or Key Store Location Property

Use this property to specify the location of the Certificate Store or Key Store. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Certificate or Key Store Location** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Key Algorithm Property

Select DSA (default) or RSA to specify the Key Algorithm to sign the SOAP Digital Signature. See Web Service Block and Signed SOAP Requests. Use this property to specify the Key Store Password. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Key Algorithm** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Key Store Password Property

To select a variable:

1. Select the **Key Store Password** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

Private Key Property

Use this property to specify private key of the Client Certificate. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Private Key** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Private Key Password Property

Use this property to specify the private key password. See Web Service Block and Signed SOAP Requests. To select a variable:

1. Select the **Private Key Password** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list.

Custom Prefix Property

Use this property to set custom namespace to the generated SOAP message tags. If this property is set it will overwrite the default / WSDL namespace prefix.

Note: To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.

Add Namespace Prefix Property

Use this property to add Namespace prefix to the generated SOAP message. By default Composer Web Service client doesn't generate namespace prefixes.

1. None - Do not add any namespace prefix to the SOAP:Body elements.
2. Method Name Tag Only - Add namespace prefix only to the Method Name tag (Operational name tag).

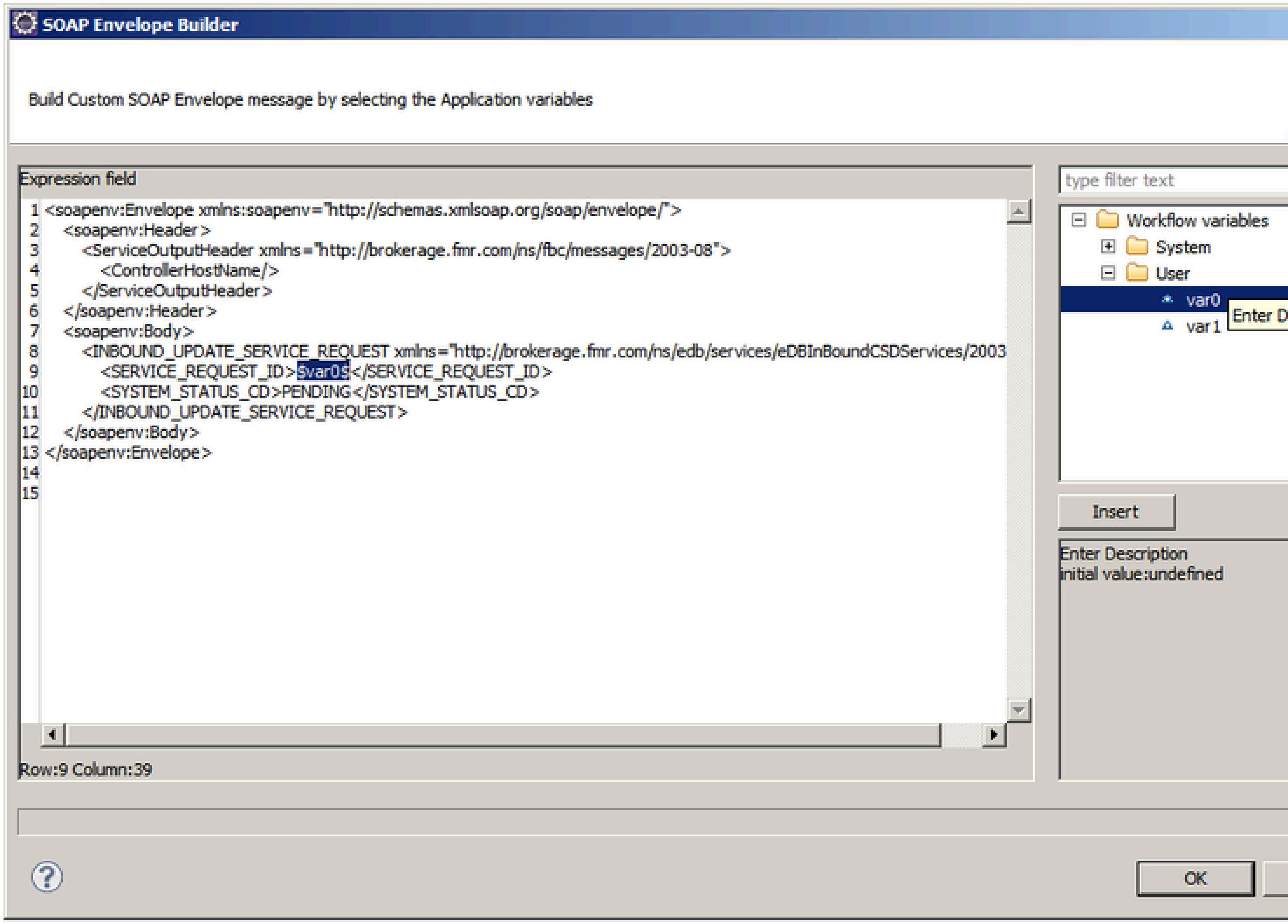
3. Method Name Tag and Child Tags - Add namespace prefix to all the tags in the SOAP message.

Note: To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.

Custom SOAP Envelope Property

Use this property to set Custom SOAP Envelope messages. If this property is set, the Composer Web Service run-time client will use this message to get a Web Service response.

1. Click the **Custom SOAP Envelope** property under the **SOAP Message Generation** category. The Custom SOAP Envelope dialog is displayed.
2. Add the custom SOAP Envelope message (the message can be generated using any Web Service client tool).



Build Custom SOAP Envelope message by selecting the Application variables

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <ServiceOutputHeader xmlns="http://brokerage.fmr.com/ns/fbc/messages/2003-08">
      <ControllerHostName/>
    </ServiceOutputHeader>
  </soapenv:Header>
  <soapenv:Body>
    <INBOUND_UPDATE_SERVICE_REQUEST xmlns="http://brokerage.fmr.com/ns/edb/services/eDBInBoundCSDServices/2003
      <SERVICE_REQUEST_ID>$var0$</SERVICE_REQUEST_ID>
      <SYSTEM_STATUS_CD>PENDING</SYSTEM_STATUS_CD>
    </INBOUND_UPDATE_SERVICE_REQUEST>
  </soapenv:Body>
</soapenv:Envelope>
```

Row:9 Column:39

Enter Description
initial value:undefined

Custom SOAP Envelope Dialog

The custom message is sent to the Web Service URL at run-time. Diagram application variables can be used to form dynamic contents. Variables can be used in the custom message with a **$<Variable_Name>$** notation.

Note:

1. To access this property, ensure that the **Show Advanced Properties** option is selected on the toolbar.
2. This property is supported for both Java Composer Projects and .NET Composer Projects.
3. Callflow-Root document variables and Workflow-Project variables are not supported in this property.

Output Result Property

When the Map Output Values to Variables property above is set to true, the Output Result property maps the Web Service response keys to AppState variables. If Map Output Values to Variables is set to false, the entire Web Service response will be assigned to a variable. The Output Result property is the variable used to get back the invoked Web Service result. To select a variable:

1. Select the **Output Result** row in the block's property table.
2. In the **Value** field, select one of the available variables from the drop-down list. Does not need to match the variable name that is coming back as a result of the web request.

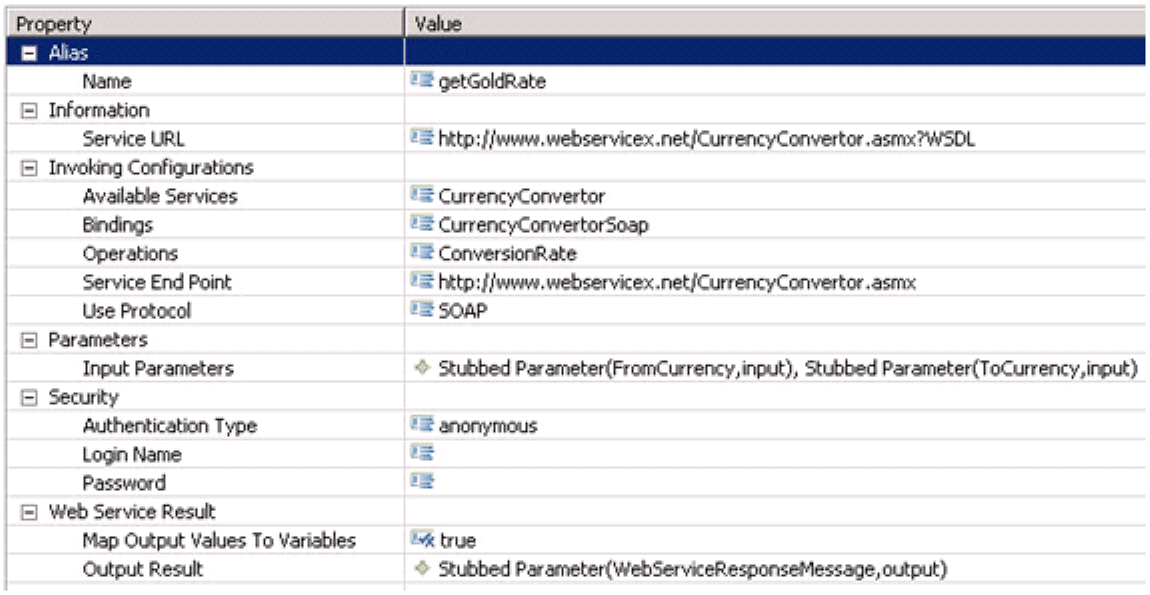
Map Output Values to Variables Property

The Map Output Values to Variables property indicates whether or not to map the Web Service response keys to `AppState` variables. To select a value for the Map Output Values to Variables property:

1. Select the **Map Output Values to Variables** row in the block's property table.
2. In the **Value** field, select true or false from the drop-down list.

Example Block Properties

Example properties for a Web Service block are shown below.



Property	Value
⊟ Alias	
Name	getGoldRate
⊟ Information	
Service URL	http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
⊟ Invoking Configurations	
Available Services	CurrencyConvertor
Bindings	CurrencyConvertorSoap
Operations	ConversionRate
Service End Point	http://www.webservicex.net/CurrencyConvertor.asmx
Use Protocol	SOAP
⊟ Parameters	
Input Parameters	Stubbed Parameter(FromCurrency,input), Stubbed Parameter(ToCurrency,input)
⊟ Security	
Authentication Type	anonymous
Login Name	
Password	
⊟ Web Service Result	
Map Output Values To Variables	true
Output Result	Stubbed Parameter(WebServiceResponseMessage,output)

ExampleWebService.gif

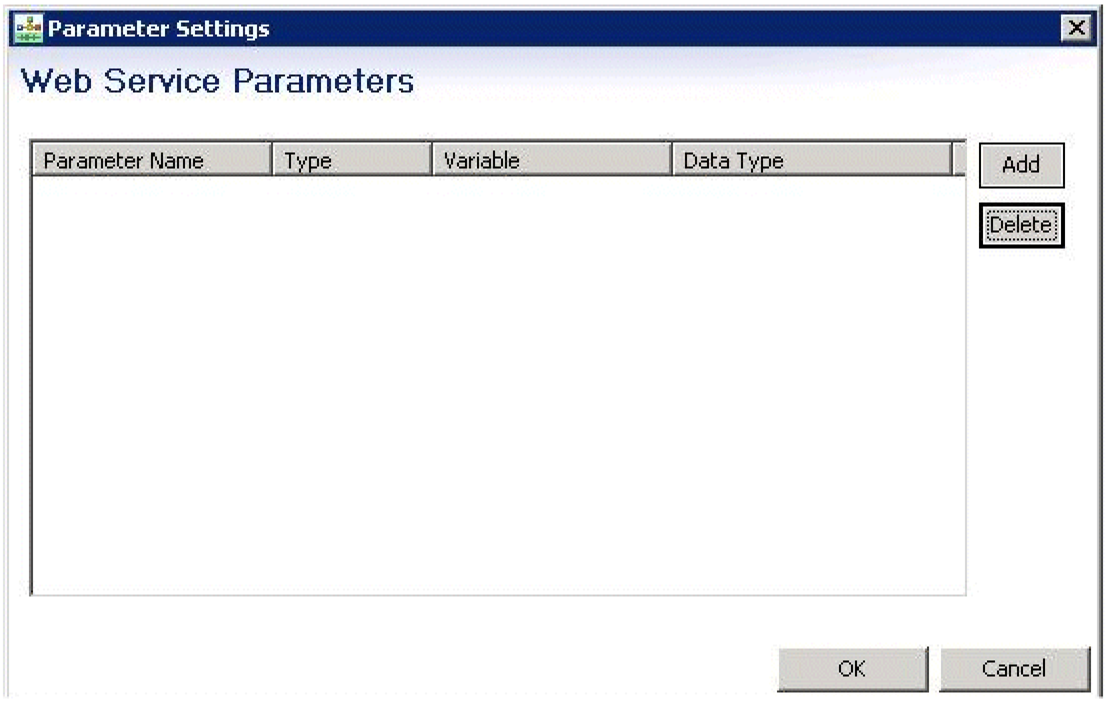
Web Services Description Language (WSDL) Support

Composer supports WSDL definitions conforming to the version WSDL 1.1 schema.

Errors in WSDL Parsing

The following Composer symptom may indicate errors in WSDL parsing:

- If the WSDL definition contains any of the unsupported types and elements, Composer may not be able to parse the WSDL correctly to identify the input and output parameters of the Web Service.
- If the Composer WSDL parser is unable to properly parse the WSDL definition for the Web Service, the input and output parameters fields in the Web Service Parameters dialog box might be empty, with no pre-configured parameters as shown below.



Workarounds

Currently, the following workarounds are available to change the schemas to work with Composer:

- Use qualified elementFormDefault (elementFormDefault="qualified") and define types with fully qualified namespace definitions.
- Define all wsdl types in one schema.
- Replace reference attributes with the actual types being referenced.
- Use the Add/Delete buttons to add or remove any parameters that may not have been automatically displayed. The SOAP request that will be generated at application runtime will take these changes into account.

Note

Composer Web Service Block generated SOAP messages does not have prefix in the SOAP elements. Web Services created using Metro / JAX-WS framework may return Null Pointer Exception or Unexpected Result due to the prefix limitation. Updating the JAX-WS API's / GlassFish server / Metro WS Framework to latest versions may help.

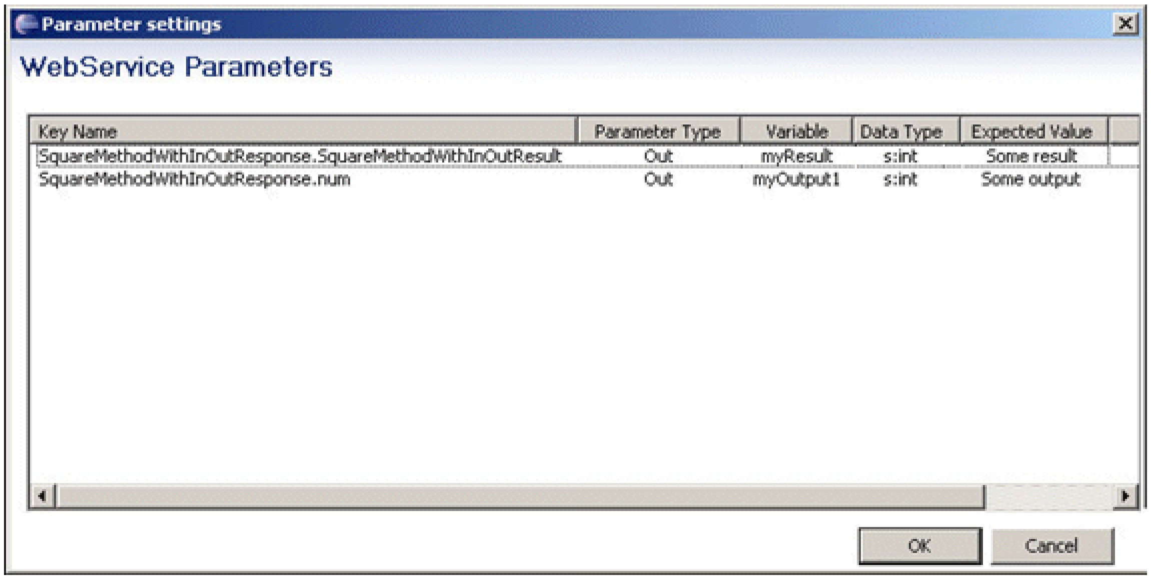
Web Service Stubbing

Composer's Web Services stubbing feature allows you to work with Web Services in off-line mode when you do not have access to the Web Service itself or if the Web Service is under development. This feature is intended to be used in a test environment. It is not intended for a production environment unless there is need to remove an active Web Service from a callflow for debugging purposes.

Using Web Services Stubbing

To use Web Services stubbing:

1. To enable stubbing, add the variable COMPOSER_WSSTUBBING to your Entry block and set its value to 1 indicating stubbing is turned on (0 = stubbing is turned off). In Composer 8.0.2 and later, this variable is present by default in the Variables Setting dialog box, which opens from the Entry object.
2. Create the Web Service block.
3. Place the Web Service Description Language (WSDL) file in your Project. The assumption is that the WSDL file for the Web Service is available at all times.
4. For the Service URL property, use a local URL to the WSDL file. When the Web Service is ready to be used, change this local URL to the correct URL.)
5. To specify the expected output value for the Web Service result as well as any output parameters, use the Output Result property of the Web Service block. An example is shown below.



Key Name	Parameter Type	Variable	Data Type	Expected Value
SquareMethodWithInOutResponse.SquareMethodWithInOutResult	Out	myResult	s:int	Some result
SquareMethodWithInOutResponse.num	Out	myOutput1	s:int	Some output

Using the above examples:

- If stubbing is on, the myResult variable will be assigned the value Some result and myOutput1 will be assigned the value of Some output.

- If stubbing is off, the value returned by the Web Service will be stored in these two variables.

Limitation

Web Service stubbing currently does not support auto-synchronization of output parameters in case of Web Services with complex return types.

Web Service SOAP Messages

Use the examples below when configuring the Web Service block.

SOAP Message Level Basic Authentication

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
              <SOAP-ENV:Header>
                           <h:BasicAuth xmlns:h="http://soap-authentication.org/basic/2001/
10/" mustUnderstand="1">
                                       <Name>UserName</Name>
                                       <Password>Pass</Password>
                          </h:BasicAuth>
              </SOAP-ENV:Header>
              <SOAP-ENV:Body>
                            <p:getNumber xmlns:p="http://webservice.com"/>
              </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Signed Using Client Digital Certificate (DSA Key Algorithm)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" SOAP-ENV:mustUnderstand="1">
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-2">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="SOAP-
ENV"/>
</ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<ds:Reference URI="#id-1">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"><ec:InclusiveNamespaces
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"  PrefixList=""/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#sha1"/><ds:DigestValue>yMmgdHRevOnFPGtnSZx4JV9hiuI=</ds:DigestValue></ds:Reference></ds:SignedInfo><ds:
Id="KI-8D7856F18A7AB8CF5613098009924472">
<wsse:SecurityTokenReference wsu:Id="STR-8D7856F18A7AB8CF5613098009924473">
<wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-
profile-1.0#X509v3">MIICwDCCAn2gAwIBAgIETfBxZzALBgcqhkjOOAQDBQAwQzELMAkGA1UEBhMCVVMxDDAKBgNVBAoTA1N1bjERMA8GA1U
yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/
IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/
```

```
mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTD
5Ykhco6lMBRRncJwGuWB/mFPhaX8Odfj8NMEih1+ICIjhVwGk1p6P3Gu2Dm+45TYJCxBktdOlU0uy/
Uj8E61NZSaeQL9WA4gGz5Hb5uMAsGByqGSM44BAMFAAMwADAtAhQqfbMb9hd1vpBAAJntCDSOY5uP2AIVAJGy1E7Zx4268n3fD34gLcpkZoKc</
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="id-1">
<p:methodName xmlns:p="http://example.com"><key>value</key></p:methodName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Signed Using Client Digital Certificate (RSA Key Algorithm)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" SOAP-ENV:mustUnderstand="1">
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-2">
<ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="SOAP-
ENV"/>
</ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#id-1">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"><ec:InclusiveNamespaces
xmlns:ec=
"http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList=""/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>yMmgdHRevOnFPGtnSZx4JV9hiuI=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>MX9c9C5Tpfvp32e2pPjkCv4ycZhcuZVMFHo8DlGKWi331fnG3oqXLg==</ds:SignatureValue>
<ds:KeyInfo Id="KI-8D7856F18A7AB8CF5613098009924472">
<wsse:SecurityTokenReference wsu:Id="STR-8D7856F18A7AB8CF5613098009924473">
<wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-
profile-1.0#X509v3">MIICwDCCAn2gAwIBAgIETfBxZzALBgcqhkjOOAQDBQAwQzELMAkGA1UEBhMCVVMxDDAKBgNVBAoTA1N1bjERMA8GA1U
yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/
IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/
mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTD
5Ykhco6lMBRRncJwGuWB/mFPhaX8Odfj8NMEih1+ICIjhVwGk1p6P3Gu2Dm+45TYJCxBktdOlU0uy/
Uj8E61NZSaeQL9WA4gGz5Hb5uMAsGByqGSM44BAMFAAMwADAtAhQqfbMb9hd1vpBAAJntCDSOY5uP2AIVAJGy1E7Zx4268n3fD34gLcpkZoKc</
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
```

```
utility-1.0.xsd" wsu:Id="id-1">
<p:methodName xmlns:p="http://example.com"><key>value</key></p:methodName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Signed SOAP Requests

The Web Service block enables Composer applications to invoke Web Services, which require message-level authentication. The message level security support provided by the Web Service block is limited to one-way signed SOAP requests from the Composer application to the Web Service. Web Services can then verify that the request received from a Composer application includes a valid certificate.

Prerequisites

The prerequisites are:

- Web Service is able to verify only the signature (Timestamp, UsernameToken and Encryption are not supported).
- Web Service sends an unsigned response, i.e., Web Service is not configured to process outgoing response (only InflowSecurity is configured).
- X.509 certificate for the client is available and is trusted by the Web Service. Certificates can be purchased from a certificate authority or can be generated (for testing) using tools, such as OpenSSL.
- Certificates should be based on one of the supported encryption algorithms, RSA or DSA.
- Certificate Stores:
- For Java projects, certificates and keys should be available in a Java Keystore (*.jks file). OpenSSL and Keytool (available in JDK 1.6) can be used to create and import certificates.
- For .NET projects, certificates and keys should be available in the Windows Certificate Store. OpenSSL can be used to create certificates and Certificates (snap-in in Microsoft Management Console) can be used to import certificates.
- For .NET projects, WSE 3.0 (runtime) should be installed on the machine running Composer.

Enabling Signing of SOAP Messages

To enable signing of SOAP messages, set the Authentication Type property in the Security section to one of the following values

- SOAPDigitalSignatureAuthentication -- for signing messages when not using HTTP Basic authentication.
- SOAPSignatureWithHTTPBasicAuthentication -- for signing messages when used along with HTTP Basic Authentication (Security Basic Authentication Credentials section is specified)

Once enabled to sign the request, the application will need information regarding the public key (certificate) and private key (key) as below:

- Certificate Store Name (.NET only) -- Windows Store Name containing the client certificate and private key. Value should be one of the following predefined Windows Certificate Stores or the name of a custom Store in which the certificate and key are stored. Note that this Store should contain the client certificate (should include the private key as well).

- AddressBook -- The X.509 Certificate Store for other users.
- My -- The X.509 Certificate Store for personal certificates.
- TrustedPeople -- The X.509 Certificate Store for directly trusted people and resources.
- Certificate Alias -- Alias that identifies the certificate and key in the Store. For .NET projects, this refers to the subject of the certificate, e.g., CN=ComposerCertificate.
- Certificate or Key Store Location -- Path to the Certificate Store location containing the certificate and private key. In .NET, the value should be set to one of the following:
- StoreLocation.LocalMachine (default when value is not one of these)
- StoreLocation.CurrentUser
- Key Algorithm -- Algorithm to be used for encryption. This is the same as the algorithm that was specified when the key was generated; usually received from the certificate authority issuing the certificate.
- Key Store Password (Java only) -- Java Key Store password for the key store specified as the key store location.
- Private Key Alias (Java only) -- Alias by which the private key is identified in the key store.
- Private Key Password (Java only) - Password to access the private key to be used when signing a message. For .NET projects, it is expected that the password be stored as part of the settings for the certificate.

At run time, the Composer application will create a SOAP message and then sign it using its private key. The signed message will include an encrypted signature in the SOAP header and the SOAP request as the body. This signed message is sent to the Web Service for processing. Web Service will decrypt the signature using the client certificate (public key previously imported into the Web Service certificate store) and hence authenticating that the source of the request is valid.

Signature Validation Failure Causes

Signature Validation by the Web Service may fail for the following reasons:

- Syntax of request (signature) doesn't conform to the policy enforced by the Web Service. Example: Timestamp is required by the Web Service but was not included in the request because Composer doesn't support Timestamp policy.
- Validation of signature failed. Example: Web Service uses RSA key, but the request was signed using DSA key.
- Application validation policy rejects the request. Example: Signature created by an untrusted key.

Once signature validation is successful, the Web Service will process the request and then send the unsigned response back to the Composer application. Composer processes the response without signature validation. The above will ensure that Web Services will process requests only from legitimate clients, the Composer application being one of them.

Connection and Read Timeout

By default, the Web Service and Web Request blocks use 20 seconds each for the connection timeout and read timeout. The following steps describe how to configure the timeouts:

1. Create a new folder called `WEB-INF` inside the Composer Project.
2. Create a property file named `composer.properties` inside the `WEB-INF` folder.
3. Add the following properties (case-sensitive) to composer.properties with the timeout values:
    - `web.connectionTimeout=40000`
    - `web.readTimeout=40000`

The value specified should be in milliseconds. The connectionTimeout property is a specified timeout value, in milliseconds, to be used when opening a communications link to the resource referenced by the URL. The readTimeout property is a nonzero integer value, in milliseconds, to be used when reading from an input stream when a connection is established to a URL resource.

Server-Side Troubleshooting

The table below lists Server Side block troubleshooting situations and steps to remedy.

Situation	Block	Steps to Troubleshoot
I entered the Service URL but getting a pop-up with Check the Web Service URL	Web Service Block	Verify that the WSDL definition is accessible in a web browser. Check the Composer logs for possible errors in fetching the WSDL. Location: <workspace>\.metadata Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
I entered the Service URL and I can choose the SOAP operations, but the parameters do not show up in the dialogs	Web Service Block	Verify that the WSDL definition is accessible in a web browser. Check the Composer logs for possible errors in fetching the WSDL. More details can be found in the logs in the following location: <workspace>\.metadata\.plugins\com.genesyslab.studio.model folder Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
Using the Web Services Explorer utility	Web Service Block	The Web Services Explorer is a JSP Web application hosted on the Apache Tomcat servlet engine contained within Eclipse. The Web Services Explorer is provided with Composer and allows you to explore, import, and test WSDL documents. Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
Errors during runtime	Web Service Block Web Request Block Backend Block	Check the Composer logs for possible errors in fetching the WSDL. Check the backend logs. For ASP.NET projects, check the IIS logs For Java Projects, check the Tomcat standard logs. Check that the WSDL definition is accessible and test with the Web Services Explorer utility as described in the Troubleshooting section.
I copied my callflow/workflow from one project to another but my Backend block does not work	Backend Block	Check that any custom backend libraries or applications have also been copied to the new project.

Backend block		

Logs:

- Java Composer Projects Server Side Backend logging can be controlled using the `log4j.xml` file present in the `$ComposerInstalledLocation\tomcat\lib` folder.
- For DotNetComposer Projects web.config file can be used to control the Server Side logging.
- Java Composer Projects exported as WAR files will have the `log4j.xml` bundled inside the `WEB-INF\lib` folder. If the `log4j.xml` configuration format is not working, you can add a `log4j.properties` in the `tomcat/webapps/<application name>/WEB-INF/classes` folder.

Notes:

- Service URL has to end with wsdl or WSDL
- Cannot use - or other reserved characters in the Entry block for a variable value. Enter the value directly in the input parameters dialog by typing the value in the Expression column as a string; example: 'atm near 37.771008, -122.41175'

Common Functionality

This section describes the Functionality that can be used for both Orchestration routing and GVP voice applications.

Note: Composer does not support copying blocks from a callflow to a workflow or vice-versa.

You can also create Custom Blocks.

Special Note on Entry and Exit Blocks

These two blocks are also present on the palette for both routing and voice applications. The Entry block for routing applications has a different set of events than the Entry block for voice applications.

Link Tools

Routing and voice applications use the same link tools:

- Tools Detail Outlink
- Tools Detail Exception Link

Composer Help Wiki URL

The URL to the Composer Help wiki is configurable by using the Online Wiki URL field: **Window** > **Preferences** > **Help**. The default works with English but if, for example, Japanese pages were available in a different location, then you could change the URL accordingly.

Code Generation

The process of generating code creates a properly-formatted VoiceXML file from a callflow diagram built with Composer or a SCXML file from a workflow diagram. Static pages (pure VXML or SCXML code) are generated in the src-gen folder of the Composer Project.

Generating Code

You can generate code in a couple of ways:

- Select **Diagram** > **Generate Code**.
- Click the Generate Code icon on the upper-right of the Composer main window when the callflow/ workflow canvas is selected.

Note: If your project uses Query Builder or Stored Procedure Helper-generated queries in DB Data Blocks, the process of code generation creates one SQL file in the db folder for each such DB Data block. These SQL files will be used at runtime and should not be deleted.

Code Generation of Multiple Callflows

When using the Run as Callflow function, Composer automatically generates the VXML files from the diagram file that you want to run. When generating code, with the generate code function for a Java Composer Project that has multiple callflows, Composer attempts to generate the VXML for all the callflows before running (because the application might move between multiple callflows for subdialogs).

However, if one of the callflows has an error, Composer provides the option to continue running the application anyway, because the erroneous callflow may be a callflow that ia not used by the one being run (if there are two or more main callflows, for example). When this happens, the VXML files are basically out of sync with the diagram files and this may affect execution. Genesys recommends that you fix all errors before running the application.

Generate All

Use the 'Generate All' feature to generate VoiceXml and SCXML code for all the Diagram files (Callflows, Workflows and IPD) present in a Composer Project.

Generate All can be invoked in two ways.

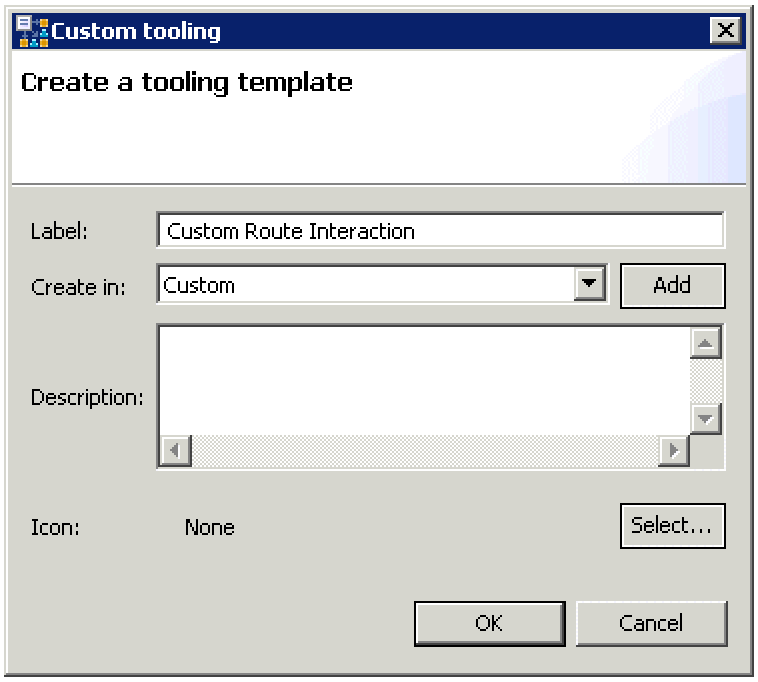
- Select **Project** > **Generate All**.

- Click the Generate All icon on the upper-right of the Composer main window toolbar.

Custom Blocks

A custom block can also be thought of as a pre-filled template for a block. To create a custom block:

1. Add the block to use as the pre-filled template.
2. Fill out the properties to be the basis of the template.
3. Right-click the block to bring up a shortcut menu.
4. From the shortcut menu, select **Add as custom tool**. Note this option is not available for the IPD blocks. The Custom Tooling dialog box opens. A sample is shown below.



Label: Custom Route Interaction
Create in: Custom
Description:
Icon: None

5. Opposite **Create in:**, Leave **Custom** to place the block in the Custom category on the palette. Or click the down arrow and select another palette drawer.
6. If you wish to upload an icon or graphic for the custom block, click **Select** to open the Custom Icon Library dialog box. Select an icon, and click OK. The selected icon appears in the dialog box.
7. Click **OK** in the Custom Tooling dialog box. Composer adds the block to the palette in the selected category.

Note: Custom Entry and Exit icons do not appear on the palette. It still shows the default Entry/Exit icons.

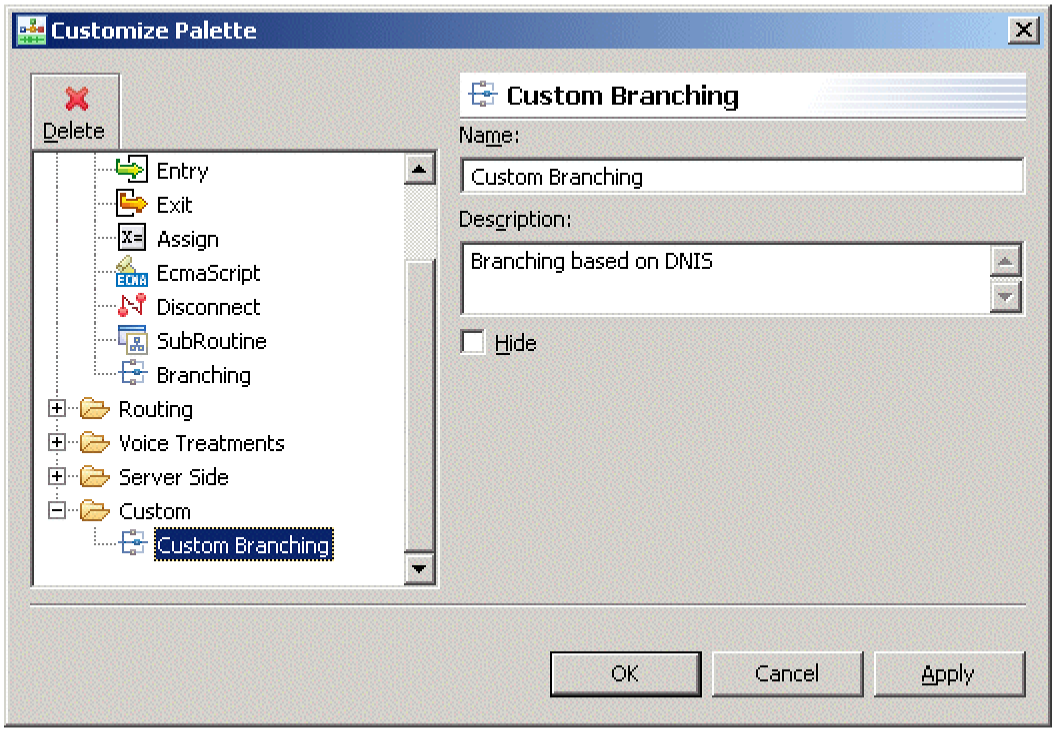
Using a Custom Block

Use the block as you would any others. Fields defined as relevant in the Block Data Map are automatically populated in the Properties tab based on the state of the original block when you made the tooling.

- Note: When you create a new block, you must add a mapping to define which fields should be template-aware.

Changing Existing Custom Blocks To change an existing custom block:

1. Within the palette, right-click a custom block and select Customize... from the context menu. This opens the Customize Palette dialog box.
2. Navigate to/select the block you want to change.



Name: Custom Branching

Description: Branching based on DNIS

3. You can:

    - Change the label.
    - Edit the description.
    - Delete the block entirely.

Deleting a Custom Block

To delete a custom block:

1. Within the palette, right-click a custom block and select **Customize…** from the context menu. This opens the Customize Palette dialog box shown above.
2. Navigate to/select the block you want to delete.
3. Click the **Delete** button at the top of the Customize Palette dialog box.
4. Click **OK**.

Hiding the Custom Category

To hide the Custom category on the palette:

1. In the Customize Palette dialog box (see above steps), select the Custom folder.
2. Click the **Hide** checkbox. Later, if you need the Custom category back on the palette, select the Custom folder and un-click **Hide**.
3. Click **OK**.

Import/Export of Custom Blocks

To import or export a custom block so that it can be shared across multiple users/Composer installations:

1. Select the custom block in the palette.
2. From the Diagram menu, select:

    - **Import Custom Blocks…** to open the Select Custom Tooling Definition dialog box. Browse to the location for the previously exported custom block file, which will have a *.ctooling extension, select the file, and click **OK.**
    - **Export Custom Blocks...** to open the Create Custom Tooling Definition dialog box. Name the file, keep the file type as custom tooling, and click **OK**. The custom block is saved as file with a *.ctooling extension.

Customization Manager

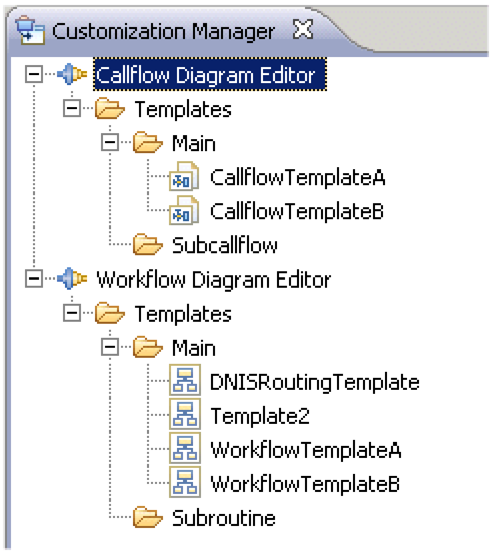
Purpose

Use Customization Manager to store/manage various aspects of your Composer installation that you have customized. You can store/manage the workflow and callflow diagram templates that you create as described in the Diagram Templates topic. topic.

Interface

The user interface takes the form of the Customization Manager view. To display this view:

1. Select **Window** > **Show View** > **Other**.
2. In the resulting Show View dialog box, select Customization Manager and click **OK**. A Customization Manager view appears at the bottom of the Composer window listing registered plug-ins. An example is shown below.



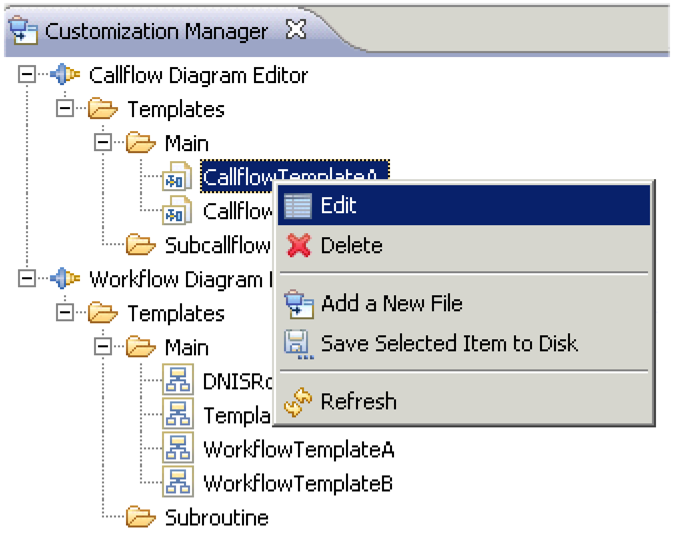
Customization Manager

- Callflow Diagram Editor
  - Templates
    - Main
      - CallflowTemplateA
      - CallflowTemplateB
    - Subcallflow
- Workflow Diagram Editor
  - Templates
    - Main
      - DNISRoutingTemplate
      - Template2
      - WorkflowTemplateA
      - WorkflowTemplateB
    - Subroutine

Managing Templates

You can use this view to manage the workflow and callflow diagram templates that you create as described in the Saving Diagrams as Templates topic. To manage diagram templates:

1. Right-click a callflow or workflow diagram to bring up a menu.



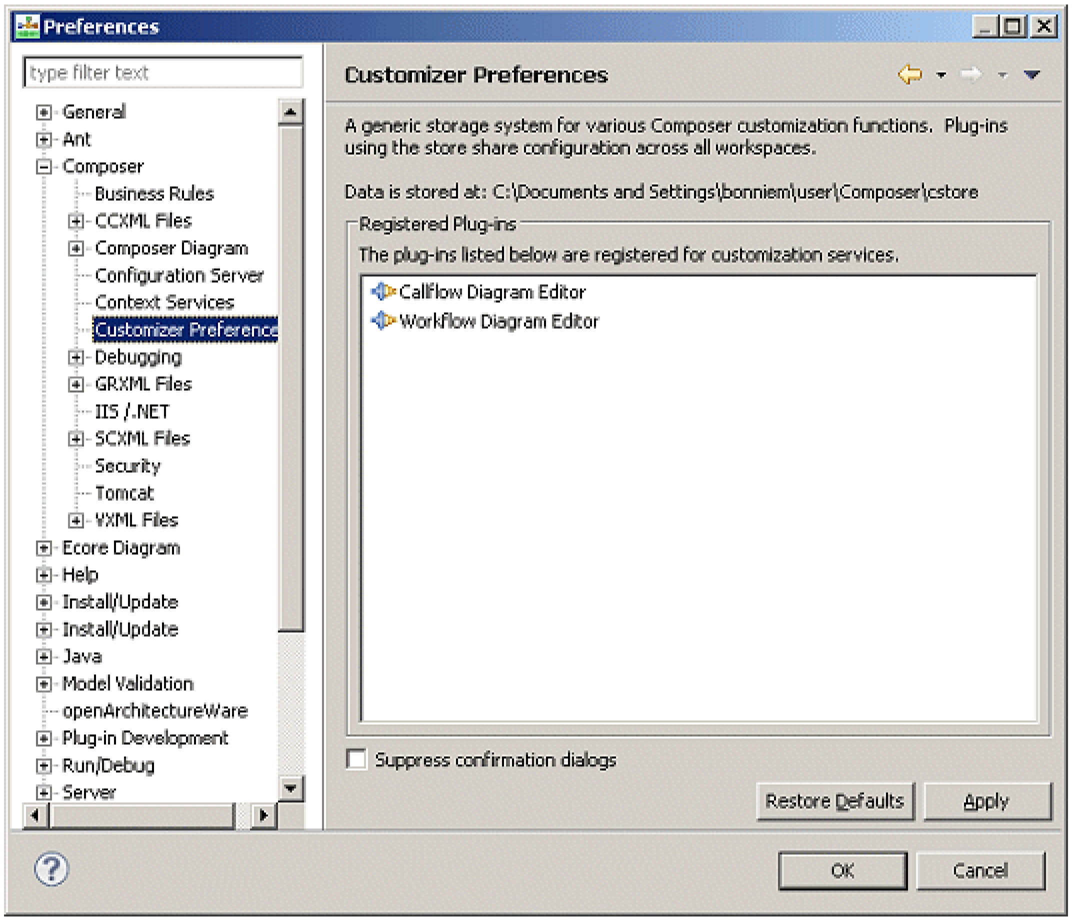
2. Select from the following:

   - **Edit**--Use to edit a selected diagram template.
   - **Delete**-Use to delete a selected diagram template
   - **Add a New File**--Use to import a callflow or workflow diagram.
   - **Save Selected Item to Disk**--Use to save a selected diagram to a disk.
   - **Refresh**

Customizer Preferences

To bring up Customizer Preferences:

1. Click the **Window** menu.
2. Select **Preferences**.
3. Expand **Composer**.
4. Select **Customizer Preferences**. The Customizer Preferences dialog box opens. The Callflow Diagram Editor and Workflow Diagram Editor customization plug-ins are display only and can be used for debugging.



A generic storage system for various Composer customization functions. Plug-ins using the store share configuration across all workspaces.

Data is stored at: C:\Documents and Settings\bonniem\user\Composer\cstore

The plug-ins listed below are registered for customization services.

Callflow Diagram Editor
Workflow Diagram Editor

☐ Suppress confirmation dialogs

The Customizer Preferences dialog box:

- Reports on the location of the storage area (`cstore` directory) on disk. Diagrams that you save as templates are stored here.
- Lists registered plug-ins as shown in the Customization Manager view.
- Allows you to suppress confirmation dialogs associated with plug-ins. If checked, it suppresses the success/failure indicator message when you save a diagram as a template.

Diagram Preferences

Select **Window**> **Preferences** > **Composer** > **Composer Diagram**. The following preferences for diagrams can be set in the Preferences dialog box:

Global Settings

1. Select or clear the check box for each of the following diagram global settings:

    - **Show Connection Ports**. If enabled, connection ports (both exception ports and out ports) are always displayed on blocks. This makes it convenient to draw links between blocks and to get immediate feedback on how many ports each block provides. However, in this case, the ability to reposition connections on a block is not available. If switched off, connection ports are not displayed by default, but repositioning or finer control over connection link placement becomes available. Note: This preference applies to all projects and is not available for individual projects.)
    - **Show popup bars**. If enabled, this setting displays basic blocks from the blocks palette in a pop-up bar if you hover your mouse on the diagram for one or two seconds without clicking. Note: blocks are shown in icon view only.)
    - **Enable animated layout**. If enabled, causes diagrams to gradually animate to their location when the Diagram \> Arrange \> Arrange All menu option is clicked.
    - **Enable animated zoom**. If enabled, while using the zoom tools, shows a gradual transition between the initial and final state of the diagram on the canvas. If off, the zoom is instantaneous. Similar behavior for animated layout when the Diagram \>\> Arrange \>\> Arrange All menu option is clicked.
    - **Enable anti-aliasing**. If enabled, improves the appearance of curved shapes in the diagram. You can see its effect on the circles in the Entry and Exit blocks.
    - **Show CodeGen success message**. If unchecked, then the confirmation dialog at the completion of code generation will not be shown.)
    - **Prompt to Save Before Generating Code**. If checked, when you generate code for an unsaved diagram, a prompt appears indicating the diagram has been modified and asking if you want to save the changes before generating code. The dialog box also contains a checkbox: Automatically save when generating code and do not show this message again.
    - **Show Validation success message**. If unchecked, then the confirmation dialog at the time of Validation will not be shown.)
    - **Enable Validation for Prompt Resources**. This preference is used for voice applications. If unchecked, then a validation check for missing prompts is not performed at the time of Validation.
    - **Interaction Process Diagram**. If unchecked, Composer will save Interaction Process Diagrams before publishing.
    - **Prompt to delete Published objects when Interaction Process Diagram is deleted**. If unchecked, Composer will attempt to delete any Published objects when an Interaction Process Diagram is deleted. If Composer is not connected to Configuration Server, object

deletion will not work.

2. Click Apply.

Colors and Fonts

1. Select **Appearance** under Composer Diagram.
2. Click **Change** and make selections to change the default font if you wish.
3. Click the appropriate color icon beside any of the following and make selections to change color:
    - **Font color**
    - **Fill color**
    - **Line color**
    - **Note fill color**
    - **Note line color**
4. Click **Apply**.

Connections

1. Select Connections under Composer Diagram.
2. Select a line style from the drop-down list:
    - **Oblique**
    - **Rectilinear**
3. Click **Apply**.

Pathmaps

1. Select **Pathmaps** under Composer Diagram.
2. Click **New** to add a path variable to use in modeling artifacts, or If the list is populated, select the check box of a path variable in the list.
3. Click **Apply**.

Printing

1. Select **Printing** under Composer Diagram.
2. Select **Portrait** or **Landscape** orientation.
3. Select units of Inches or Millimetres.
4. Select a paper size (default is Letter).
5. Select a width and height (for inches, defaults are 8.5 and 11; formillimeters, defaults are 215.9 and 279.4).
6. Select top, left, bottom, and right margin settings (for inches, defaults are 0.5; for millimeters, defaults are 12.7).
7. Click **Apply**

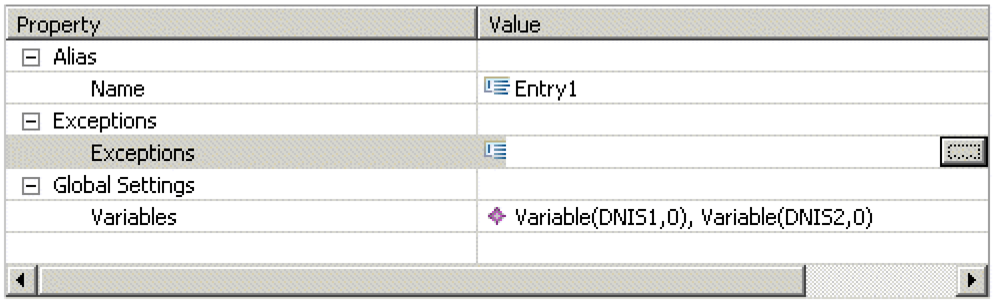
Rulers and Grid

You can make use of rulers and grids when creating diagrams. Rulers and grids can provide a backdrop to assist you in aligning and organizing the elements of your callflow diagrams.

1. Select Rulers and Grid under Studio Diagram.
2. Select or clear the **Show rulers for new diagram** check box (not selected by default).
3. Select ruler units from the drop-down list:
    - **Inches**
    - **Centimeters**
    - **Pixels**
4. Select or clear the Show grid for new diagrams check box (not selected by default).
5. Select or clear the Snap to grid for new diagrams check box (selected by default).
6. Type a value for grid spacing (for inches, the default is 0.125; for centimeters, the default is 0.318; for pixels, the default is 12.019).
7. Click **Apply**.

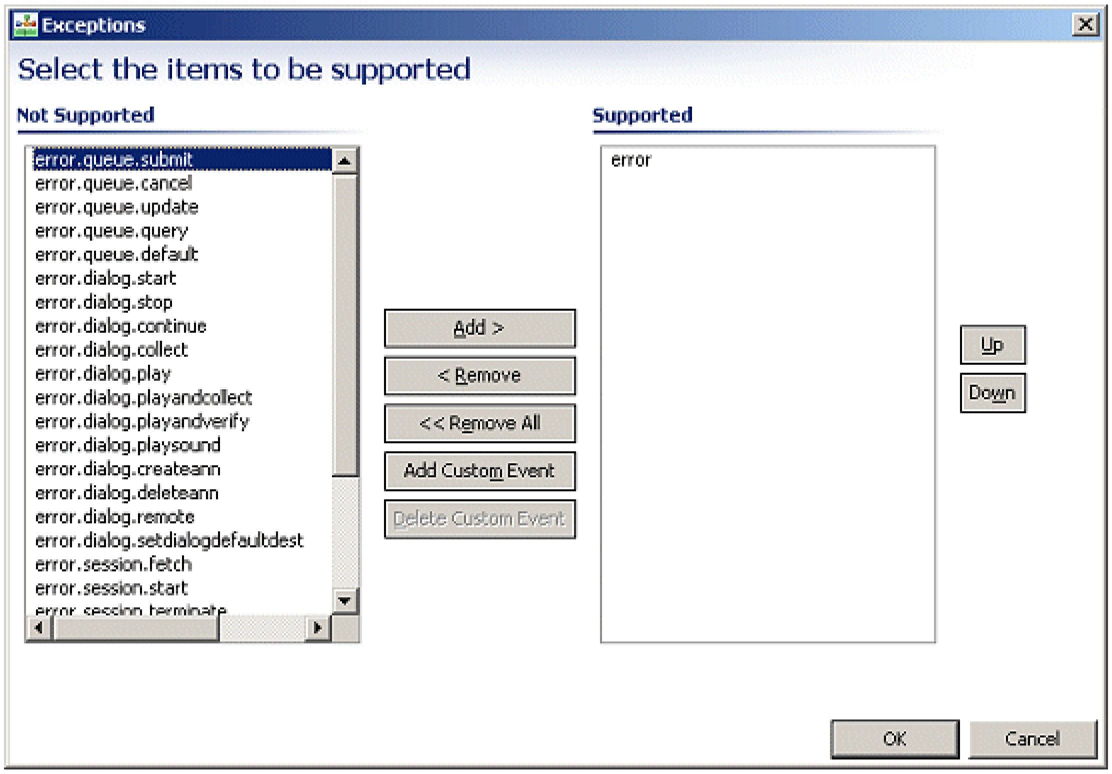
Exception Events

Certain callflow exception events and routing exception events application blocks have an Exceptions property. The Properties view for the routing Entry block is shown below as an example.



Property	Value
⊟ Alias	
Name	Entry1
⊟ Exceptions	
Exceptions	
⊟ Global Settings	
Variables	Variable(DNIS1,0), Variable(DNIS2,0)

Clicking the button opens a dialog box where you can select events to be supported.



Exceptions

Select the items to be supported

Not Supported

```
error.queue.submit
error.queue.cancel
error.queue.update
error.queue.query
error.queue.default
error.dialog.start
error.dialog.stop
error.dialog.continue
error.dialog.collect
error.dialog.play
error.dialog.playandcollect
error.dialog.playandverify
error.dialog.playsound
error.dialog.createann
error.dialog.deleteann
error.dialog.remote
error.dialog.setdialogdefaultdest
error.session.fetch
error.session.start
error.session.terminate
```

Supported

```
error
```

Exceptions.gif

The voice Entry block also has an Exceptions property with a different set of exceptions that can be supported.

Exception Support Levels

Exceptions can be configured at two levels:

1. At the top level in the Entry block using the **Exceptions** property.
2. At the individual block-level for local exception handling using the **Exceptions** property. These block level exceptions may not be present in all blocks.

To support any of these exceptions globally throughout your application:

1. Select the Entry block's **Exceptions** property.
2. Select an event from the **Not Supported** pane, and then click the **>** button to move the event to the **Supported** pane.

Note: If the same exception is defined at the top level and the block level, the block-level exception takes precedence for that block.

Callflow Event Handling Guidelines

1. For Main callflow:

Always handle the default event handlers--disconnect, error, and all.

- If selecting a specific error event type, always place it before any other errors. For example, error.badfetch.http must be placed before error.badfetch, which must be placed before the error event in the application.
- Select specific handlers like error.badfetch only if the application has to handle that exception differently than the generic error or all event handler.

2. For Subcallflows:

- Unless special processing is required at the local level, it is best to handle events at the global level in the Main callflow.

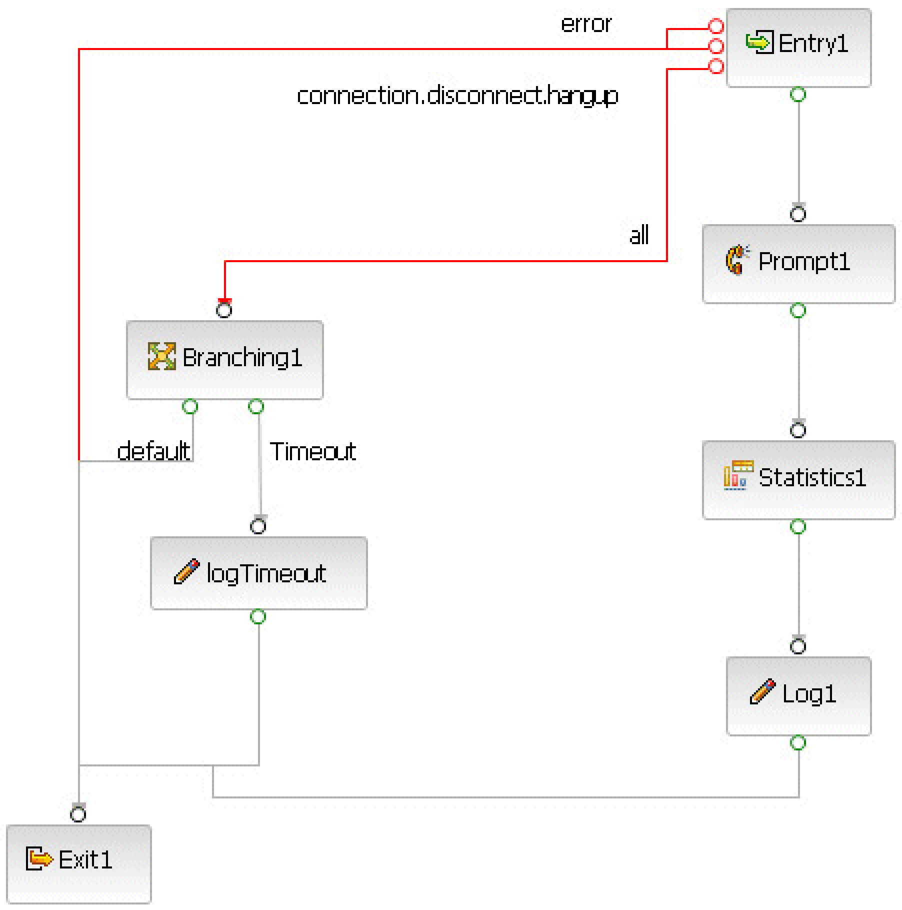
Exception Handling

This section discusses Event Handling and provides some guidelines. The VXML/SCXML interpreters throw exceptions when they encounter errors or conditions for which exceptions are needed so that the condition may be communicated back to the VXML/SCXML application. For example, for voice applications, events such as NOINPUT or NOMATCH in an Input block fall under this category. Composer exposes exceptions at two levels:

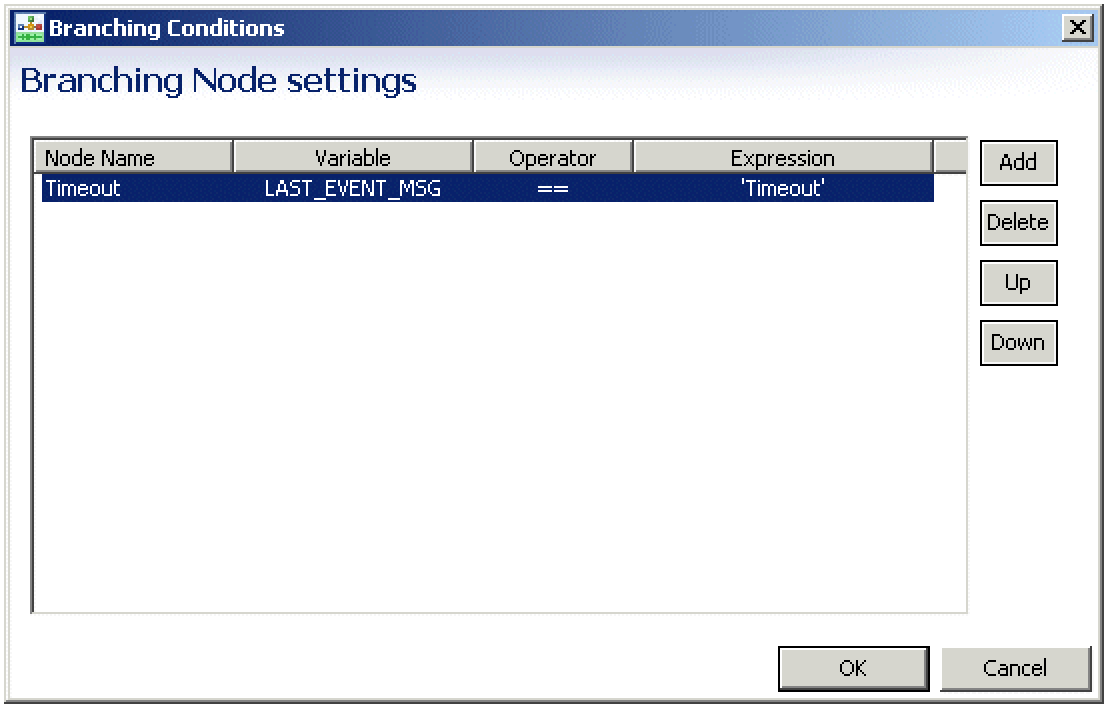
- the block level
- the overall callflow level

Any exceptions exposed at the block level can be handled at the block level itself. If the exception is thrown, control does not wind its way all the way back to the Entry block. For example: the DB Data block exposes a `dbconnectionerror` exception if connection to a database fails. This exception is thrown and handled at the block level itself: `<form id="DBData1">   <block> <data name="DBData1Data" src="../include/dbrequest.aspx" method="post" namelist="AppStateString db_query db_name db_timeout" /> …   <throw event="error.com.genesyslab.composer.dbconnectionerror"   messageexpr="DBData1Data.errorMsg" /> </block> <catch event="error.com.genesyslab.composer.dberror">   <!--handle exception here--> </catch> </form>` The Entry blocks for routing and voice applications also expose a number of exceptions. These are handled in the Entry block itself. The following are some recommendations about good practices in handling exceptions for callflows.

- For the Main callflow, always handle the default event handlers--`disconnect`, `error`, and `all`.
    - If selecting a specific error event type, always place it before any other errors. For example, `error.badfetch.http` must be placed before `error.badfetch`, which must be placed before the error event in the application.
    - Select specific handlers like `error.badfetch` only if the application has to handle that exception differently than the generic error or all event handler.
- For Subcallflows, it is straightforward to handle exceptions that Composer does not expose directly. Any exceptions that are not listed explicitly are thrown as the `all` exception. A branching block can be added to this exception path and a different execution path can be chosen based on the contents of the exception. The conceptual diagram below shows this approach:



The branch condition will look something like this (variables are described in the Entry block):



Node Name	Variable	Operator	Expression
Timeout	LAST_EVENT_MSG	==	'Timeout'

Unless special processing is required at the local level, it is best to handle events at the global level in the Main callflow.

Exception Event Descriptions

The table below names and describes Composer exception events for both callflows and workflows.

Exception Event Name	Description
all	This is a generic catch-all exception handler that will catch any exception not handled by the Entry block. It should be the last catch handler in the sequence of exceptions for a block.
cancel	The caller has requested to cancel playing of the current prompt. (Available when the Universals property includes Cancel.)
com.genesyslab.composer.toomanynoinputs	Occurs when the number of no inputs exceeds the maximum retries in Menu, Input, and Record blocks, and blocks do not have local noinput exception ports.
com.genesyslab.composer.toomanynomatches	Occurs when the number of no matches exceeds the maximum retries in Menu, Input, and Record blocks, and blocks do not have a local nomatch exception port.
com.genesyslab.externalmessage	For handling asynchronous external events. (Available when com.genesyslab.externalevents.enable is set to

	true.)
connection.disconnect.hangup	The caller hangs up. Applies at any time except during blind transfers.
connection.disconnect.transfer	The call was "blind transferred" to another line and will not return.
error	The platform encountered any error, such as error.noresource.
error.badfetch	A fetch of a resource failed due to semantic errors in the VoiceXML page.
error.badfetch.badxmlpage	The page that was fetched is an invalid XML page.
error.badfetch.grammar.load	The platform failed to load a grammar.
error.badfetch.grammar.syntax	A grammar has a syntax error.
error.badfetch.grammar.uri	The platform failed to fetch a grammar uri.
error.badfetch.http	A fetch of a resource failed, and the platform returned an HTTP Response failure code.
error.com.genesyslab.composer.badfetch	Bad request.
error.com.genesyslab.composer.badgateway	Bad gateway.
error.com.genesyslab.composer.customernotfound	Context Services Identify Customer Block, Suppress Customer Not Found Exception Property: If set to false (default), this exception is raised when no matching customer is found.
error.com.genesyslab.composer.dbconnectionerror	This error indicates that there was an error encountered while attempting to connect to the database.
error.com.genesyslab.composer.dberror	All database errors (other than dbtimeout and dbconnectionerror) result in this exception. The error text associated with this exception will contain the exact error returned by the database.
error.com.genesyslab.composer.dbtimeout	This error indicates that there was a timeout while waiting for query results to be received from the database. The timeout period is controlled by the Timeout property
error.com.genesyslab.composer.dbemptyresultset	This error indicates that the query or a stored procedure execution resulted in an empty result set, i.e., No records were returned. If the application will handle this condition and the exception is not required to be thrown then it can be suppressed using the Suppress Empty Result Set Exception property.
error.com.genesyslab.composer.forbidden	Forbidden plus specific error message from the server.
error.com.genesyslab.composer.invalidkey	This is the event error for handling an invalid key name. For example, the userdata key being accessed by the Interaction Data block is not a valid key.
error.com.genesyslab.com.composer.notautorized	Not authorized.

error.com.genesyslab.com.composer.notfound	Not found.
error.com.genesyslab.composer.operationtimeout	This error occurs when the request for Userdata, AccessNumGet, or Statistics times out at the Iserver.
error.com.genesyslab.composer.receiveerror	This error occurs when the Iserver is down at the udata(getstat,AccessNumget) request.
error.com.genesyslab.composer.recordCapture.failure	Error occurred while saving the recorded file on the Web server side.
error.com.genesyslab.composer.servererror	This error occurs when there is an error in processing the request for the server side blocks such as "Invalid parameters" or "Web server is down." For Context Services blocks: Internal Server Error plus specific error message from the server.
error.com.genesyslab.composer.serviceunavailable	Service unavailable.
error.com.genesyslab.composer.webservice.batchfetch	For attribute descriptions, see the *SCXML Language Reference*.
error.com.genesyslab.customernotfound	For attribute descriptions, see the *SCXML Language Reference*.
error.com.genesyslab.composer.unsupported	This error occurs when the CTI applications designed for CTIC supported features like AccessNumget, Statistics, Interaction-Delete, Replace are called against a SIP Server environment.
error.com.genesyslab.subdialog.maxdepthexceeded	Error when subdialog depth limit is exceeded.
error.connection	This is the base exception for any connection-related error. For example, error.connection.protocol.nnn is thrown when the protocol stack for the connection raised an exception in the case of a bridged transfer. This error can be caught using this base exception.
error.connection.baddestination	The destination URI is malformed in the Transfer block.
error.connection.noauthorization	The caller is not allowed to call the destination, after initiating a transfer using the Transfer block.
error.connection.noresource	The platform could not allocate resources to place the call initiated by the Transfer block.
error.connection.noroute	The platform was not able to route the call to the destination, in a case where the destination URI (phone number) has the correct format.
error.dialog.collect	For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.continue	For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.createann	For attribute descriptions, see the *SCXML Language*

	Reference.
error.dialog.deleteann	For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.play	This indicates that an error occurred while trying to perform the <play> request. For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.playandcollect	This indicates that an error occurred while trying to perform the <playandcollect> request. For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.playandverify	For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.playsound	This indicates that an error occurred while trying to perform the <playsound> request. For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.remote	For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.setdialogdefaultdest	This indicates that an error occurred while trying to perform the <setdialogdefaultdest> request. For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.start	This indicates that an abnormal condition occurred while trying to perform the start request. This event will be sent as a result of a timeout of the request as well as problems with the request itself. For attribute descriptions, see the *SCXML Language Reference*.
error.dialog.stop	This indicates that an error occurred while trying to perform the <stop> request. For attribute descriptions, see the *SCXML Language Reference*.
error.interaction.redirect	For attribute descriptions, see the *SCXML Language Reference*.
error.msgbased.createmessage	This event is sent when the <createmessage> request has failed for some reason.
error.msgbased.sendmessage	This event is sent when the <sendmessage> request has failed for some reason.
error.noresource	The specified language is not supported by the TTS/ASR server, or the TTS/ASR server/service is down.
error.queue.cancel	For attribute descriptions, see the *SCXML Language Reference*.
error.queue.default	For attribute descriptions, see the *SCXML Language Reference*.
error.queue.query	For attribute descriptions, see the *SCXML Language*

	Reference.
error.queue.start	For attribute descriptions, see the SCXML Language Reference.
error.queue.stop	For attribute descriptions, see the SCXML Language Reference.
error.queue.submit	This indicates that an abnormal condition occurred while trying to perform the submit request. This event will be sent as a result of a timeout of the request as well as problems with the request or interaction itself. Busy treatment exceptions are raised as the error.queue.submit exception and not as exceptions listed in individual treatment blocks. See the routing application Target block for more information.
error.queue.update	For attribute descriptions, see the SCXML Language Reference.
error.semantic	.
error.session.fetch	An error occurred while trying to perform the fetch of a resource in a workflow.
error.session.start	For attribute descriptions, see the SCXML Language Reference.
error.session.terminate	For attribute descriptions, see the SCXML Language Reference.
error.unsupported	The platform encounters any error of type unsupported, such as error.unsupported.builtin.
error.unsupported.builtin	The built-in grammar is not supported by the platform or the ASR engine.
error.unsupported.format	An unsupported grammar format or audio format is encountered.
error.unsupported.language	The platform does not support the language for either speech synthesis or speech recognition.
error.unsupported.objectname	The requested object is not supported.
error.unsupported.receive	The application's access to messaging is disabled in MCP. This may be received as a VXML event for <receive> blocks.
error.unsupported.send	The application's access to messaging is disabled in MCP. This may be received as a VXML event for <send> blocks. Note: This will be applicable to all blocks that use <send>, such as the Interaction Data block.
error.unsupported.transfer	This is the base exception for any unsupported transfer settings.
error.unsupported.transfer.blind	The platform (configuration) does not support blind transfers, and the application specifies that it wants to do a blind transfer.
error.unsupported.transfer.consultation	The platform does not support consultative

	transfers, and the application specifies that it wants to do a consultative transfer.
error.unsupported.transrec.type	The platform does not support the media format.
error.unsupported.uri	The platform does not support the URI format (for example, fax://...).
exit	The caller has asked to exit. (Available when the universals property includes exit).
help	The caller has asked for help. (Available when the universals property includes help).
maxspeechtimeout	The caller input was too long, exceeding the property maxspeechtimeout.
noinput	The application expects voice or DTMF input, but it has received none from the caller within the timeout interval.
nomatch	The caller input something, but it was not recognized.

Exception Events for eServices UCS Blocks

The following eService blocks for Universal Contact Server use the exception events listed below:

- Identify Contact
- Update Contact
- Create Interaction

See the individual block topics for the list of supported exceptions.

Exception Event Name	**Error Code**	**Error Description**
error.com.genesyslab.composer.badfetch	400	Bad Request.
error.com.genesyslab.composer.notfound	404	Not Found.
error.com.genesyslab.composer.servererror	500	Internal Server Error + specific error message from Composer.
error.com.genesyslab.composer.notext	105	No text found in standard response : <UCS error message>.
error.com.genesyslab.composer.missingparameter	201	Missing parameter name.
error.com.genesyslab.composer.incompatibleparameter	202	Parameter ‘1’ and ‘2’ are not allowed.
error.com.genesyslab.composer.invalidparametertype	203	Incorrect type for parameter <parameter_name>, expected type 1 but was type 2.
error.com.genesyslab.composer.invalidparametervalue	204	Incorrect value for parameter <parameter_name>, expected value 1 but was value 2.

error.com.genesyslab.composer.invalidmessagetype	502	Invalid third party message type.
error.com.genesyslab.composer.objectnotfound	510	Object not found in database.
error.com.genesyslab.composer.incorrectsubtype	512	Incorrect subtype for interaction <interaction_name>, expected type 1 but was type 2.
error.com.genesyslab.composer.servererror	701	Unexpected error exception message.
error.com.genesyslab.composer.dberror	710	Connection to database failed.
error.com.genesyslab.composer.serveroverloaded	716	Server overloaded, request rejected.
error.com.genesyslab.composer.noattributes	730	No searchable attribute.
error.com.genesyslab.composer.invalidtenant	732	Invalid Tenant <tenant_name>.

Expression Builder

Use for both voice callflows and routing workflows to build expressions for branching and conditional routing decisions. You create expressions in Expression Builder; you can assign them to variables using the Assign block. You can also build ECMAScript expressions that use the Genesys Functional Modules in Expression Builder). Also see Skill Expression Builder used for routing applications.

Opening Expression Builder

Expression Builder opens from the various blocks, which include but are not limited to:

- Assign--Assign Data Property
- Branching--Conditions Property
- ECMAScript (for workflows)--Script Property
- Entry--Variables Property
- Log--Logging Details Property
- Looping--Exit Expression Property

Data Categories Accessed

Expression Builder gives access to the following categories of data, which can be used in expressions:

- Project Variables
- Workflow Variables (if accessed from a workflow)
- Callflow Variables (if accessed from a callflow)
- Workflow Functions (if accessed from a workflow)
- Callflow Functions (if accessed from a callflow)
- JavaScript (Array, Boolean, Date, Math, Number, String functions)
- Orchestration Server Functions
- Context Services
- Configuration Server

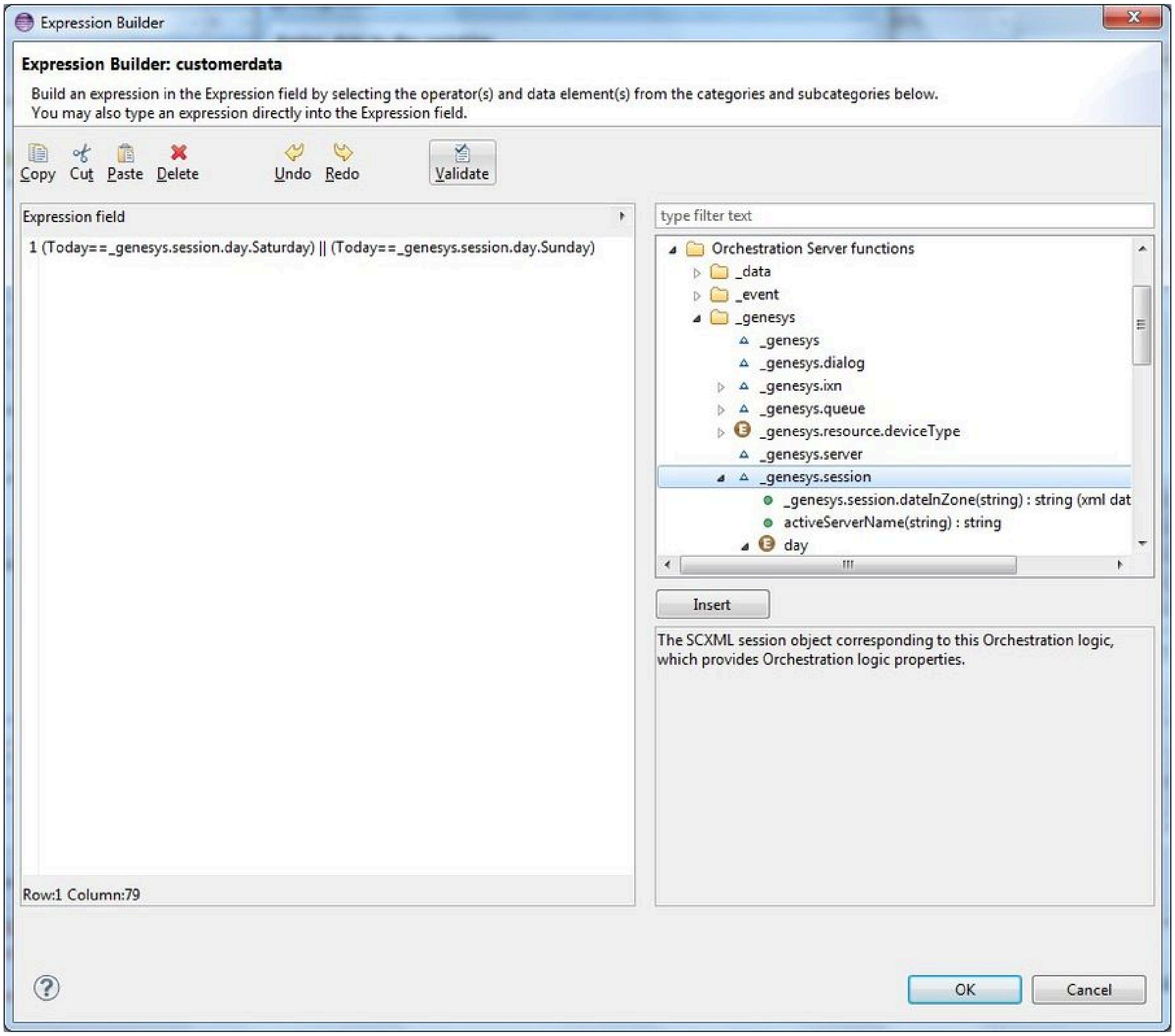
Note: Depending on the calling context (IPD, workflow editing, callflow editing), some of the above categories may be hidden.

Expression Builder Toolbars

This section discusses the editing toolbar the top of Expression Builder as well as the Operators toolbar.

Editing Toolbar

Expression Builder has an editing toolbar with buttons representing copy, cut, paste, delete, undo, redo, and validate . After you entering an expression and click the button to validate, syntax messages appear under the Expression Builder title. In the figure below, the syntax message is: No syntax error was found.



Expression Builder: customerdata

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

```
1 (Today==_genesys.session.day.Saturday) || (Today==_genesys.session.day.Sunday)
```

Row:1 Column:79

The SCXML session object corresponding to this Orchestration logic, which provides Orchestration logic properties.

Operators

Expressions can consist of comparisons joined by AND (&) and OR (|), which have the same priority. URS uses integer arithmetic in its calculations, such as for expression evaluation. For this reason, you must always create expressions based on integer arithmetic, not float. When an expression contains more than one logical comparison, the logical operation to the left has precedence over the operation to the right. Use parentheses to overrule this precedence. When the order of logical operations is not explicitly specified by parentheses, the operation to the left has precedence over the operation to the right. For example: `Portuguese > 5 | Africa = 7 & SpTours >3` is equivalent to `(Portuguese > 5 | Africa = 7) & SpTours > 3` Each comparison consists of two data values that are compared against each other. The table below shows the operators used in logical expressions.

Symbol	**Meaning**	**Example**
==	use for comparison (e.g. x==y to compare is x equal to y)	see example screen shot
=	equal to done for assignment (e.g. x=y to assign x equal to y)	Day = Monday
!=	not equal to	Day != Sunday
&	Single & is an AND operators which manipulates internal bits of integers	
&&	&& and || are used for comparisons to build boolean (true/false) expressions	
||	see &&	see example screen shot
>	greater than	Time > 9:00
>=	greater than or equal to	Day >= Monday
<	less than	Time < 16:00
<=	less than or equal to	Day <= Friday

Mathematical Symbols Allowed

The allowed mathematical symbols in expressions are restricted to those symbols for addition, subtraction, division, and multiplication (+, -, /, *). These symbols work with numeric values only, so any participating String argument is automatically converted to a number.

Examples of Expressions

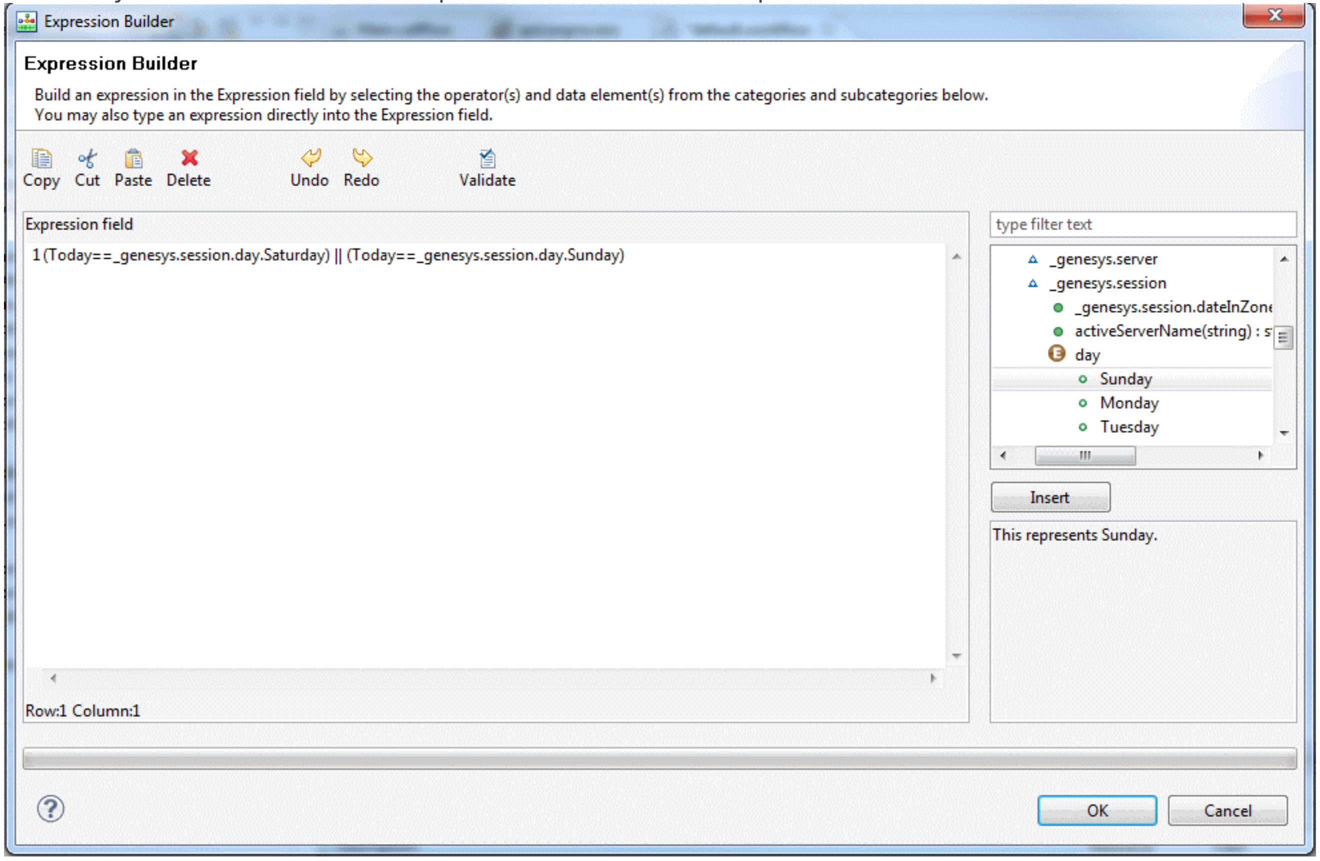
The table below shows example expressions.

Example	**Interpretation**
Day=Saturday|Day=Sunday	If the day is Saturday or Sunday
Time<=5:00 |Time>=18:00	If the time is less than or equal to 5:00 (5:00 AM) or

	greater than or equal to 18:00 (6:00 PM); in other words, if the time is between 6:00 PM and 5:00 AM.
Day>=Monday&Day<=Friday & ANI!=8004154732	If the day is a weekday and the ANI is not 8004154732
SData[1123@SF.A,StatTimeInReadyState]>120	If agent 1123 has been in Ready state for more than 120 seconds
UData[Acct]>=9000	If the value associated with the key Acct in the Interaction data structure is equal to or greater than 9000

Creating Expressions

Assume you wish to create the expression shown in the Expression field below.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

Expression field

```
1 (Today==_genesys.session.day.Saturday) || (Today==_genesys.session.day.Sunday)
```

Row:1 Column:1

This represents Sunday.

In the above figure, the expression `(Today ==_genesys.session.day.Saturday) || (Today == _genesys.session.day.Sunday)` was created as follows:

- In the left text field under the toolbar, type an open parenthesis (().
- If you already defined the "Today" variable, select it under **Workflow variables** or **Project variables**. Otherwise, if not yet defined, type "Today" (no quotes) in the text after the open parenthesis.
- Type "==" (no quotes) in the text.

- In the right text box, expand:

`Orchestration Server Functions _genesys genesys.session.day`

- Double-click **Saturday**.
- In the left text field, type a close parenthesis ())
- Click the **|| Operators** button. Repeat the above steps except, at the end before the close parenthesis, select _genesys.session.day.Sunday.

Note: You may also use the search field on the right side to filter items that include **Saturday**

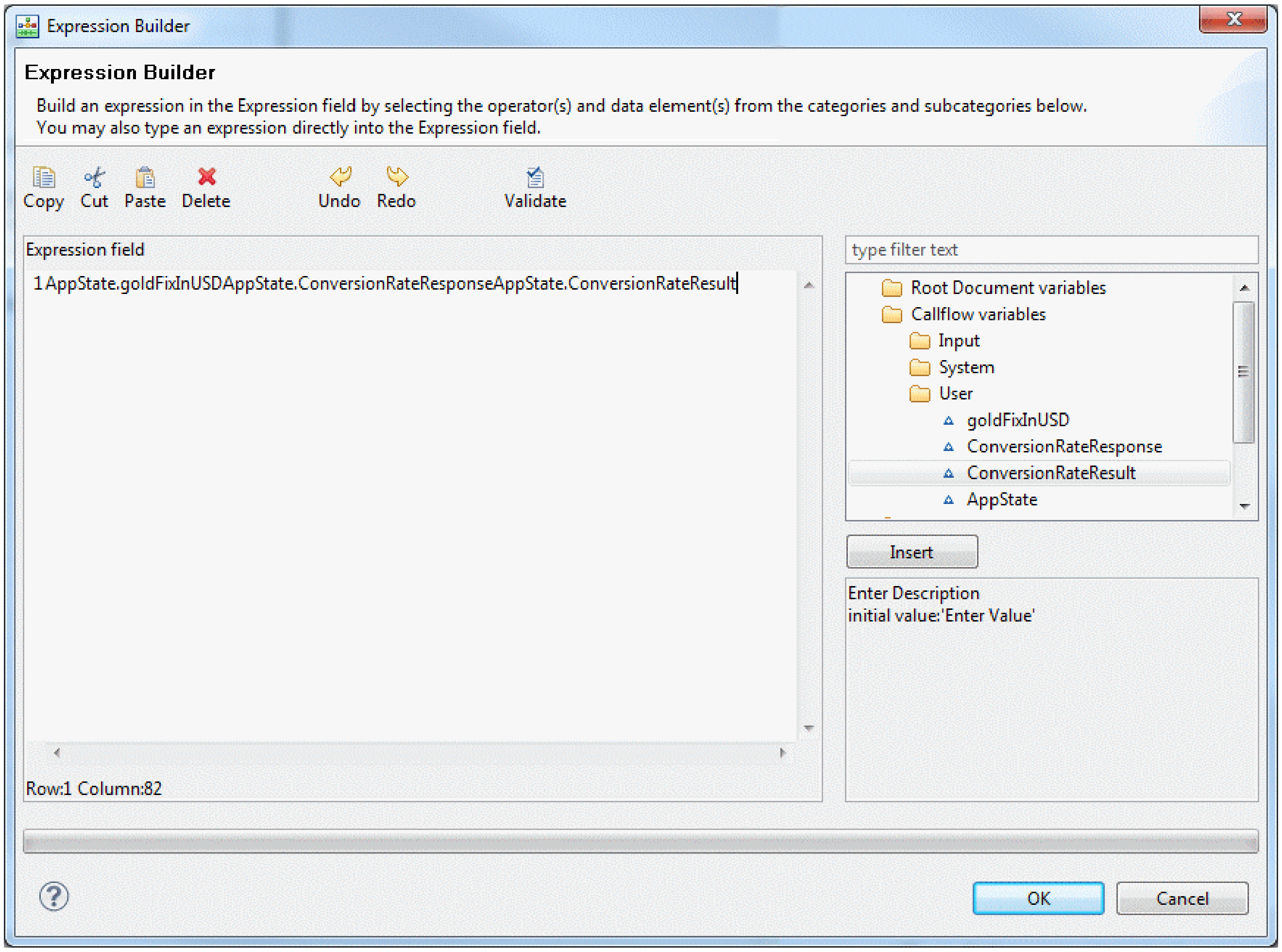
Usage of Variables in Expressions

Note: The steps for using variables in Expression Builder varies slightly depends on whether you are creating a voice callflow or a routing workflow.

- If creating a voice callflow, the right selection area contains Callflow Variables.
- If creating a routing workflow, the selection area contains Workflow Variables.

Using Variables in a Callflow

Assume you wish to use variables in a callflow to create the following: `AppState.goldFixInUSDAppState.ConversionRateResponseAppState.ConversionRateResult` The figure below shows the entry in Expression Builder.



```
1 AppState.goldFixInUSDAppState.ConversionRateResponseAppState.ConversionRateResult
```

Row:1 Column:82

Enter Description
initial value:'Enter Value'

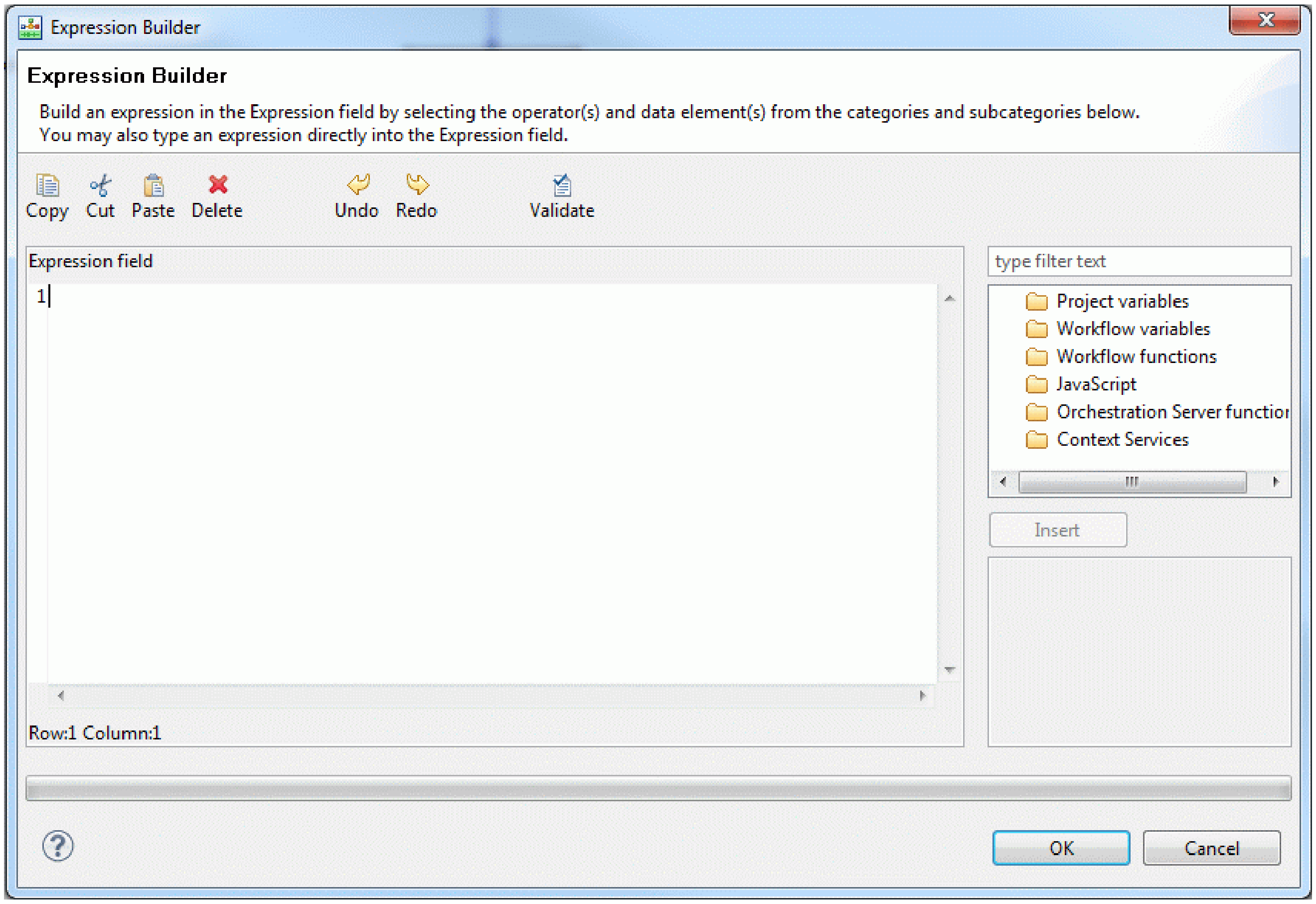
Assume you have already defined these variables in the Entry object. The above entry could then be created as follows:

1. The right selection area lists **Callflow Variables**, **Java Script**, and **Context Services**.
2. Expand **Callflow Variables**.
3. Expand **User** to view variables defined in the Entry block.
4. To place the variables in the Expression field at the top of Expression Builder, double-click **goldFixInUSD**,**ConversionRateResponse**, and **ConversionRateResult**. The AppState VoiceXML Data Model Object will be appended automatically to variables used inside Expression Builder. The code inside the Expression field will be directly substituted within the generated VoiceXML code.
5. Continue creating the expression.

Expression Builder Data Categories

Expression Builder accesses various categories of data Including Orchestration Server and Universal

Routing Server functions. The folders shown on the right depend on whether you are working with a callflow or workflow. The figure below shows Expression Builder Data Categories when working with a workflow.


Expression Builder
Expression Builder
Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.
Copy
Cut
Paste
Delete
Undo
Redo
Validate
Expression field
1
Row:1 Column:1
type filter text
Project variables
Workflow variables
Workflow functions
JavaScript
Orchestration Server functior
Context Services
Insert
OK
Cancel

Project variables

Use Project variables when you need to share information across different workflows.

Workflow variables

Use Workflow variables when you need to share information across different blocks in the same workflow.

Workflow functions

Use the Workflow functions category when creating routing workflows. Selecting a function displays a description. See the Orchestration Server wiki for information on functions available for use in Composer when building routing workflows. **Notes**: Functions getCallType and getIxnMediaType can be used to identify the call type and/or media type for the purpose of segmenting incoming interactions.

- `getCallType(ixnID)`--This function gets the call type `_genesys.ixn.interactions[].voice.type` for the specified interaction. If the ixnID is not specified, it will return the calltype for the current interaction.
- `getIxnMediaType(ixnID)`--This function gets the correct media type ENUM from `_genesys.FMName.MediaType` for the specified ixnID. If ixnID is not specified, the current interaction id will be used. If the interaction's media type cannot be determined or the specified ixnID does not exist, the function will return undefined.

Use Case: A call arrives on a routing pointing, initiating a routing workflow. The workflow checks for the call type. If the call type is outbound, then the call is immediately moved to the front of the queue and routed to an available agent. If the call type is inbound, the call is assigned a priority and routed based on the desired service level. For information on the other functions, see eServices Blocks.

Callflow functions

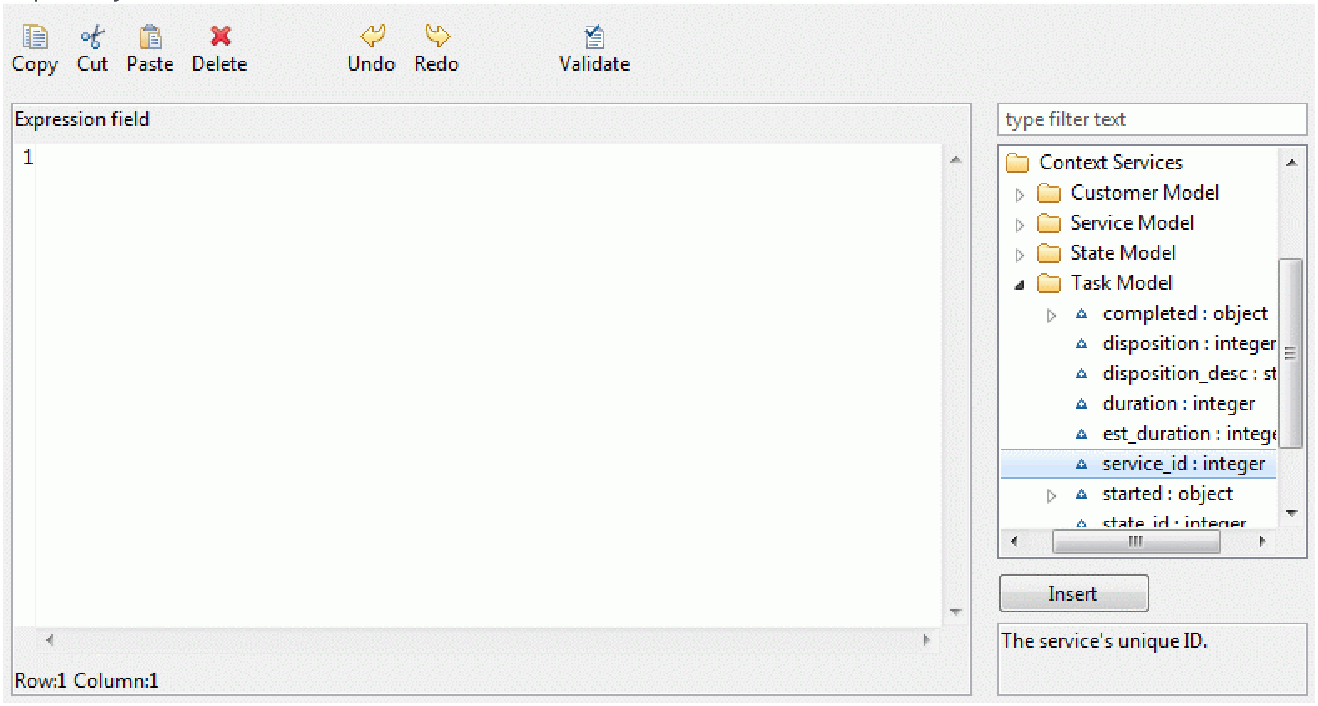
Note: Function `getSIPHeaderValue(headername)` returns the SIP header value associated with the given SIP headername. You may wish to use this function with the Assign block.

JavaScript

Use JavaScript to access those functions categorized as follows: Array, Boolean, Date, Math, Number, String.

Context Services

Use Context Services when creating expressions that use attributes associated with this optional capability of Universal Contact Server Database.



- Context Services
  - Customer Model
  - Service Model
  - State Model
  - Task Model
    - completed : object
    - disposition : integer
    - disposition_desc : st
    - duration : integer
    - est_duration : intege
    - service_id : integer
    - started : object
    - state_id : integer

The service's unique ID.

Configuration Server

This category is displayed for workflows when the Expression Builder is called from the ECMAScript and Branching blocks. If connected to Configuration Server, Composer can fetch standard responses and category codes.

Standard Response Library

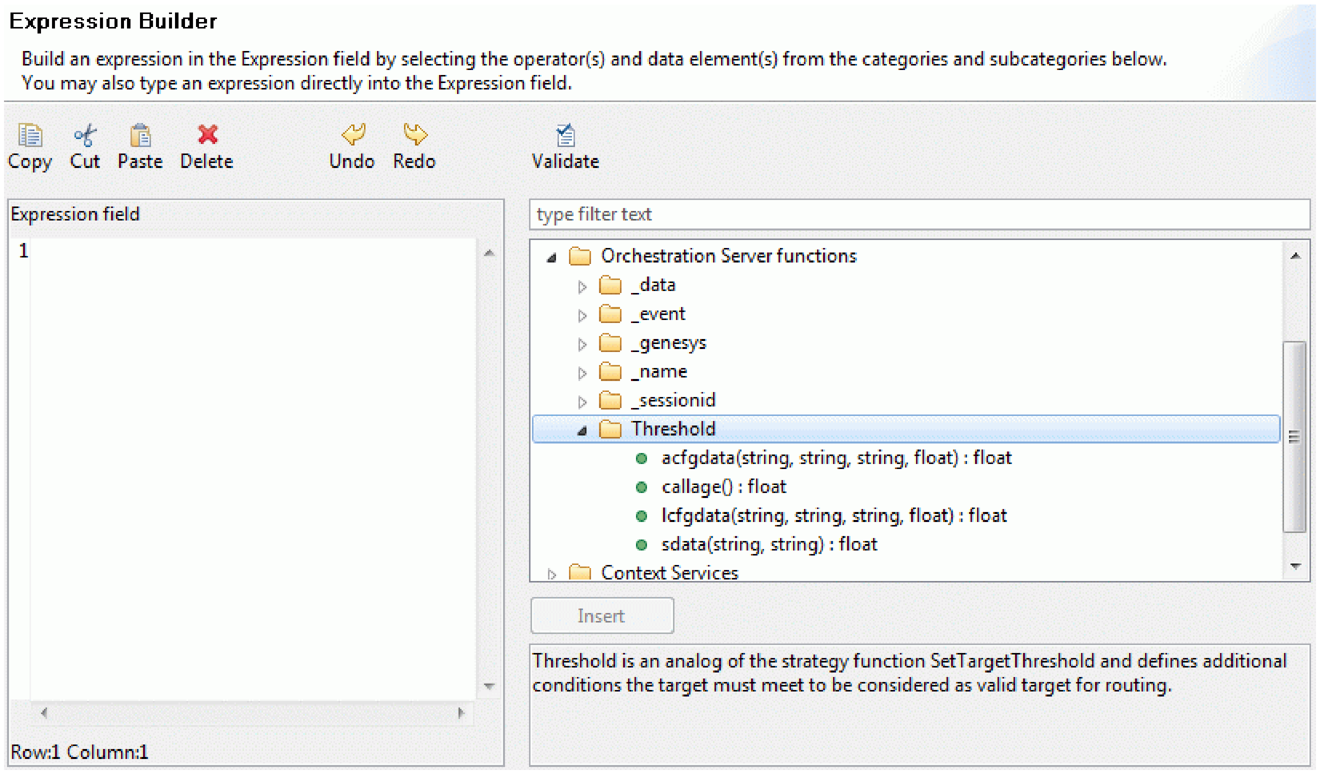
This category allows you to access pre-written responses for customers that have been defined in Knowledge Manager (as described in the *eServices 8.1 User's Guide*).

Orchestration Server and URS Functions

Composer's Expression Builder provides access to many Orchestration Server and Universal Routing Server (URS) functions. For more information see Using URS Functions.

Threshold Functions

Universal Routing Server's threshold functions can be used for conditional routing. For example, the threshold functions can be used in the Target block for a type of conditional routing called "share agents by service level agreement routing." This type of routing enables a business user that manages multiple business lines to define the triggering conditions and constraints that allow agents to be shared among business lines. By constructing a single threshold expression, you can define the triggering conditions for borrowing agents from other business lines as well as the conditions that apply to the lending business line.
Threshold is an analog of the URS strategy function `SetTargetThreshold` as defined in *Universal Routing 8.1 Routing Application Configuration Guide*. It defines additional conditions the target must meet to be considered as valid target for routing. The threshold functions are available in Composer's Expression Builder:

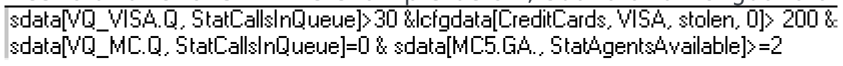

Expression Builder

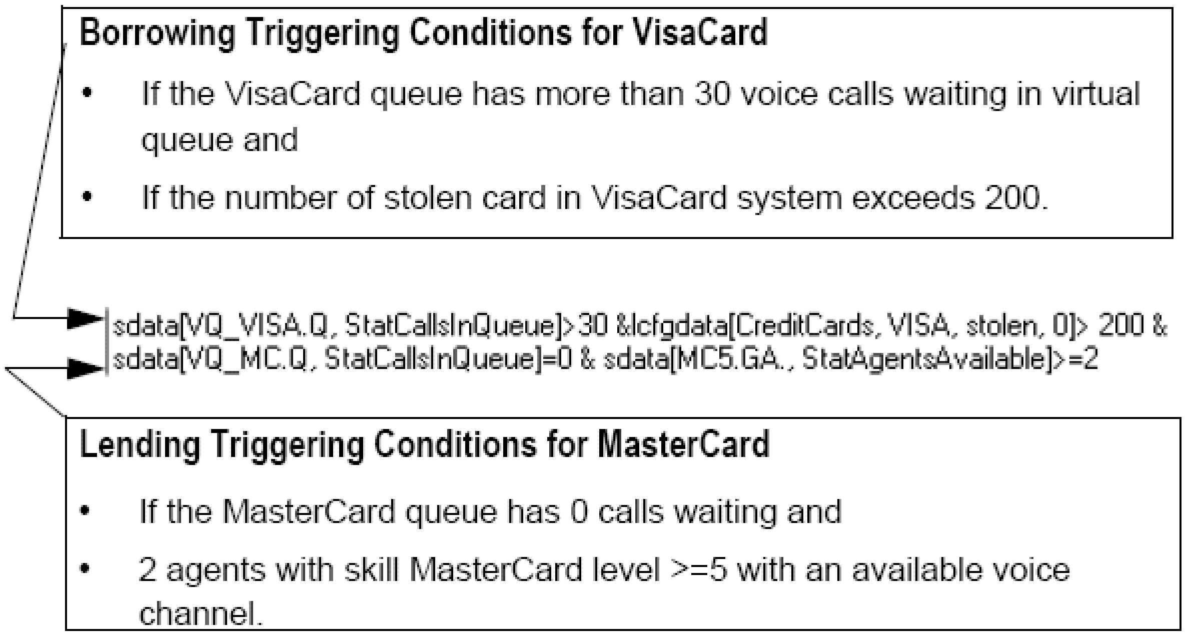
Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

Threshold is an analog of the strategy function SetTargetThreshold and defines additional conditions the target must meet to be considered as valid target for routing.

- `acfgdata(Application name, folder, property, default value)`. Use this function to affect routing conditions based on external data stored in properties of Configuration Database Application objects (ApplicationConFigDATA). Returns a numeric value for a specified Application option. If an Application has no such option then the default value is returned. Return value type: FLOAT. Example: sdata(Group2.GA, StatAgentsAvailable)>acfdata(URS, default, MinNumOfRdyAgents, 2)
- `callage`. Use this function to return the age of an interaction in seconds. Use for time-based routing conditions, such as a call that can only be routed if it waits more then 60 seconds. Return value type: FLOAT.
- `lcfgdata(list name, item, attribute, default value)`. Use this function to affect routing conditions based on external data stored in List objects. Returns a numeric value for a specified attribute of a List object's item If a List object has no such item or attribute, the default value is returned. Works like acfgdata, but uses a List object (`ListConFigDATA`) instead of an Application. Return value type: FLOAT.
- `sdata(target, statistic)`. Use this function to affect routing conditions based on statistics. Specify targets and statistics just like for the `SData[]` function described in the *Universal Routing 8.1 Reference Manual*. You can use the URS predefined statistics (see Statistics Manager and Builder), such as: PositionInQueue, CallsWaiting, and InVQWaitTime. Return value type: FLOAT. Example: sdata(Group2.GA, StatAgentsAvailable)>2

Example Threshold Expression

A threshold expression is text string very similar to the regular expressions used for branching, but uses the threshold functions. In the example below, `sdata` and `lcfgdata` are the predefined threshold functions. 
sdata[VQ_VISA.Q, StatCallsInQueue]>30 &lcfgdata[CreditCards, VISA, stolen, 0]> 200 &
sdata[VQ_MC.Q, StatCallsInQueue]=0 & sdata[MC5.GA., StatAgentsAvailable]>=2
In this example, both the borrowing and lending conditions are defined in a single threshold expression:



Borrowing Triggering Conditions for VisaCard

- If the VisaCard queue has more than 30 voice calls waiting in virtual queue and
- If the number of stolen card in VisaCard system exceeds 200.

sdata[VQ_VISA.Q, StatCallsInQueue]>30 &lcfgdata[CreditCards, VISA, stolen, 0]> 200 &
sdata[VQ_MC.Q, StatCallsInQueue]=0 & sdata[MC5.GA., StatAgentsAvailable]>=2

Lending Triggering Conditions for MasterCard

- If the MasterCard queue has 0 calls waiting and
- 2 agents with skill MasterCard level >=5 with an available voice channel.

For detailed information on using the threshold functions, see *Universal Routing 8.1 Routing Application Configuration Guide* and threshold attribute in Orchestration Developers Guide.

GetMediaTypeName Function

This function returns the name of media type associated with interaction as defined in the Configuration Database. Located in Expression Builder as follows: `_genesys.ixn.mediaType` Expression Builder lists the Media type enumerators supported by the URS platform. The main difference between this function and IRD's `getMediaTypeName` function is that

- The IRD function takes a parameter for (the current interaction media type) and returns a String name of the media type

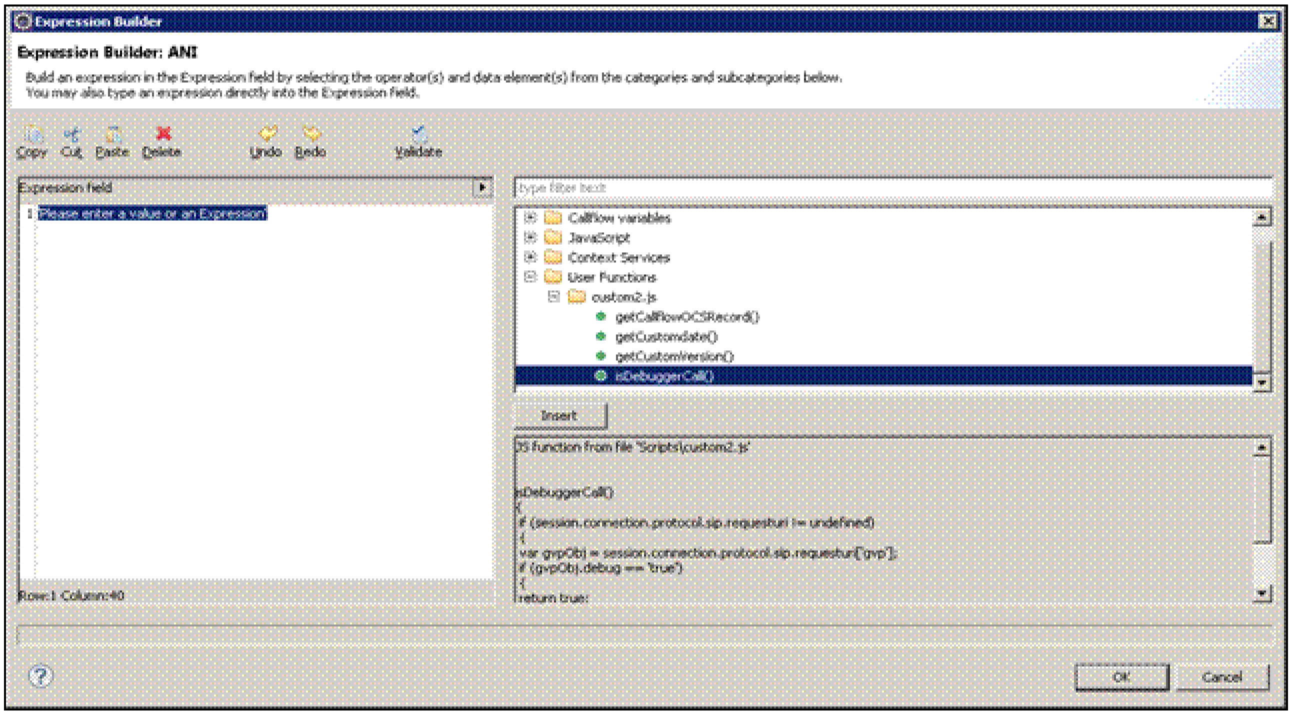
Whereas:

- Composer's function `getIxnMediaType(ixnID)` takes a parameter of any interaction and returns a ENUM type `_genesys.FMName.MediaType` of the media type. Returns the media type of the given interaction (ixnID), otherwise the current interaction.

User defined JavaScript Functions

Expression Builder shows the user defined JS methods under 'User Functions' category to list user defined JS methods for easy access.

- For Callflow diargams, JavaScript files added in the Scripts property of the Entry BlockScripts are considered.
- For Workflow diargams, JavaScript files added inside the include/user folder are considered.



Expression Builder: ANI

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

Expression field

1 Please enter a value or an Expression

Row:1 Column:40

type filter text

- Callflow variables
- JavaScript
- Context Services
- User Functions
  - custom2.js
    - getCallflowOCSRecord()
    - getCustomdate()
    - getCustomVersion()
    - isDebuggerCall()

```
JS function from file 'Scripts\custom2.js'


isDebuggerCall()
{
if (session.connection.protocol.sip.requesturi != undefined)
{
var gvpObj = session.connection.protocol.sip.requesturi['gvp'];
if (gvpObj.debug == 'true')
{
return true;
```

GAX Server Preferences

See GAX Server Preferences for voice applications.

Getting Using Email Addresses

This topic describes several ways to get the e-mail address of a customer using Context Services blocks and using it in the To property of the Create Email block.

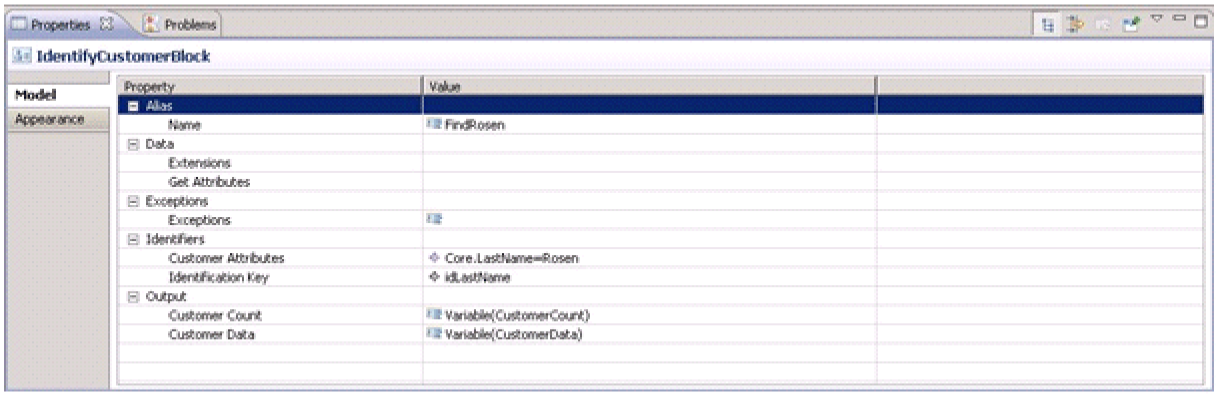
Getting a Customer Email Address

There are two ways to get a customer's e-mail address from the Context Services server:

- Use the Query Customer block. If the Customer ID is unknown, first follow steps 1 and 2 below. Then, follow the steps 3 and 4.
- Use the Identify Customer block. Follow step 1 and set the property Get Attributes to Yes. Then follow step 4 and change the ECMAScript expression to `_data.EmailAddress= _data.CustomerData[0].EmailAddress`

Identifying a Customer

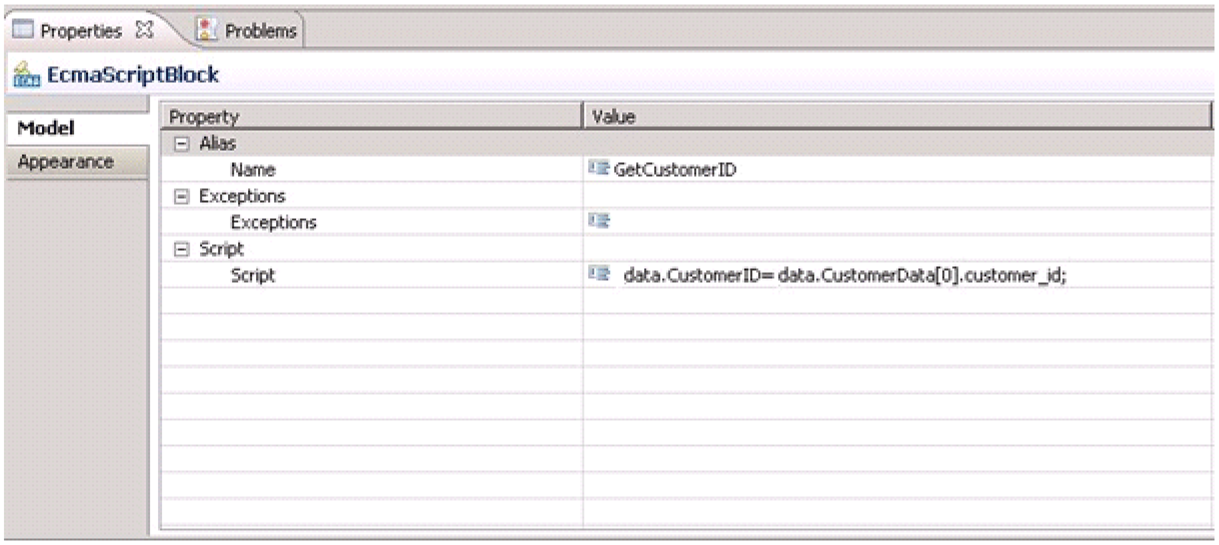
1. **Identifying a Customer**. A customer ID is necessary to use the Query Customer block. If the customer ID is unknown, you may use the Identify Customer block to get the Customer ID. In the example below, we are trying to identify a customer by the last name. The result of the request is stored in the CustomerData variable.



IdentifyCustomerBlock

Property	Value	
⊟ Alias		
Name	FindRosen	
⊟ Data		
Extensions		
Get Attributes		
⊟ Exceptions		
Exceptions		
⊟ Identifiers		
Customer Attributes	Core.LastName=Rosen	
Identification Key	idLastName	
⊟ Output		
Customer Count	Variable(CustomerCount)	
Customer Data	Variable(CustomerData)	

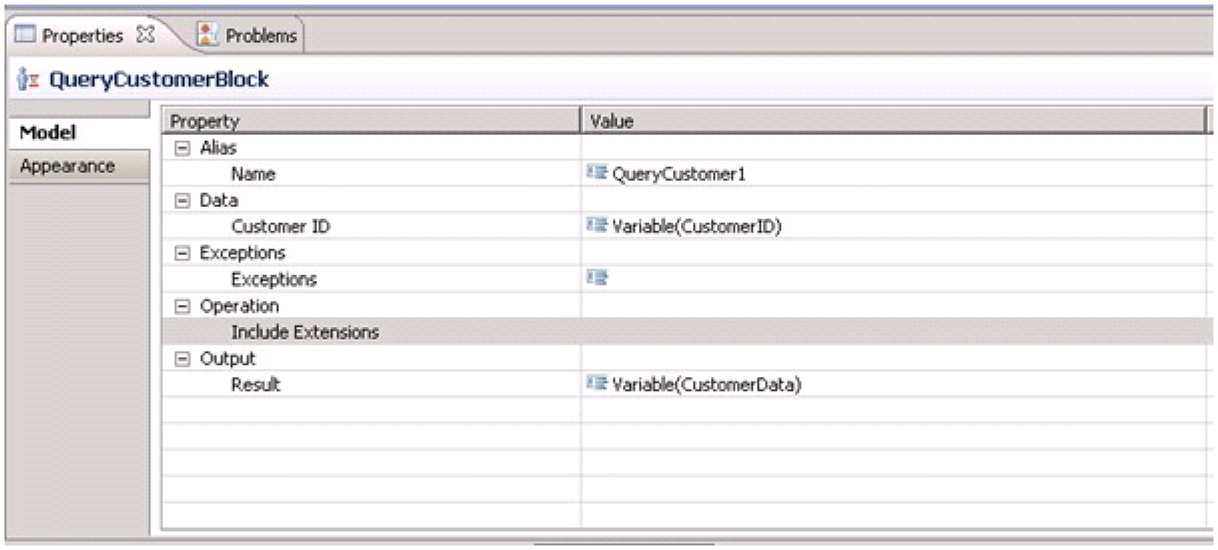
2. **Getting the Customer ID**. An ECMAScript block is used to extract the Customer ID after identifying the customer. The variable CustomerID is used.



EcmaScriptBlock

Property	Value
⊟ Alias	
Name	GetCustomerID
⊟ Exceptions	
Exceptions	
⊟ Script	
Script	data.CustomerID= data.CustomerData[0].customer_id;

3. **Querying the customer profile**.The CustomerID variable is used in a Query Customer block. The result (JSON) is put in the CustomerData variable.



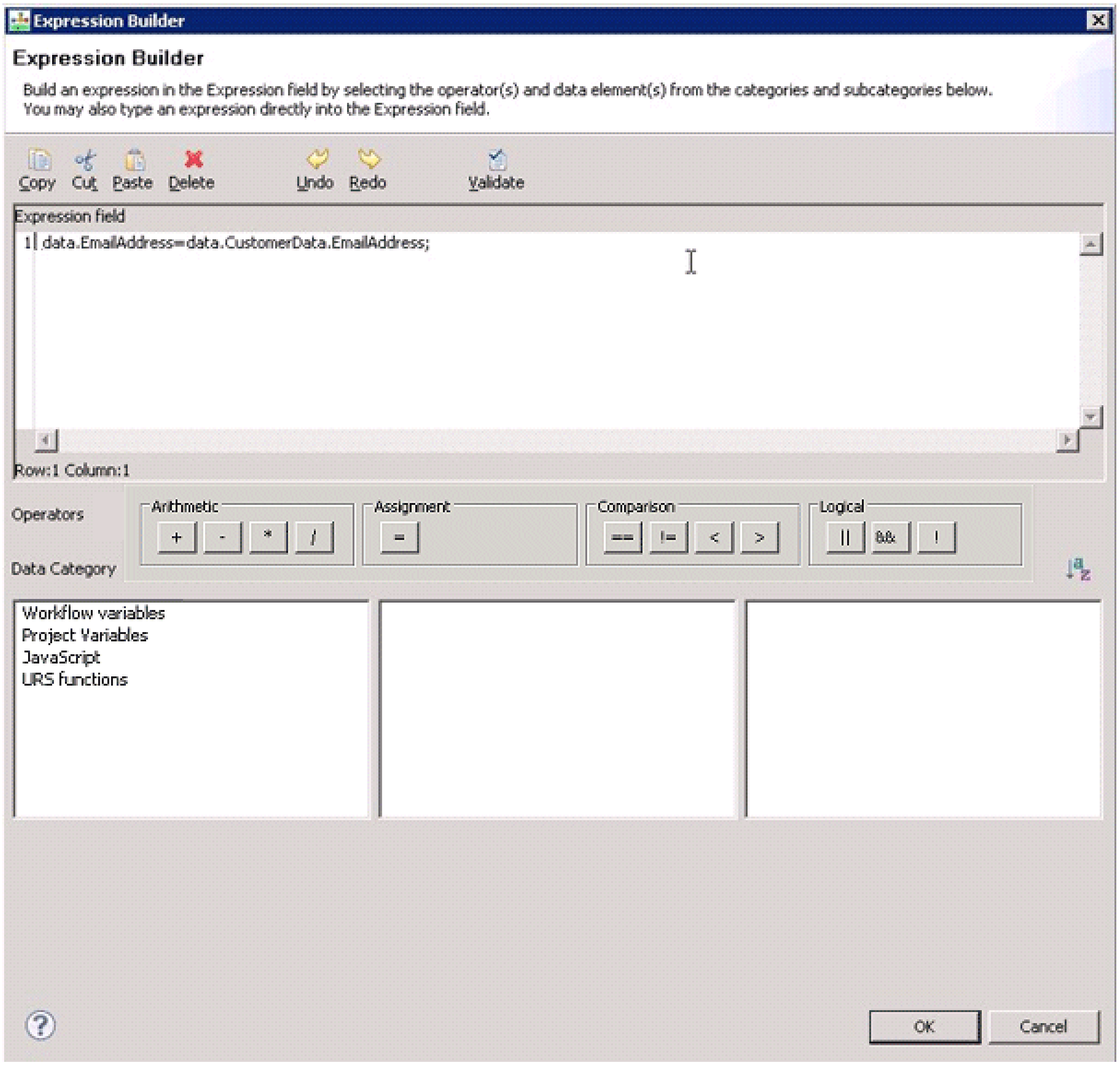
QueryCustomerBlock

Property	Value
⊟ Alias	
Name	QueryCustomer1
⊟ Data	
Customer ID	Variable(CustomerID)
⊟ Exceptions	
Exceptions	
⊟ Operation	
Include Extensions	
⊟ Output	
Result	Variable(CustomerData)

With the default Context Services schema, EmailAddress is a field of the customer profile. The CustomerData content sample:

```
{"EmailAddress":"Roger.Rosen@genesyslab.com","FirstName":"Roger","PhoneNumber":["+1.219.12345678
Rosen ","customer_id":"00001a5GD0A80040"}
```

4. **Extracting the e-mail address field**. You can then use a simple expression in an ECMAScript block to extract the e-mail address. The example below illustrates how to use Expression Builder to extract the e-mail address from the Core customer profile when using the default Context Services schema.



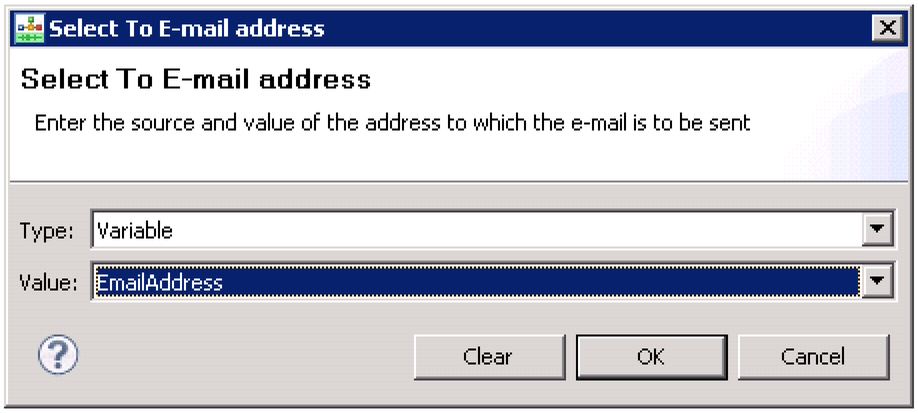
Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below. You may also type an expression directly into the Expression field.

```
data.EmailAddress=data.CustomerData.EmailAddress;
```

Row:1 Column:1

Using the E-mail Address in the To Field

In the Create Email block, when defining the **To** property, select the **Type** as Variable and select the variable that was previously used to save the e-mail address (`data.EmailAddress`) from above.


Select To E-mail address

Select To E-mail address

Enter the source and value of the address to which the e-mail is to be sent

Type: Variable

Value: EmailAddress

Using Variables Mapping for E-mail Addresses

With the Variables Mapping feature, getting an e-mail address is easy. When Customer ID is unknown:

- Set the Identify Customer/**Get Attributes** property to "Yes"
- Set the Identify Customer/**Variables Mapping** property to map the variable "EmailAddress" to the Context Services attribute "EmailAddress"

When Customer ID is known:

- Set the Query Customer/Variables Mapping property to map the variable "EmailAddress" to the Context Services attribute "EmailAddress"

No additional ECMAScript block is needed. The "EmailAddress" variable will automatically be assigned the customer's email address.

Import and Export

Composer provides several import options as described in this topic.

Importing External Files into Your Composer Project

If you have files, such as pre-recorded audio prompts for your application, that you would like to add to a Composer Project, you can add the files to the appropriate folder (such as the Resources\Prompts\en-US folder for audio prompt files) by dragging and dropping the files from Windows Explorer. Or, you can use the Composer Import function as follows:

1. Select File > Import, expand the General folder, then select File System and click Next.
2. In the Import File System dialog box, click the From directory Browse button to navigate to the folder that contains your audio prompt files.
3. Select the files and folders to import.
4. Click the Into folder Browse button and navigate to the location within an existing Composer Project (preferably within the Resources folder) where the files will be stored.
5. If you know you want to overwrite the resources in the selected folder, select the Overwrite existing resources without warning check box.
6. Select either Create complete folder structure or Create selected folders only as your situation dictates.
7. Click Finish to launch the import.

Use the Resource Type in the Prompt Settings dialog box of your Prompt block to add an audio file into the actual prompts of your application.

Importing Composer Projects into Your Workspace

Eclipse provides import and export options for copying Projects. Use the Import Wizard to copy the Composer Project from any other location into your workspace.

1. Select File > Import, expand the General folder, then select Existing Projects into Workspace and click Next.
2. Specify the path of the location from where you want to import the Project.
3. All Projects in the workspace will be shown in the Projects list. Select the ones that you wish to copy. Select the check box at the bottom to copy the files into your current workspace.
4. Click Finish to import the Composer Project.

Importing a CallFlow or Workflow Diagram Template

Diagrams saved as templates can be imported from the file system. See Diagram Templates for more information.

Exporting Application Prompts

You can export all the prompts from your Composer Project as XML or CSV files. This is very useful when sending the voice scripts to a Recording Studio where audio files would be recorded. Developers can also make good use of this feature for doing a quick review and sanity check of all application prompts.

1. Select File > Export, expand the Composer folder, then select Export Prompts Listing and click Next.
2. Select the Composer Project whose prompts you wish to export, the format for the export (XML or CSV), and the destination location. A file with the same name as the name of your Composer Project will be created in the destination location. The file extension will depend upon the format of the export selected.
3. Click Finish to begin the export.

Exporting a Composer Project as a WAR File

Exporting a Composer Project as a .war file is the first required step to deploy the voice application on your Production Web Application Server. Therefore, this subject is covered under Deploying Voice Applications. Note: When you export a Project to a .war file, SQL files are not generated from DB Data blocks. Generate code once before exporting a project. Note: The following folders are not needed after export:

- simulation
- Callflows
- Debugging-results

Exporting Composer Projects to an Archive File

To export to an archive:

1. Select File > Export, expand the General folder, then select Archive File and click Next.
2. Select the Composer Project(s) to save as an archive.
3. Navigate to a destination for the archive file.
4. Under Options, save as a .zip or .tar file, create a directory structure or only selected directories, and decide whether to compress the contents of the archive file.

5. Click Finish to begin the export.

Note: The following folders are not needed after export:

- simulation
- Callflows
- Debugging-results

Diagrams saved as templates can exported to another user's Composer. See Exporting a Diagram Template to the File System for more information. When a Project is exported, Composer creates an entry in its logs. You can find the log file in: <workspace folder>\.metadata\.log This location is fixed. You cannot move the log file to another location.

Application Reporting

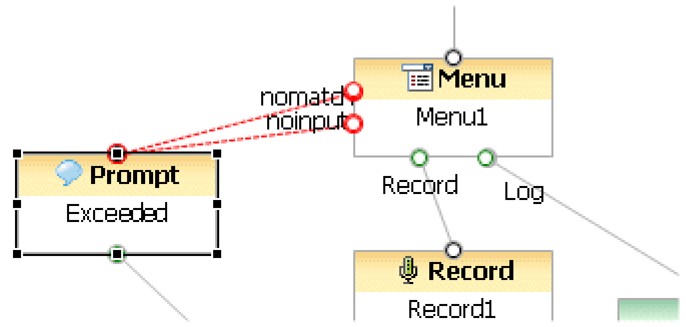
Composer applications can generate reports that can be displayed in the Reporting Server. You need the Reporting Server to be installed in your GVP setup to get application reports. The Reporting blocks can be used to send IVR-related reports from Composer-generated applications. To enable application reporting, use the EnableReports variable. Use the VAR Call Browser in Genesys Administrator to access the reported data that is generated by the Reporting blocks when EnableReports is set to true. Note: Refer to Reporting Server and Genesys Administrator documentation for more details on Voice Application Reporting concepts.

Importing and Exporting Diagram Templates

Diagrams saved as templates can exported to/imported from the file system. See Using Diagrams as Templates for more information.

Link Tool

The figure below shows examples of using the link tools:



In the above example, the red links (going from the Menu block to the Prompt block) result from using the Exception Link tool to connect the two blocks. The black links (going from the Menu block to the Record block and the Log link) result from using the Outlink tool.

Link Tools Detail

The table below contains detailed information on the tools used for connecting blocks.

OutLink Use the Outlink tool to connect blocks by selecting the tool and dragging from the source block to a target block. You can also connect blocks without using the Outlink tool. Simply start the drag operation from the source block’s outport and drop into the target block. In both cases, the link can be dropped on top of the target block or to the connection port of the target block.

ExceptionLink Use the Exception link tool for exception handling. When you define exceptions for a block in the dialog box that opens from the Properties tab, this creates the same number of exception ports on the block. Use to draw connections from an Blocks and Ports exception port to a another block.

Locales

A *locale* defines a language and region identifier that you want to work with. Composer lets you define default, active, and custom locales during Project creation or through the Project properties. The locales defined in each Composer project can be used in callflow diagrams, workflow diagrams, and Grammar Builder. When using Grammer Builder, you specify locales, which are the languages that a grammar file will support. The Grammar builder wizard uses the active locales for the Composer Project. Callflow diagrams use the default locale selection for the following Entry block variables:

- APP_LANGUAGE
- ASR_LANGUAGE

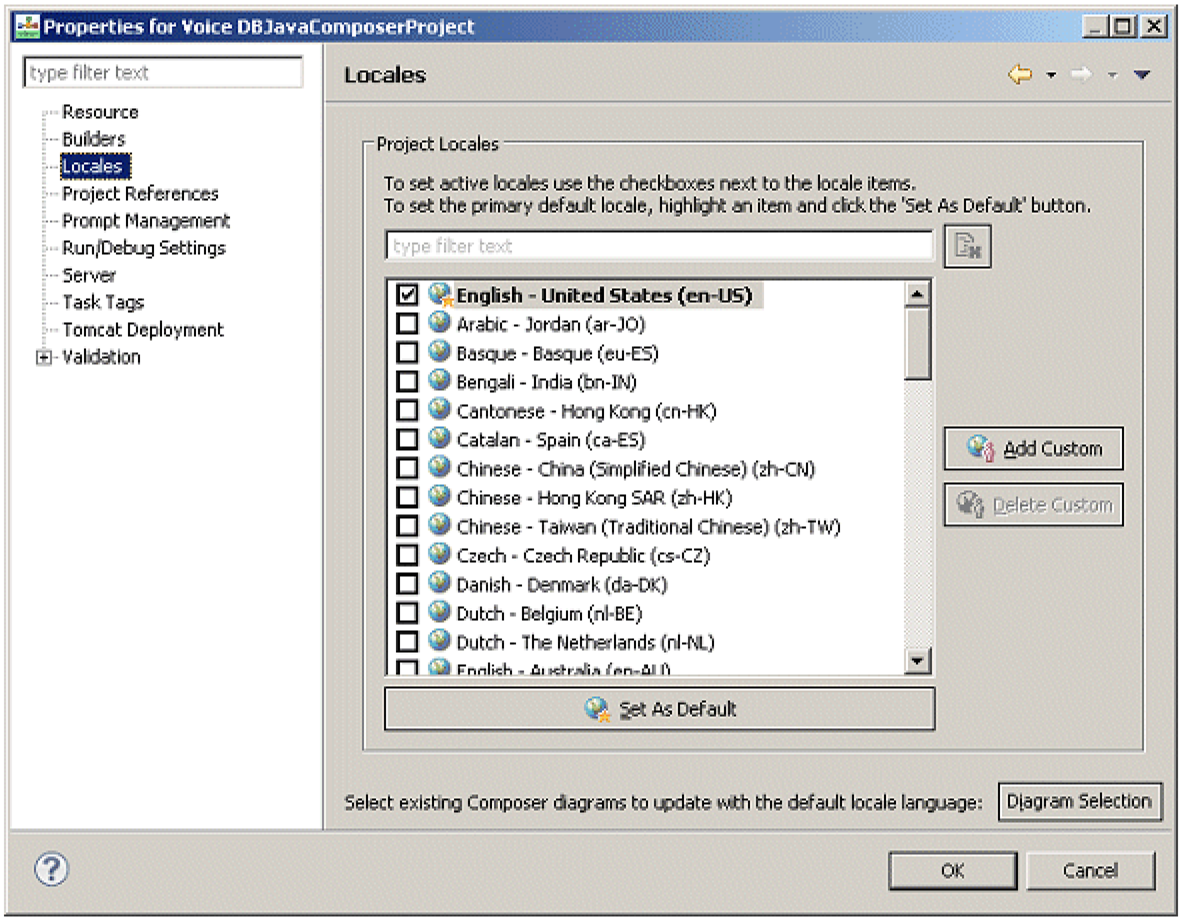
Workflow diagrams use the default locale selection for the following Entry block variable:

- system.Language

Changing Locales at the Project Level

The Project locales may be changed at the Project Properties at any time.

1. In the Project Explorer, right-click the Composer Project and select **Properties**.
2. Select the **Locales** section.
3. Enable the check boxes to set active locales for the Project.
4. To set the default locale, select the row and click **Set as Default**.



To set active locales use the checkboxes next to the locale items.
To set the primary default locale, highlight an item and click the 'Set As Default' button.

- [✓] English - United States (en-US)
- [ ] Arabic - Jordan (ar-JO)
- [ ] Basque - Basque (eu-ES)
- [ ] Bengali - India (bn-IN)
- [ ] Cantonese - Hong Kong (cn-HK)
- [ ] Catalan - Spain (ca-ES)
- [ ] Chinese - China (Simplified Chinese) (zh-CN)
- [ ] Chinese - Hong Kong SAR (zh-HK)
- [ ] Chinese - Taiwan (Traditional Chinese) (zh-TW)
- [ ] Czech - Czech Republic (cs-CZ)
- [ ] Danish - Denmark (da-DK)
- [ ] Dutch - Belgium (nl-BE)
- [ ] Dutch - The Netherlands (nl-NL)
- [ ] English - Australia (en-AU)

Select existing Composer diagrams to update with the default locale language:

Changing the Default Locale

To set the default locale, highlight the row in the Project Locales list and click **Set as Default.** Any new callflows or workflows will inherit this new default locale. You may optionally update existing callflow and workflow Entry block language variables to the desired default language. The callflow/workflow language variables (APP_LANGUAGE and ASR_APP_LANGUAGE/system.Language) will be updated.

1. After selecting the default locale language, select the box **Select existing Composer diagrams with the default locale language**.
2. Click **OK**. The Diagram Locale Update dialog box opens.

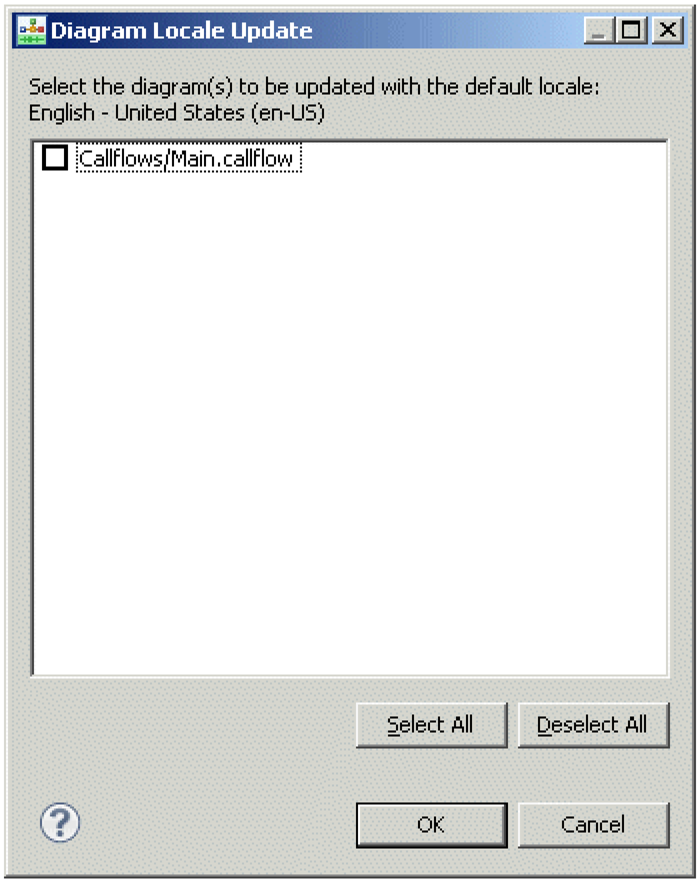


Diagram Locale Update

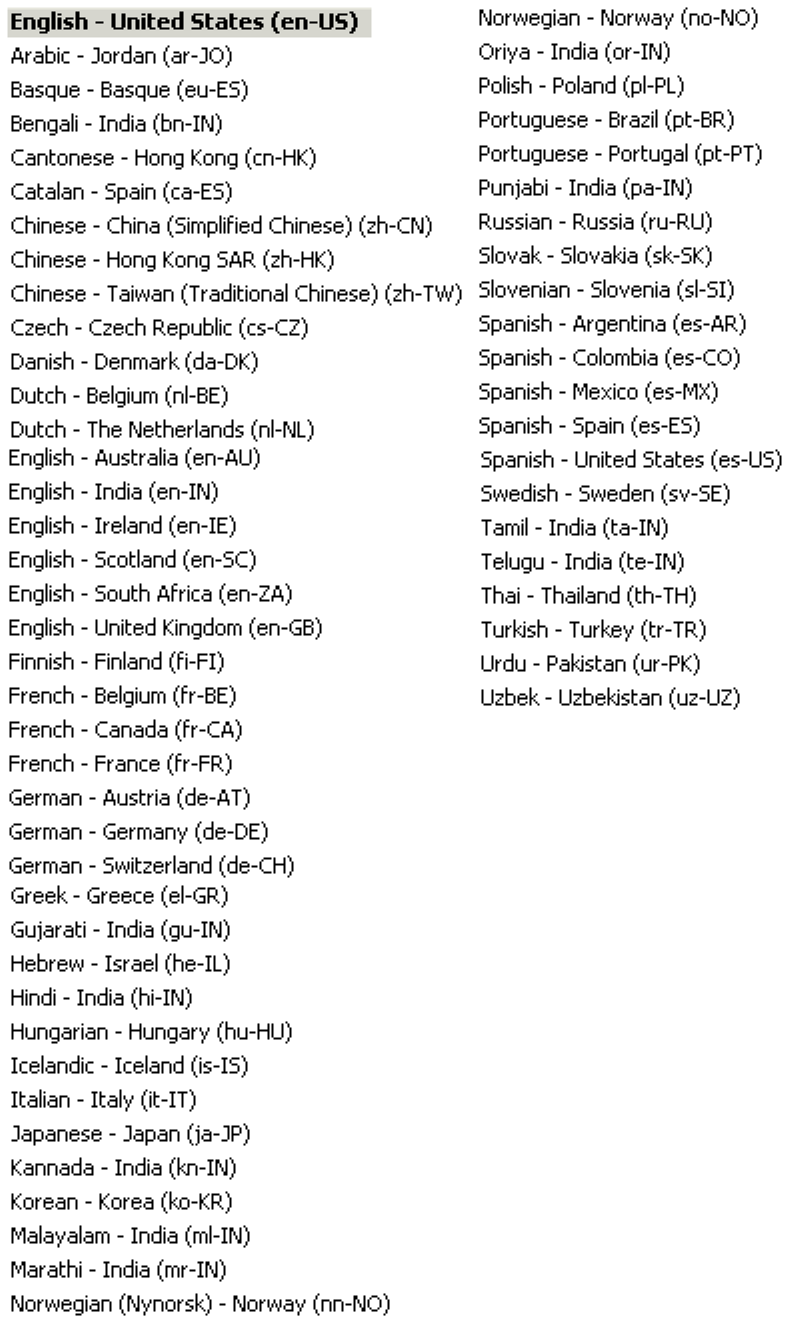
Select the diagram(s) to be updated with the default locale:
English - United States (en-US)

☐ Callflows/Main.callflow

3. Select the diagrams to be updated with the default locale language.
4. Click **OK**.

Locales Supported

The Locales dialog box lists the following locales for selection:



English - United States (en-US)
Arabic - Jordan (ar-JO)
Basque - Basque (eu-ES)
Bengali - India (bn-IN)
Cantonese - Hong Kong (cn-HK)
Catalan - Spain (ca-ES)
Chinese - China (Simplified Chinese) (zh-CN)
Chinese - Hong Kong SAR (zh-HK)
Chinese - Taiwan (Traditional Chinese) (zh-TW)
Czech - Czech Republic (cs-CZ)
Danish - Denmark (da-DK)
Dutch - Belgium (nl-BE)
Dutch - The Netherlands (nl-NL)
English - Australia (en-AU)
English - India (en-IN)
English - Ireland (en-IE)
English - Scotland (en-SC)
English - South Africa (en-ZA)
English - United Kingdom (en-GB)
Finnish - Finland (fi-FI)
French - Belgium (fr-BE)
French - Canada (fr-CA)
French - France (fr-FR)
German - Austria (de-AT)
German - Germany (de-DE)
German - Switzerland (de-CH)
Greek - Greece (el-GR)
Gujarati - India (gu-IN)
Hebrew - Israel (he-IL)
Hindi - India (hi-IN)
Hungarian - Hungary (hu-HU)
Icelandic - Iceland (is-IS)
Italian - Italy (it-IT)
Japanese - Japan (ja-JP)
Kannada - India (kn-IN)
Korean - Korea (ko-KR)
Malayalam - India (ml-IN)
Marathi - India (mr-IN)
Norwegian (Nynorsk) - Norway (nn-NO)
Norwegian - Norway (no-NO)
Oriya - India (or-IN)
Polish - Poland (pl-PL)
Portuguese - Brazil (pt-BR)
Portuguese - Portugal (pt-PT)
Punjabi - India (pa-IN)
Russian - Russia (ru-RU)
Slovak - Slovakia (sk-SK)
Slovenian - Slovenia (sl-SI)
Spanish - Argentina (es-AR)
Spanish - Colombia (es-CO)
Spanish - Mexico (es-MX)
Spanish - Spain (es-ES)
Spanish - United States (es-US)
Swedish - Sweden (sv-SE)
Tamil - India (ta-IN)
Telugu - India (te-IN)
Thai - Thailand (th-TH)
Turkish - Turkey (tr-TR)
Urdu - Pakistan (ur-PK)
Uzbek - Uzbekistan (uz-UZ)

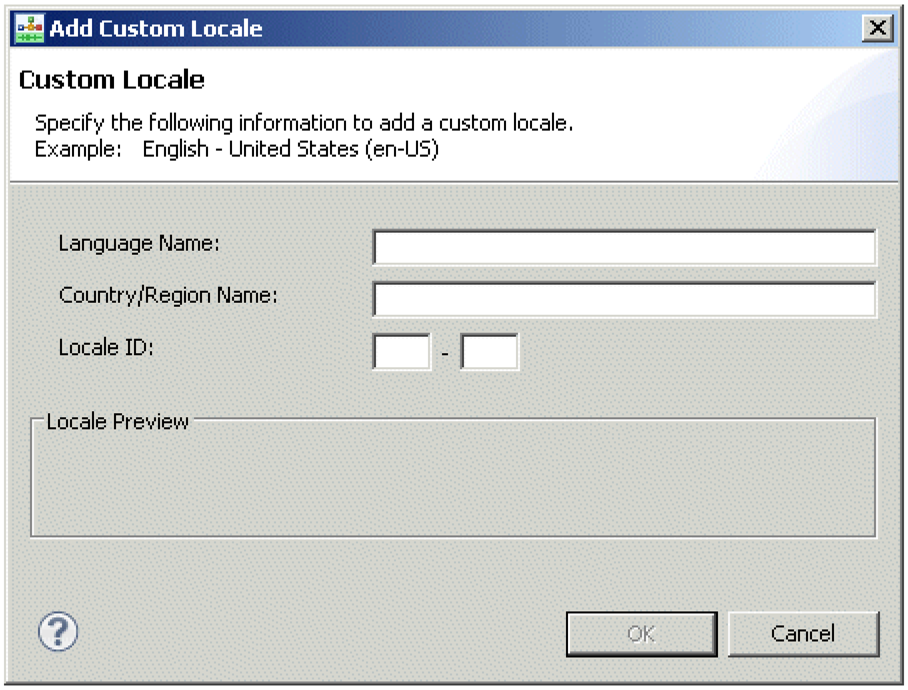
Defining Custom Locales

There are two methods to define locales not already pre-defined in Composer:

- When creating a new Project via the New Composer Project wizard. Select **File** > **New** > **Java**

Composer Project or **File** > **New** > **.NET Composer Project**. The Define Composer Project Locales page appears after the option to select a template. Click **Add Custom** to open the Add Custom Locale dialog box shown below.

- For an existing Project, right-click the Composer Project in the Project Explorer, select **Properties,** select the **Locales**, and then click **Add Custom**. This opens the Add Custom Locale dialog box shown below.


Add Custom Locale

Custom Locale

Specify the following information to add a custom locale.
Example: English - United States (en-US)

Language Name:

Country/Region Name:

Locale ID: -

Locale Preview

1. Enter the **Language Name**.
2. Enter the **Country/Region Name**.
3. Enter the **Locale ID** by specify the two-letter language codes. Examples:
    - en-IN (which represents English-India)
    - en-SC (which represents English-Scottish)
    - en-IE (which represents English-Irish))
4. Click **OK**. After clicking OK, the custom locale is listed for selection on the Locales properties page. Adding a custom locale also activates the **Delete Custom Locale** button on that page.
5. Create the processing logic. See Processing Prompts Other Than en-US.

Locales Other than United States

When Composer is first installed, the default locale is set to English (en-US). Composer bundles a sample set of prompt file resources specifically for en-US, which is located in the following Composer Project directory `../Resources/Prompts/en-US/` The prompt audio resources located in this directory are used when processing a prompt. such as a prompt for a date/time, and so on. Composer provides the business logic to process prompts for locale United States -- English (en-US). This file is located in the Composer project location: `../Resources/Prompts/en-US/en-US.js`

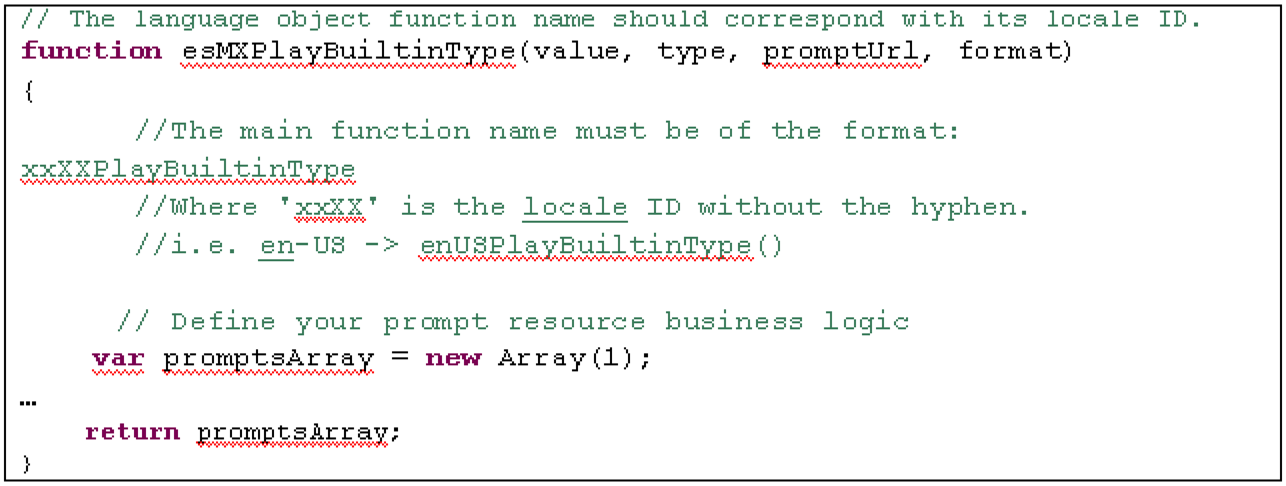
Processing Prompts Other Than en-US

When using locales other than en-US in a callflow application, you must define your own prompt business logic. Certain steps are needed to ensure that the prompts will process correctly. The steps are as follows:

1. Under each locale directory (`../Resources/Prompts/<locale>/`), create a JavaScript locales file with the locale's ID as its file name. A sample locale template JavaScript file is provided with instructions located under the Composer Project: `../Resources/Prompts/xx-XX.js`.For example:`..Resources/Prompts/es-MX/es-MX.js`

2. Define the contents of the locale file created in previous step:

The function name must be of format xxXXPlayBuiltinType, where xxXX is the locale ID without the hyphen; for example: `es-MX   =` esMPlayBuiltinType In the PlayBuiltinType function, define the business logic for processing the prompt for that particular locale.



```
// The language object function name should correspond with its locale ID.
function esMXPlayBuiltinType(value, type, promptUrl, format)
{
       //The main function name must be of the format:
xxXXPlayBuiltinType
       //Where 'xxXX' is the locale ID without the hyphen.
       //i.e. en-US -> enUSPlayBuiltinType()

      // Define your prompt resource business logic
    var promptsArray = new Array(1);
…
    return promptsArray;
}
```

Note: When declaring other functions in the JavaScript locale file, the function names must not duplicate the function names in other JavaScript locale files. Composer recommends pre-pending the locale ID to the function names, i.e., for locale es-MX use esMXMyFunctionName).

Developing Multi-Lingual Applications

This section provides information on how you can build multilingual applications in Composer. **How Can I Work with Multiple Languages in a Composer Project?** A Composer Project has a working set of active language locales that maybe used in an application. These language locales can essentially be used within a Composer application. You may set the Project locales in two ways.

1. During the creation of a Composer Project, the new Composer Project wizard lets you select the default

and active locales within the Project.

2. Once a Composer Project is created, you may also modify the working set of language locales through the Project properties under the Locales section.

How Can I Change the Language for an Application at Runtime? For a multilingual application, if the application needs to change the language for the prompts and grammars based on your input or other settings, use the following steps:

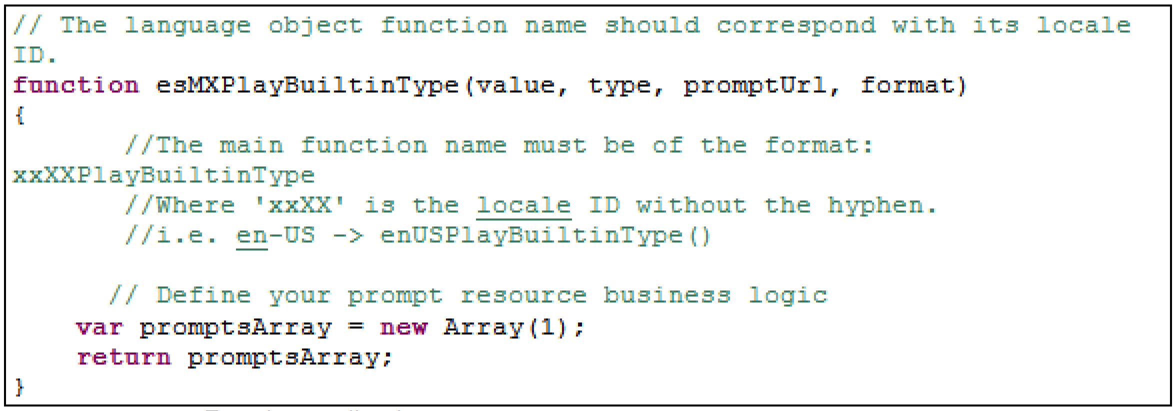
1. On the callflow diagram, the Entry block variables property will contain the variable APP LANGUAGE . Set the value of this variable to the correct TTS language value.
2. Place the appropriate blocks and links on the diagram. To switch to another language in the callflow, use the Set Language block.
3. The blocks subsequent to the Set Language block will then be of the switched language. When using Prompt blocks, the prompt resources located in the appropriate language locale directory will be used. For example:
    - United States -- English (en-US) will use resources from `../Resources/Prompts/en-US/`
    - France -- French (fr-FR) will use resources from `../Resources/Prompts/fr-FR/`

If you are using a TTS engine and/or an ASR engine, you must have the language packs for the TTS vendor and/or the ASR vendor installed on the TTS and/or ASR servers. Why does my application fail to play prompts of different locales? When Composer is first installed, the default locale is set to English (en-US). Composer provides the business logic to process prompts for en-US. In a callflow application, when using locales other than en-US, some steps are needed to ensure that the prompts will process correctly. See the Using Non-U.S. Locales below for more information.

Multi-Lingual Application Prompts

You can create applications that play a prompt type resource in one language (for example: `../Resource/Prompts/en-US/`), switch the language using Set Language block, then play another prompt type resource using a different language (for example: `.../Resource/Prompts/es-MX/`). **Multi-Lingual Application Prompts Using Composer 8.0.4 and Later** To create a multi-lingual application prompts:

1. Create a multilingual callflow diagram where the language is being switched using the Set Language block and the prompts are set accordingly.
2. Generate the code for the callflow diagram file. The VXML is generated.
3. Under each locale directory (`..Resources/Prompts/<locale>/`), create a JavaScript locales file with the locale's ID as its file name. Use the template locales file provided in the Project Explorer (..Resources/Prompts/xx-XX.js) and follow its instructions about creating the corresponding locales .js file; for example: `..Resources/Prompts/es-MX/es-MX.js`
4. In the prompts locale file, created in previous step, implement the following content:
    - Create the language object function. The function name must be of format 'xxXXPlayBuiltinType', where 'xxXX' is the locale ID without the hyphen. For example: `es-MX -> esMXPlayBuiltinType`
    - In the language object function, implement the main function as shown below and define the business logic for processing the prompt.



```
// The language object function name should correspond with its locale
ID.
function esMXPlayBuiltinType(value, type, promptUrl, format)
{
       //The main function name must be of the format:
xxXXPlayBuiltinType
       //Where 'xxXX' is the locale ID without the hyphen.
       //i.e. en-US -> enUSPlayBuiltinType()

      // Define your prompt resource business logic
    var promptsArray = new Array(1);
    return promptsArray;
}
```

5. Run the application

Using 8.0.3 or Older Composer Projects To upgrade an existing multi-lingual application with the en-US locale:

1. In the Project Explorer, right-click the Project and select **Upgrade to Composer 8.1**.
2. (Optional) If you have previously modified `PlayBuiltinType.js`, copy over any changes. The resource '`../Resources/Prompts/en-US/PlayBuiltinType.js`' has already been upgraded. Previous Composer Project files are archived in the archive folder during an upgrade.
3. Under each locale directory '`..Resources/Prompts/<locale>/`', create a JavaScript locales file with the locale's ID as its file name. Use the locales template file provided in the Project Explorer (`..Resources/Prompts/xx-XX.js`). Follow its instructions about creating the corresponding locales .js file; for example: `..Resources/Prompts/es-MX/es-MX.js`
4. In the prompts locale file created in previous step, implement the following content:
    - Create the language object function. The function name must be of format 'xxXXPlayBuiltinType', where 'xxXX' is the locale ID without the hyphen. For example: es-MX -> esMXPlayBuiltinType
    - In the language object function, implement the main function as shown as shown above in step 4.b. for Composer 8.0.4 and Later and define the business logic for processing the prompt.

Non en-US Locales

To upgrade an existing multi-lingual application with non en-US locales:

1. In the Project Explorer, right-click the Project and select **Upgrade to Composer 8.0.4**.
2. Under each locale directory '`..Resources/Prompts/<locale>/`', create a JavaScript locales file with the locale's ID as its file name. Use the template locales file provided in the Project Explorer (`..Resources/Prompts/xx-XX.js`) and follow its instructions about creating the corresponding locales `.js` file; for example: `..Resources/Prompts/es-MX/es-MX.js`
3. In the prompts locale file created in the previous step, implement the following content:

- Create the language object function. The function name must be of format ‘xxXXPlayBuiltinType’, where ‘xxXX’ is the locale ID without the hyphen. For example: `es-MX -> esMXPlayBuiltinType`
- In the language object function, implement the main function with the business logic for processing the prompt inline.

Time Zone Preferences

Composer displays all date/time elements in the user-preferred time zone with the time zone identifier. You can change the preferred time zone in **Window** > **Preferences** > **Composer** > **Context Services**.

Using User Data

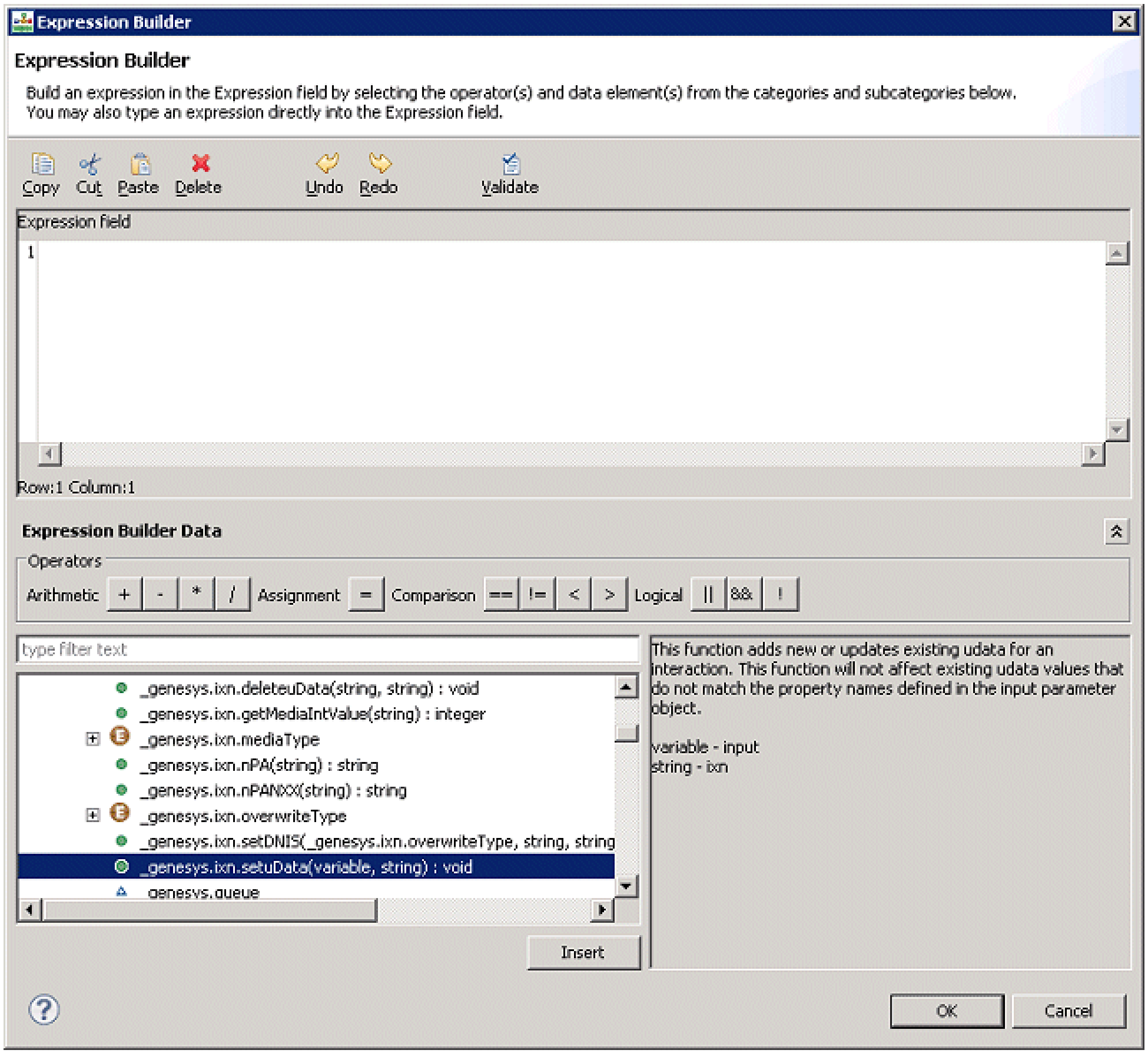
Blocks for User Data

To work with User Data in Composer, you can use:

- The Interaction Data block for voice applications.
- The User Data property of the External Service block if you wish to pass User Data to an external service (routing and voice).
- The Entry block to access User Data (routing and voice).

For routing applications, you can use:

- The User Data block.
- The Create Email block, which lets you pick up standard response text from User Data.
- The Create SMS block, which lets you pick up message text from User Data.
- The Identify Contact block, which provides an option to update the interaction's User Data with the parameters returned by Universal Contact Server (Contact attribute data).
- The Create Interaction block, which lets you create a new interaction record in UCS database based on User Data.
- The ECMAScript block. The Script property lets you use Universal Routing Server User Data functions. Open Expression Builder. Select URS Functions and then _genesys.ixn.deleteuData (to add a User Data property or delete all properties) or _genesys.ixn.setuData (to add new or update existing User Data).



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

_genesys.ixn.deleteuData(string, string) : void
_genesys.ixn.getMediaIntValue(string) : integer
_genesys.ixn.mediaType
_genesys.ixn.nPA(string) : string
_genesys.ixn.nPANXX(string) : string
_genesys.ixn.overwriteType
_genesys.ixn.setDNIS(_genesys.ixn.overwriteType, string, string
_genesys.ixn.setuData(variable, string) : void
_genesys.queue

This function adds new or updates existing udata for an interaction. This function will not affect existing udata values that do not match the property names defined in the input parameter object.

variable - input
string - ixn

Hints

- A specific variable 'xyz' can be accessed directly; for example: _genesys.ixn.interactions[0].udata.xyz
- To write to User Data, use the setuData() function in an ECMAScript snippet. Usage is similar to the example below.

```
var input = new Object();

input.xyz = InputValue1; // Specify a value for the key 'xyz'.

input['my-key-nname'] = 'value'; // Use this notation if the key or property name has
a hyphen in it. Note that'my-key-nname'has hyphens.
```

```
_genesys.ixn.setuData(input);
```

- Reading User Data is easier using the Assign block than with the ECMAScript block.

Mandatory Data for UCS Blocks

When working with the Update Contact and Render Message blocks (which map to Universal Contact Server services), certain properties must exist in the interaction User Data.

For the Update Contact block, ContactId must exist.

For the Render Message block, ContactId (if some contact-related Field Codes (as described in the *eServices 8.1 User's Guide*) are used in the text to render). Also InteractionId (if some interaction-related Field Codes are used in the text to render)and OwnerEmployeeId (if some agent-related Field Codes are used in the text to render).

As is the case with IRD, these properties are not set in the blocks themselves. Instead, the properties are assumed to be put in the interaction's User Data by some other block earlier in the workflow, such as the Identify Contact block or Create Interaction block with the Update User Data property set to true. In case no other block does this, the User Data block may be used for this purpose.

If these properties are not available, an explicit UCS error message (missing parameter) shows in the Orchestration Server log.

Callflow User Data

Also see the following blocks used for callflows:

- Interaction Data
- Route Request

Variables Mapping

The Context Services blocks listed below support variables mapping.

- Identify Customer
- Query Customer
- Query Services
- Query States
- Query Tasks

In addition, the eServices block, Identify Contact, also supports variables mapping. Each of the above blocks can return data as JSON objects, which can then be stored in callflow/workflow variables. When your application needs to access a property in the JSON object, you must use an Assign block to copy the property value into a variable where it can be accessed. Variable mapping allows you to specify the variables for certain properties of the resulting JSON object and, at run time, populate the variables with the property values. When the returned data is an array of JSON objects, the variables are populated with the property values of the first item in the array. In cases where an application needs to iterate over each of the items in the array and also populate the variables with data from each item, use the Looping block with a reference to the block responsible for the retrieving the data. To set up variable mapping for a block, use the Variables Mapping dialog box, which opens from the Variables Mapping block property in one of the Context Services blocks listed above.

1. Click under **Value** to display the ... button.
2. Click the ... button to open the Variables Mapping dialog box.
3. Click **Add** to open the Configure Variables Mapping dialog box.
4. Click the down arrow opposite **Variable** and select the variable into which the value needs to be stored.
5. Opposite **Mapping**, you can specify it in two ways:
    - Type in the property path to access the JSON property.
    - Select from a tree. Since the property hierarchy specification may be tedious and prone to errors, a mapping selector tree is provided below the Mapping text field. Clicking on any node within the tree will update the Mapping text field with the JSON property hierarchy of the selected node. If the selected node is an element within a JSON Array, then the mapping will display a [ ] implying that the hierarchy contains an array. Leaving the [ ] will automatically default to [0] in first item of the array.
6. Click **OK**. The Variables Mapping dialog box reflects the assignment.
7. Click **Add** again to map another variable.
8. You may continue assigning customer attributes to variables in this fashion.

Sample Applications and Templates

Composer provides a set of predefined Project templates containing sample applications.You can either start off with a blank Project template or use one of the predefined templates as a starting point.

Project Templates

To create a new Project using a template:

1. In Composer perspective:
    - **File** > **New** > **Java Composer Project,** or
    - **File** > **New** > **.NET Composer Project**.
2. Specify the name of the Project. By default all Projects will be saved in your workspace location.
3. In the Project dialog box, type a name for your Project. 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 Project.
4. Select the Project type:
    - **Integrated Voice and Route**. Select to create a Project that contains both callflows and workflows that interact with each other. For example, a routing strategy that invokes a GVP voice application. For more information on both voice and routing applications, see How Do Voice Applications Work and What Is a Routing Workflow, respectively.
    - **Voice**: Select to create a Project associated with the GVP 8.x. This type of Project may include callflows, and related server-side files. For more information on this type of Project, see topic, How Do Voice Applications Work.
    - **Route**: Select to create a Project associated with the URS 8.x SCXML Engine/Interpreter. For more information on this type of Project, see topic, What Is a Routing Workflow.
5. Click **Next**.
6. Expand the appropriate **Project Type** category and select a template for your application.
7. Click **Next.**
8. Select the Project Locale(s) to be used for prompts, grammars, and other locale-related resources.
9. Click **Finish**. The new Project folder set will be displayed in the Project Explorer.

Diagram Templates

You can save a diagram as a template and have it appear on the list of available templates when creating a new callflow diagram or creating a new workflow diagram. Diagrams saved as templates can exported to/imported from the file system.

Saving a Diagram as a Template

To save a diagram as a template:

1. In the Project Explorer, right-click the diagram in the Workflows or Callflows folder.
2. Select **Save** <Callflow or Workflow> **as Template**. The Add Template dialog box opens.
3. Name and describe the template.
4. Click **OK**. Upon a successful save the following message appears: Custom template added to your configuration.

Accessing Saved Diagram Templates

To view a diagram previously saved as template:

1. From the File menu, select **New** <Callflow or Workflow> **Diagram**. The New <Callflow or Workflow> dialog box opens. The template appears in the Main Workflow tab under Custom Templates.
2. Select the template and click one of the following:
    - **Next** to name the diagram, select the Project, and then click **Finish**.
    - **Finish** to keep the template name and save in the Workflows or Callflows folder under the current Project.

Removing a Diagram Template

To remove a diagram template from the Custom Templates list, use Customization Manager.

Exporting a Diagram Template to the File System

To export a diagram to the file system or another user's Composer:

1. In the Project Explorer, right-click the diagram in the Workflows or Callflows folder.
2. Select **Export**. The Export dialog box opens.
3. Under General, select **File System** and click **Next**.
4. In the File System dialog box, select the folder containing the diagram(s).
5. On the right, click check boxes to indicate the diagrams (s) to export.
6. Opposite **To directory**, select the Composer installation to export to or Browse for the destination directory.
7. Under **Options**, select one of the following:
    - Overwrite existing files without warning
    - Create directory structure for files
    - Create only selected directories (default).
8. Click **Finish**.

Importing a Diagram Template

To import a diagram previously saved as a template:

1. In the Project Explorer, right-click the diagram in the Workflows or Callflows folder.
2. Select **Import**. The Import dialog box opens.
3. Under **General**, select **File System** and click **Next**.
4. In the File System dialog box, opposite **From directory**, click **Browse**.
5. Open the workspace directory followed by the Project folder.
6. Within the Project folder, select the Workflows folder that contains the template to import and click OK.
7. In the File System dialog box on the right, click check boxes to indicate the template(s) to import.
8. Opposite **Into folder**, browse for and select the folder to import into.
9. Under Options, select one of the following:
    - Overwrite existing files without warning
    - Create directory structure for files
    - Create only selected directories (default).
10. Click **Finish**.

Editing a Diagram Template

Use Customization Manager: **Window** > **Show View** > **Other** > **Customization Manager**. Select

Workflow Diagram Editor or **Callflow Diagram Editor.**

GVP Voice Project Templates

Composer provides the following Project templates for voice applications:

- Business Logic Project
- CCXML Project
- Database Query Result Access Project
- Database Stocks Project
- NBest Results Handling Project
- OSDM Project
- Transfer Project
- User Input
- Voice Recording

Sample: NBest Results Handling Project

This application demonstrates processing of NBest Results and confirming the user input. For example, in a voice application there may be cases where the application must clarify a user response, such as:

- A Speech Recognizer may return multiple results in the case of a noisy environment or indistinct pronunciation of words.
- Grammar complexities may cause the user's input to be ambiguous.

NBest processing logic helps a voice application to clarify user responses.

1. The Entry block enables NBest properties:
    - Instructs the Speech Recognizer to return multiple results by setting maxnbest to an integer greater than one.
    - Sets the confidencelevel decimal value (value values from 0.0 to 1.0) indicating the recognizer's confidence that the utterance matches what the user actually said.
2. An Input block Obtains the results:
    - Instructs the user about the expected input details.
    - An external grammar file will be used by the ASR engine to process the user input.
    - Enables shadow variables.
3. A Branching block uses the Input block’s shadow variable to check whether there are multiple results.

4. An Assign block assigns the obtained results to a variable:
    - Creating a custom VXML page to process the multiple results array:
    - Iterating over the Array to prompt the user about the results.
    - Asking the user to confirm the exact result using a simple inline grammar.
5. A Subdialog block Invokes the NBest Processing VXML page:
    - `ProcessNBestResults.vxml` has been placed inside the src directory.
    - Passing the results (LastResult variable) as input for the VXML page.
    - Defining an Output variable to receive the return result from the VXML page.
6. A Prompt block announces the user confirmation result.

Integrated Voice Route Project Templates

When creating a new Composer Project using an Integrated Voice and Routing Project template, you can select from various sample applications. A few are described below.

Sample: External File-Based Routing

This sample application demonstrates voice call routing based on criteria contained in an external file.

1. The IPD contains a single Workflow block pointing to a workflow diagram named default.workflow.

2. The default.workflow diagram starts with an Assign block that sets the URL of the target definition. In this case, the target definition (routing criteria and target preferences) is defined in the target.txt file contained in the src folder of the Project (but which could be defined in a file external to the Project). Routing target preferences are based on criteria such as the number dialed (DNIS), whether the inquiry is about home or auto insurance, the caller's language, claim history, discount percentage, and so on. Calls will be routed to various agent groups/queues based this information, which is assumed to be contained in the user data of the voice interaction.

3. The next block in the default.workflow diagram is a Backend block, which is used for invoking custom backend Java Server Pages; in this case, to execute the routing logic. Using the target.txt file DNIS criteria, the Backend block parses the target definition and selects a list of matching targets that could be routed to.

4. A Branching block then segments interactions to take different paths in the workflow.

    - If a matching target is found, the Branching block sends the voice interaction to a Looping block, which iterates until an exit condition is met; in this case, when an available target is found, which could be an agent, agent group, or the result of a Skill Expression.
    - If repeated Looping block iterations do not find an available target, the voice interaction goes to a Play Application block, which could inform the caller that all agents are busy and to call back later.

Sample: Load Balancing and Working Hour Routing

The Assign Block assigns the following variables:

- CallerDay using the Orchestration Server function available in Expression Builder called _genesys.session.dayInZone('ECT')
- CallerTime using the Universal Routing Server functions available in Expression Builder called _genesys.session.timeInZone('ECT')

Next, a Branching block checks for the following conditions:

- If CallerDay is Saturday or Sunday, then a Play Application block play the Weekend message to the caller.
- If CallerDay is a week day and CallerTime is after business hours, then a Play Application block plays the After Business Hours message to the caller.
- If CallerDay is a week day and CallerTime is within business hours, then a Target block is used to load balance calls between two agent groups.

Sample: Play Application & Busy Treatment

After the Entry block, this sample starts with an Identify Customer block that attempts to identify the calling customer in the Universal Contact Server Database based on the calling party's number stored in the ANI variable defined in the Entry block. The Customer Attributes property specifies to search/retrieve core customer profile attributes for the phone number.

1. A Log block logs customer information retrieved from the database. The Logging Details property specifies a JSON string for the format of the customer data.
2. A Play Application block plays a welcome message to the caller. The Language property indicates the customer's language is English. The Parameters property specifies the customer's first and last name from the JSON string.
3. A Target block routes the call based on a skill expression to a target with an English skill greater than 9 and a Spanish skill greater than 5.
4. If there are no available routing targets, a Play Message block plays a message to the caller. The Prompts property defines an expression (created in Expression Builder) that calls a Orchestration Server function (_genesys.statistic.sData). This function returns the value of a statistic (ExpectedWaitTime) for a specified object, in this case, a Skill Expression. This could be announced to the caller via an IVR application while waiting for a target.
5. Once the call is successfully routed to an agent, a Create E-mail block creates an acknowledgement e-mail to the customer in the form of pre-written text from the Standard Response library. The customer's e-mail address is contained in the customeremail variable defined in Entry block.

Sample: Routing Based on DNIS and ANI Project

This sample application demonstrates how the originating phone number (ANI) and dialed phone number (DNIS) values are automatically made available as variables when you create a workflow.

1. The BranchOnDNIS block first checks if the incoming call was made to the DNIS for making a deposit.
2. If the call was not made to the deposit DNIS, the application will play a voice treatment before routing to the withdraw agent. The Withdraw Play Application block points to a voice callflow in the Composer Project. This will execute a VXML page which plays a message to the caller.The WithdrawAgent Target block routes the call to the agent responsible for handling withdrawals.
3. If the DNIS did match the one for deposit, the application first plays an application treatment with a message for the caller. It will then examine the ANI of the caller in the BranchOnANI block, which contains the following branching condition: ANI==Customer. In this case, "Customer" is a variable declared in the Entry block of the workflow, and it can be modified to specify the ANI of the "special" customer that you are targeting.

4. The special customer is then routed directly to the agent handling deposits. Non-special customers must go through a voice treatment which records their name before proceeding. The voice treatment is executed by the RecordCustomer Play Application block.

5. A voice callflow is then executed by the RecordCustomer block. After recording the user's name, the Menu block provides the option to listen to the recording, re-record the audio, or exit. Upon exiting the voice treatment, the call will be routed to the deposit agent.

Routing Templates and Samples

This section leads you through some Composer-supplied templates used to create routing applications. It also presents samples, which are not part of Composer's collection of templates.

Composer provides the following Route Project templates:

- Context Services Template
- Database Query Result Template
- Forward to External Resource Template
- Route After Autoresponse Template
- Routing Based on Variables Template
- Routing by Using Web Request Template
- Routing Based on Date and Time Sample
- Routing Based on a Statistic Sample
- Routing Based on Percent Allocation
- Routing Using Web Request Sample
- Last Called Agent

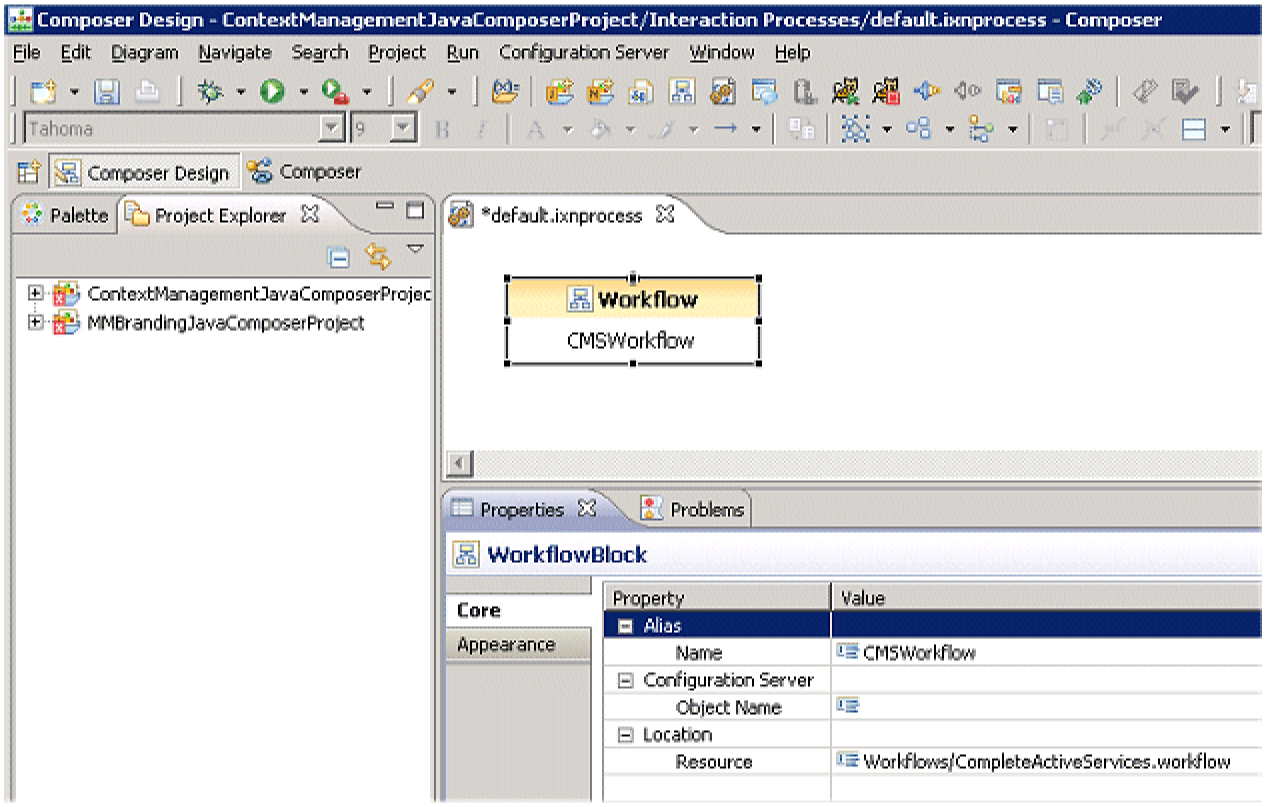
To access the Project templates:

1. Switch to Composer Design or Composer perspective.
2. Select File > New > Java Composer Project.
3. In the Project dialog box, name your Project and indicate whether you want to use the default location.
4. Select the Project type: Route.
5. Click Next. The Select a Composer Project Template dialog box opens.
6. Select a template and click Next.
7. Select the Project locale and click Finish.

Reviewing a Route Project Template

Assume you selected the Context Services Project in step 6 above. This automatically creates an interaction process diagram for routing interactions with a single Workflow block in the default.ixnprocess tab.

1. Double-click the Worfklow block to open the Properties view in the tab underneath.



Composer Design - ContextManagementJavaComposerProject/Interaction Processes/default.ixnprocess - Composer

ContextManagementJavaComposerProjec
MMBrandingJavaComposerProject

WorkflowBlock

Property	Value
Alias	
Name	CMSWorkflow
Configuration Server	
Object Name	
Location	
Resource	Workflows/CompleteActiveServices.workflow

1. In the Properties view, note that the Resource property indicates that the name of the workflow is CompleteActiveServices.workflow.
2. To view this workflow, expand the Project in the Project Explorer.
3. Expand the Workflows folder.
4. Double-click CompleteActiveServices.workflow. A commented workflow appears.
5. View the properties for each block by double-clicking the block.

Context Services Template

This application demonstrates the use of Context Services blocks in various workflows. An interaction with a customer can be interrupted for various reasons -- the customer hanging up, a lost connection, and so on. When the customer calls back to the contact center, the interrupted services can be retrieved and completed. This sample contains workflows that demonstrate completing an active service, resuming a service, and updating the customer profile.

Complete Active Service Workflow

This workflow identifies a customer based on the ANI. If the customer is found, all active services associated to that customer are queried for and completed. **Pre-requisites:** The prerequisites are as follows:

- Set the ANI correctly since it is used to identify the customer.
- Define an identification key in Universal Contact Server to allow the customer to be identified by PhoneNumber.
- Set Context Services Preferences in Composer.

The workflow does the following:

1. Identifies the customer with the Identify Customer block:

    - Identifies by setting the customer attributes to be used to search for the customer. Here, the phone number is matched to the inbound ANI.
    - Specifies the name of the key to be used for lookups (idPhoneNumber). This key must be defined in Universal Contact Server (UCS).
    - Gets the Customer Count and Customer Data information from UCS and assigns the information to variables.

1. Determines the uniqueness of the identified customer. Using the Branching block to check whether the number of customer records returned by Universal Contact Server is 1.
2. Assigns the CustomerID to a variable using the Assign block. If the customer is unique, gets the CustomerID from the customer data retrieved from Universal Contact Server.
3. Queries the active services for the customer using the Query Services block:

    - Sets the Service Identifier to the CustomerID.
    - Sets the Service Status to Active.
    - Stores the Service Data results is in an application variable.

1. Assigns the Service ID using the ECMAScript block. Retrieves the ServiceID from the Service Data.
2. Determines if the retrieved service is active using the Branching block. Determines if the ServiceID is defined.
3. Completes the active service using the Complete Service block

Resume Active Service Workflow

This workflow identifies a customer based on the ANI. If the customer is found, the active service associated to that customer is queried for and the active state within that active service is completed and a new state within the service is entered. **Pre-requisites:** The prerequisites are as follows:

- Set the ANI correctly since it is used to identify the customer.
- Define an identification key in Universal Contact Server to allow the customer to be identified by PhoneNumber.
- Set Context Services Preferences in Composer.

This workflow does the following:

1. On the right side, Identify Customer, ECMAScript, Query Services, and Branch blocks are similar to ones described above for the Complete Active Service workflow.
2. Continuing with the right side of the workflow, a new service is started with the Start Service block if an existing active service is not found. The Identifier is set to the customer ID previously retrieved in the Identify Customer Service block.
3. An ECMAScript block assigns the active service ID to a variable.
4. Using the ECMAScript block results, a Branching block determines if an active state exists for the service.
5. A Complete Services block sets the Service ID to complete the active service.
6. On the left side of the workflow, a Query Services block queries an anonymous service using the ANI as contact key if the customer is not found:
    - Set the ANI as the contact key in the Identifier property.
    - Set the Service Elements to Active States.
    - Store the Service Data in an application variable.
7. A Branching block determines if the service data retrieved from the Query Services block is defined.
8. A Start Service block starts a new service if no anonymous active service is found:
    - Set the Identifier to the ANI.
    - Store the Service ID in an application variable.
9. An Enter State block causes the service to enter a new state:
    - Set the Service ID.
    - Set the previous State ID.
    - Set the State Type property to filter for the specific Service State type.

Update Profile Workflow

This workflow identifies a customer based on the ANI. If the customer is found, the contact setting of the customer profile is updated by setting the medias/voice extension attribute value to true. **Pre-**

requisites: The prerequisites are as follows:

- Set the ANI correctly since it is used to identify the customer.
- Define an identification key in Universal Contact Server to allow the customer to be identified by PhoneNumber.
- Define the medias customer profile extension in Universal Contact Server and that extension should have the Boolean attribute voice defined.
- Configure Context Services Preferences in Composer.

This workflow does the following:

1. The first two blocks Blocks are described in the previous Complete Active Services workflow.
2. An Assign block saves the CustomerID and customer data information:
    - Retrieve and assign the CustomerData to an application variable
    - Retrieve and assign the CustomerID to an application variable
3. An Update Customer block updates the customer profile:
    - Specify the extension customer profile data. Select the medias extension and set the voice attribute to true.
    - Set the CustomerID.
    - Select the Insert Extension operation to insert the extension record.

Database Query Result Template

This application demonstrates how to access the results of a database query in a workflow.

1. First, the application makes a query in the GetStockQuotes DB Data block. The query is defined using the Query Builder interface. The results of the query are saved to variables call DBColumnNames and DBRecords, which are defined in the Entry block. They are assigned using the Database Result Set properties of the DBData block.
2. Next, the FetchFirstRecord ECMAScript block uses a script to access the data. The DBRecords variable is a two-dimensional array containing the rows and columns of the returned data. The script in the FetchFirstRecord block extracts and assigns the first row of to a variable called DBCurrentRecord.
3. The SaveToVariables Assign block assigns the columns of the row in DBCurrentRecord into separate variables.
4. The next DB Data block, DBData1, uses the variables assigned in the previous step to make a stored procedure query to the database. The stored procedure is defined with the Stored Procedure builder interface, which is similar to the Query Builder.
5. The FetchNextRecord ECMAScript block runs a script to load the next row from the DBRecords variable obtained by the first query.
6. The CheckMoreRecord Branching block checks if there are any more records (rows) in the result set and will loop back if there are in order to process the next row and repeat the stored procedure query in DBData1. Once there are no more records to process, the workflow exits.

Forward to External Resource Template

This application demonstrates screening a customer e-mail to see if it describes a known problem so that a standard (automatic) response can be sent to the customer. If the e-mail does not describe a known problem, then an acknowledgement e-mail is sent to the customer and the customer's e-mail is forwarded to an external resource (such as an expert or knowledge worker) for a reply. When the reply is received from the external resource, the reply is forwarded to the customer.

1. The interaction process diagram (IPD) starts with a Media Server block, which defines Customer E-mail and External Resource Reply endpoints. Defining two endpoints allows e-mails arriving from customers and external resources to get into the appropriate interaction queues.
2. In the first column of blocks in the IPD, the InboundEmailQ queue sends e-mails to a Workflow block, which points to the CreateForwardEmail.workflow. The workflow first screens the incoming customer e-mail using a Screen Interaction block.

    - If a screening rule match is found, an e-mail response is sent to the customer. This is achieved by using an E-mail Response block with the Response Type property set to Autoresponse. The Output Queue property is set to SendEmailQ, which allows the newly created e-mail to be queued and processed by SendCreatedEmail.workflow.

Note: The SendEmailQ queue is used by all the e-mails created in this sample (even though it is not explicitly stated in order to avoid duplication). In cases where the Output Queue property is not specified, the system interaction queue is used and the e-mail is automatically sent out by Orchestration Server. Though this latter approach does not allow for any further processing before the e-mail is sent out, it is a satisfactory approach for many scenarios.

- If a screening rule match is not found, the workflow sends an acknowledgement to the customer and the customer e-mail is forwarded to the external resource.

3. An E-mail Response block is used for the acknowledgement e-mail with the Response Type property set to Acknowledgement.
4. An E-mail Forward block is used to forward the e-mail to the external resource with Forward Type set to Forward. At this point, the external resource is expected to respond back with a reply to the forwarded e-mail.
5. The second column of blocks in the IPD is used for processing the reply from the external resource. The External Resource Reply endpoint directs the reply from the external resource into the OutboundReplyQ queue. This queue then directs the reply into the ReplyToCustomer Workflow block, which points to the CreateReplyToCustomerEmail.workflow. This workflow uses the Email Forward block with Forward Type set to ReplyToCustomer with a Standard Response to format the customer reply. It also defines the To address (Originating Email -- From) specifying that the e-mail is to be sent to the same address from which the incoming customer e-mail originated. The reply e-mail then goes to the SendEmailQ queue.
6. The third column of blocks in the IPD is used for sending the reply e-mail to the customer. The SendEmailQ queue is connected to the SendEmail Workflow block, which points to the SendCreatedEmail.workflow. This workflow contains a Send E-mail block used for sending the external resource reply to the customer.

Route After Autoresponse Template

This application demonstrates using the Create E-mail block to send an e-mail auto-response to an incoming e-mail and then using the Route Interaction block routes the e-mail to an agent.

1. When an e-mail (new multimedia interaction) arrives at the Media Server Endpoint1, it gets queued to IncomingEmailQueue.
2. Orchestration Server then pulls the interaction from the queue and starts processing the RouteToAgent workflow.
3. The RouteToAgent workflow uses the Create E-mail block to first send out an auto-response e-mail to the customer notifying that the system has received their e-mail and someone be contacting them shortly.After sending the auto-response e-mail, the original e-mail from the customer is then routed to an agent for a response.
4. The workflow then uses the Route Interaction block to route to a target.
5. The Route Interaction block also specifies a suggested queue for the new interaction (agent's response e-mail).
6. After the agent responds (using reply feature in Genesys Agent Desktop), the response is now treated as a new interaction is then placed into the suggested queue.
7. The agent then closes the current interaction (using Done feature in Genesys Agent Desktop).
8. When Orchestration Server pulls the new interaction from AgentReplyProcessing queue, the ProcessAgentReply workflow gets executed.
9. ProcessAgentReply is a simple workflow which routes the agent response e-mail to a supervisor for review.
10. Once the supervisor is satisfied with the response, the email is placed in the system queue and sent out to the customer.

At this point, there are no further interactions that need to be processed and the application (session) exits.

Routing Based on Variables Template

This application demonstrates how to route calls to DNs dynamically using variables and force routing.

1. In the GetRoutingDN Backend block the application accesses some backend business routing logic defined in the file ../include/SomeRoutingLogic.jsp. This business logic will pass back a DN to route to in the varRoutingDNFromLogic variable.
2. Depending of the DN set in the varRoutingDNFromLogic variable, the Branching block will conditionally route to the appropriate targets.
3. In the case where the variable varRoutingDNFromLogic is above 3010, the flow will be forced to route to RoutePoint 3000 as shown in the ForceRouteToDN Target block. After this is done the workflow will exit.
4. In the case where the variable varRoutingDNFromLogic is not above 3010, the DN will first be formatted in the DNVariableFormat Assign block and then be routed by the RouteToDynamicDN Target block. After this is done the workflow will exit.

Route by Using Web Request Template

This application demonstrates using the Web Request block to invoke an HTTP web request.

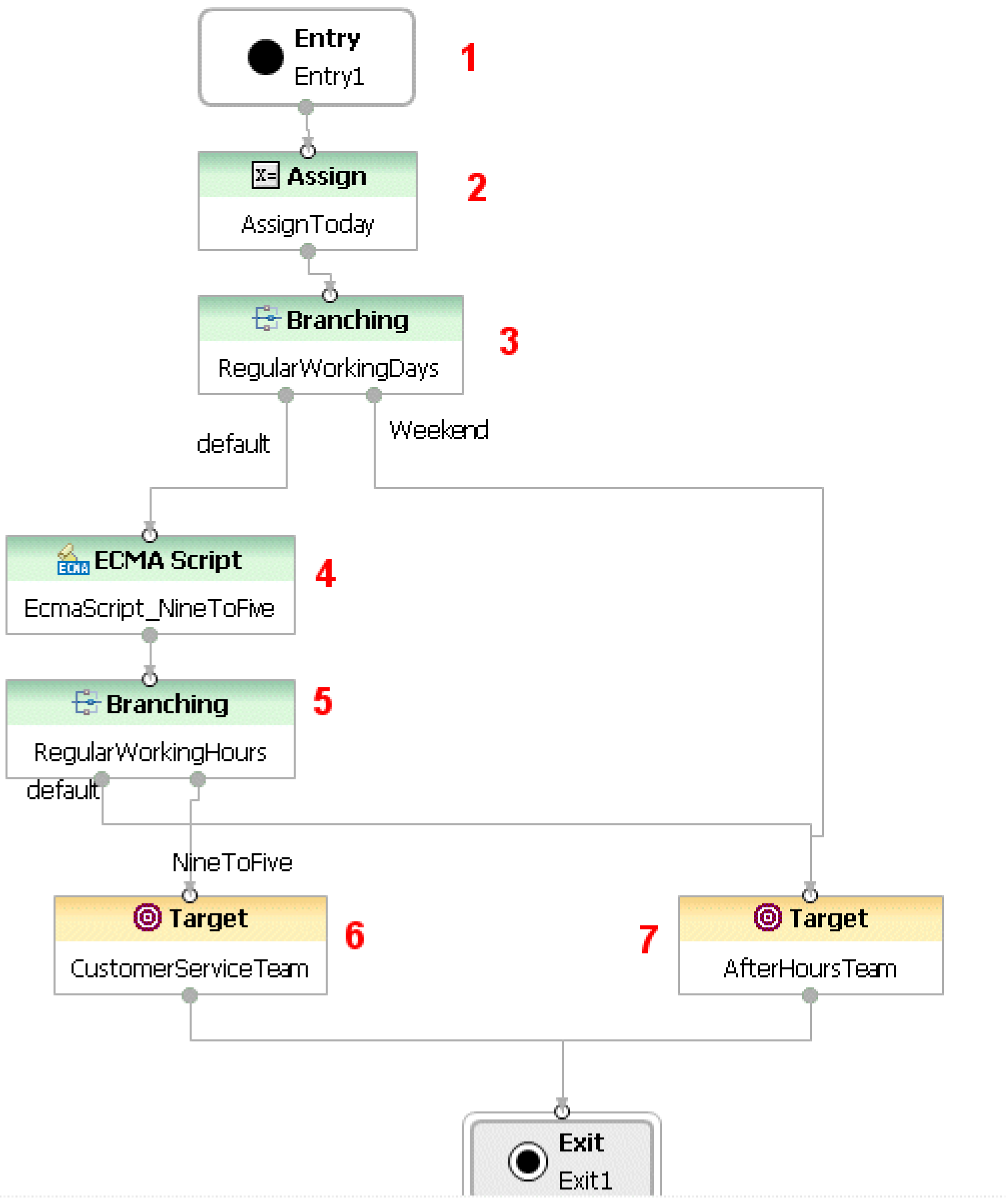
1. The Entry block contains three user-defined variables:
    - weburl: Points to the getMemoryStatus.jsp application from the src directory. This represents an application hosted on an external application server.
    - webresult: Holds the result of the get request.
    - assignresult: The content of the webresult variable is transferred to this variable in the Assign block.
2. The Web Request block Exceptions Properties view is configured as follows:
    - The Exceptions property indicates support for the error.sesson.fetch event.
    - The Request Method property has two choices: get or post with get selected indicating the type of request (HTTP Get) that Universal Routing Server will make.
    - The Uri property indicates that the Uri is contained in a variable called weburl.
    - The Authentication Type property has two choices: anonymous or basic with anonymous selected as a security setting. With this type of access, no user name/password is passed to the Web service for client authentication in order to get data.
    - The Result property indicates that the data from the get request will initially be stored in a variable called webresult.
3. The Assign block, Assign Data property indicates that the data received from the get request is assigned to a variable called assignresult.

Routing Based on Date & Time Sample

This application demonstrates routing based on business hours. A customer care company separates customer calls based on working days (Monday - Friday) and working hours (09:00 -17:00). Based on the working days and hours, the company sends the customer call to one of two targets.

- Agent Group #1: CustomerServiceTeam - Available during the working days and hours.
- Agent Group #2: AfterHoursTeam - Available for non-regular working days and hours.

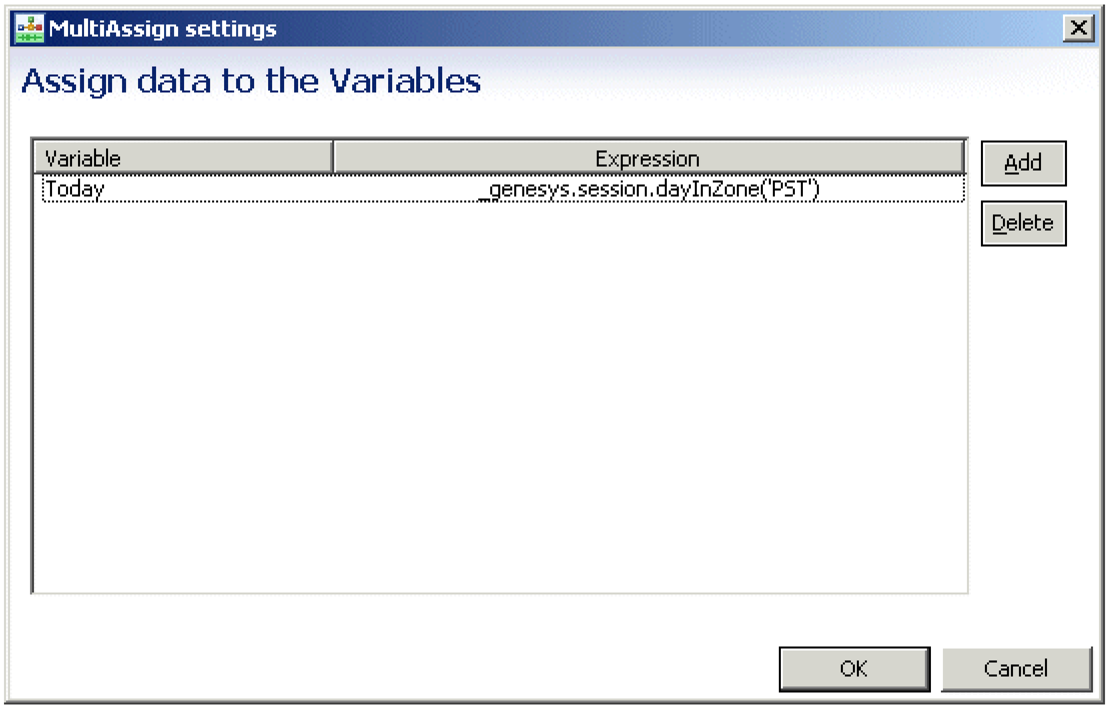
Previous Genesys Administrator (or Configuration Manager) setup involved defining Agent Groups and agents (Persons). The workflow diagram is shown below.



The above diagram is keyed to the numbers below.

1. The Entry block Variables property includes three user-defined variables. The Today and IsNineToFive variables are used for this application.

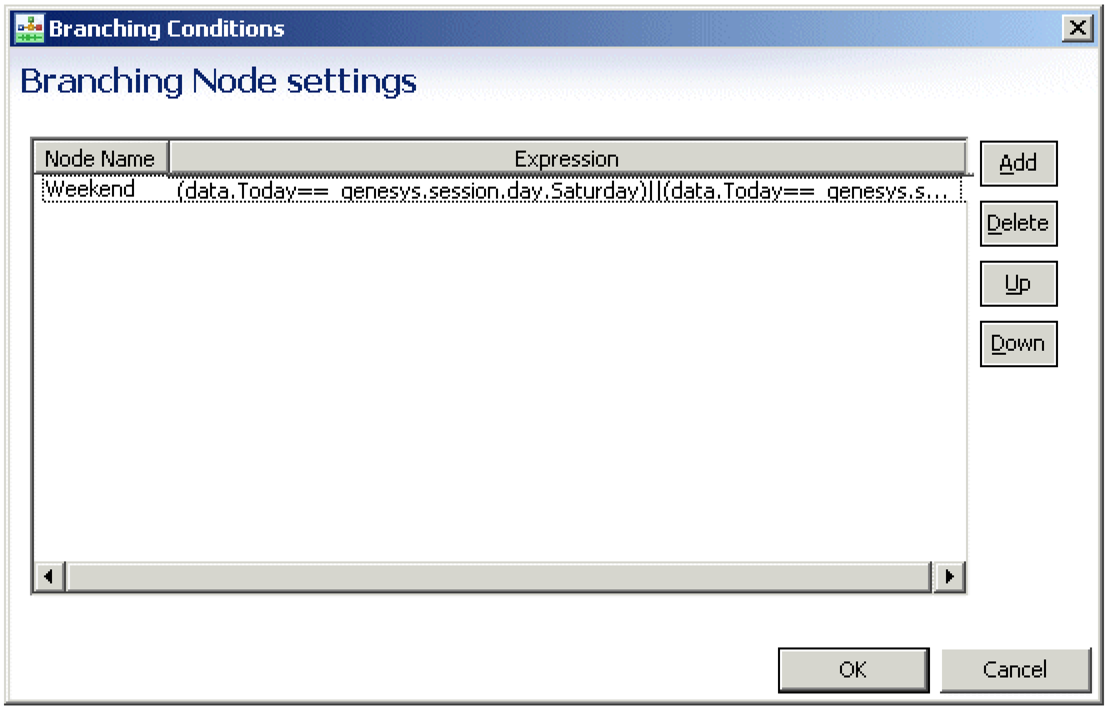
2. The Assign block Assign Data property assign a value to the Today variable. The function shown returns the current day of the week in the specified timezone.


MultiAssign settings

Assign data to the Variables

Variable	Expression
Today	_genesys.session.dayInZone('PST')

3. The Branching block Conditions property contains an expression used for segmenting interactions based on the date condition.


Branching Conditions

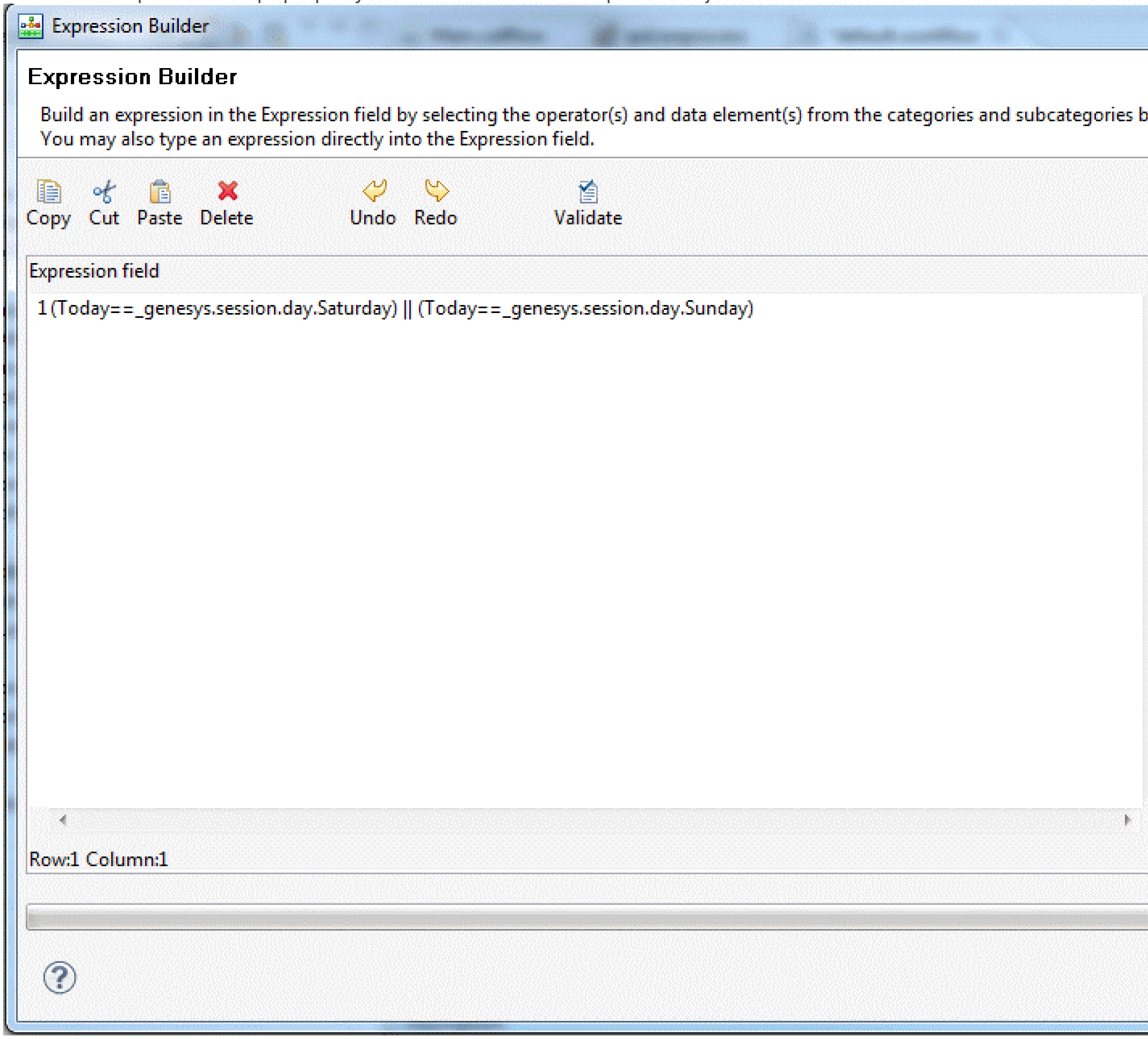
Branching Node settings

Node Name	Expression
Weekend	(data.Today== _genesys.session.day.Saturday)||(data.Today== _genesys.s...

You can open Expression Builder from the Condition property of the branching block and access Orchestration Server date/time functions (Data Category=Orchestration Server Functions > Data

Subcategory=genesys):

4. The ECMAScript block Script property uses standard ECMAScript time objects.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories b
You may also type an expression directly into the Expression field.

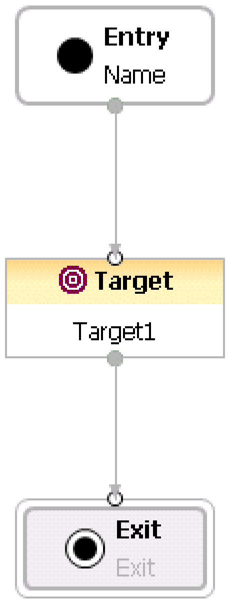
```
(Today==_genesys.session.day.Saturday) || (Today==_genesys.session.day.Sunday)
```

Row:1 Column:1

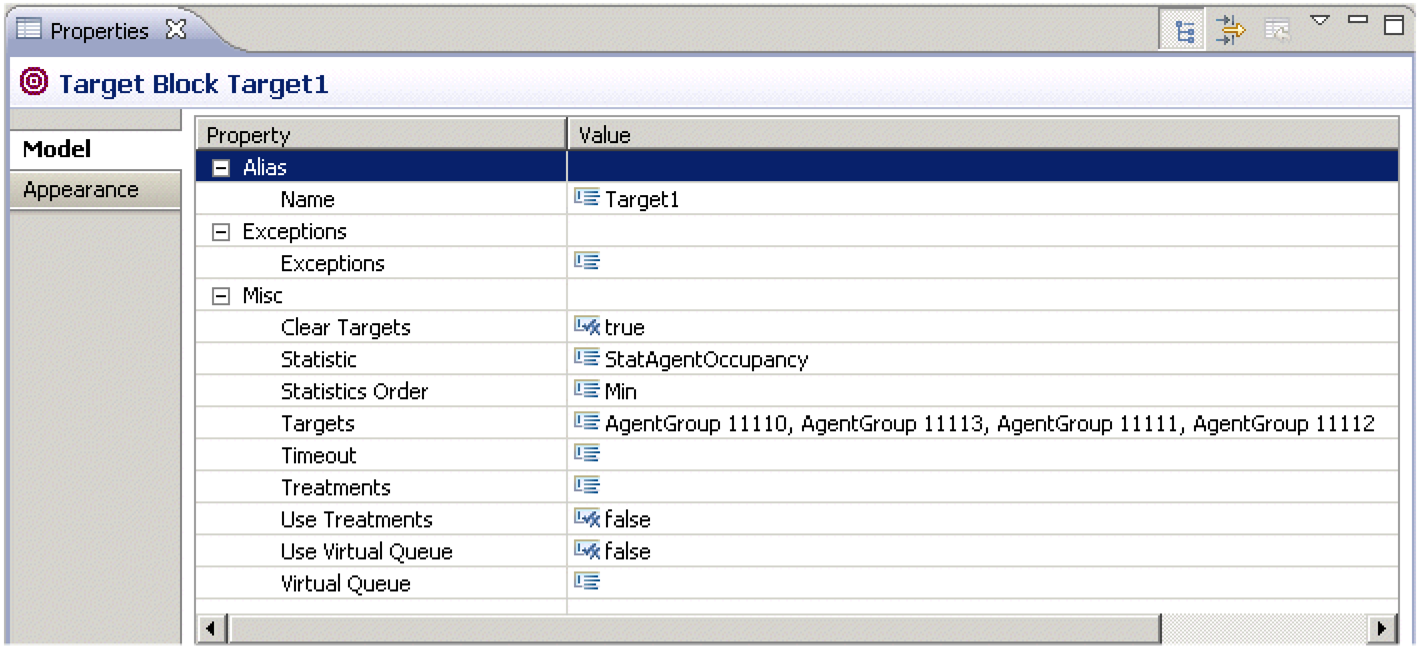
5. The Branching block Conditions property contains an expression, which determines whether the interaction goes to the CustomerServiceTeam or the AfterHoursTeam Agent Group.
6. The first Target block routes to the CustomerServerTeam.
7. The second Target block routes to the AfterHoursTeam.

Routing Based on a Statistic Sample

This application demonstrates routing based on the value of a statistic. For a definition of each statistic and recommended usage, consult the chapter on routing statistics in the *Universal Routing 8.1 Reference Manual.* The statistic you select is used by Universal Routing Server to determine which target to route the interaction to if more than one target is available.   After defining a complete set of available agents (taking agent capacity rules into consideration, if configured), URS applies the selection criteria specified in the Target block, which can include using the minimum or maximum value of the statistic (see Statistics Order property).   The workflow diagram is shown below.



The Target block Properties view is shown below.



Target Block Target1

Property	Value
⊟ Alias	
Name	Target1
⊟ Exceptions	
Exceptions	
⊟ Misc	
Clear Targets	true
Statistic	StatAgentOccupancy
Statistics Order	Min
Targets	AgentGroup 11110, AgentGroup 11113, AgentGroup 11111, AgentGroup 11112
Timeout	
Treatments	
Use Treatments	false
Use Virtual Queue	false
Virtual Queue	

The Misc properties that were configured are described below.

Clear Targets

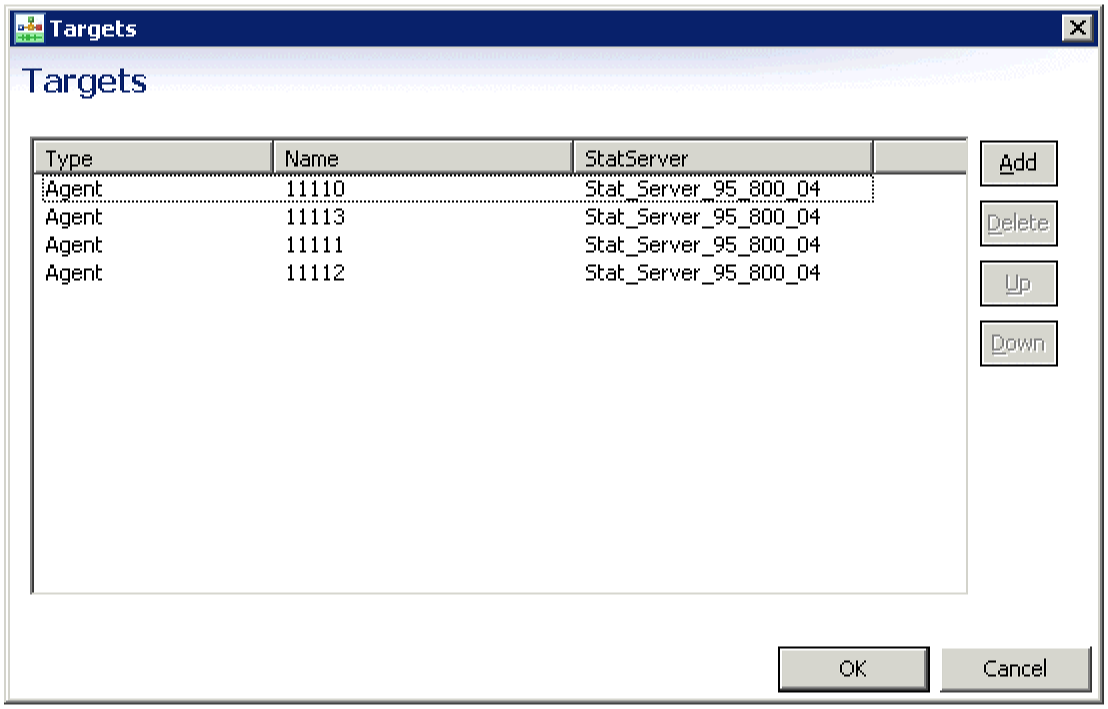
When this property is set to true, URS retains the targets listed in the block are after the interaction moves on through the strategy and encounters other Target blocks (not present in this simple sample). For more information on this property, see the Target Block Clear Targets Property. Statistic The selected statistic is StatAgentOccupancy. This statistic enables URS to route interactions to the least occupied agent, which is the agent with the lowest occupancy rate. Occupancy rate is the ratio between the time the agent has been busy since last login relative to the agent's total login time. StatAgentOccupancy enables URS to evaluate multiple available agents and select the least occupied agent so that the workload among available agents is balanced. For more information on this property, see the Target Block Statistic Property.

Statistics Order

This property can work with the Statistics property. Min was selected indicating the interaction should be routed to the target with the minimum value of the StatAgentOccupancy statistic. For more information on this property, see the Target Block Statistics Order Property.

Targets

The figure below shows the entries in the Targets dialog box.



Type	Name	StatServer
Agent	11110	Stat_Server_95_800_04
Agent	11113	Stat_Server_95_800_04
Agent	11111	Stat_Server_95_800_04
Agent	11112	Stat_Server_95_800_04

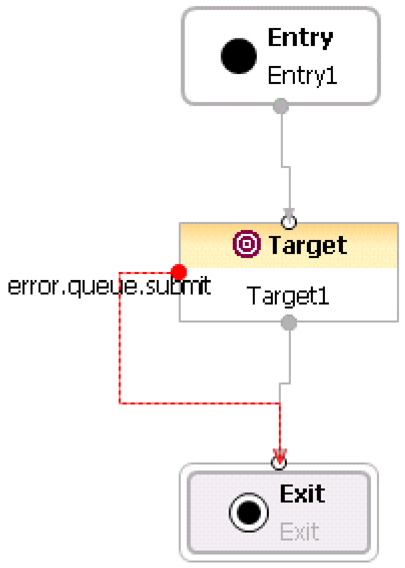
Using the StatAgentOccupancy statistic URS will select the least occupied agent among these based upon information from the specified Stat Server.

Timeout

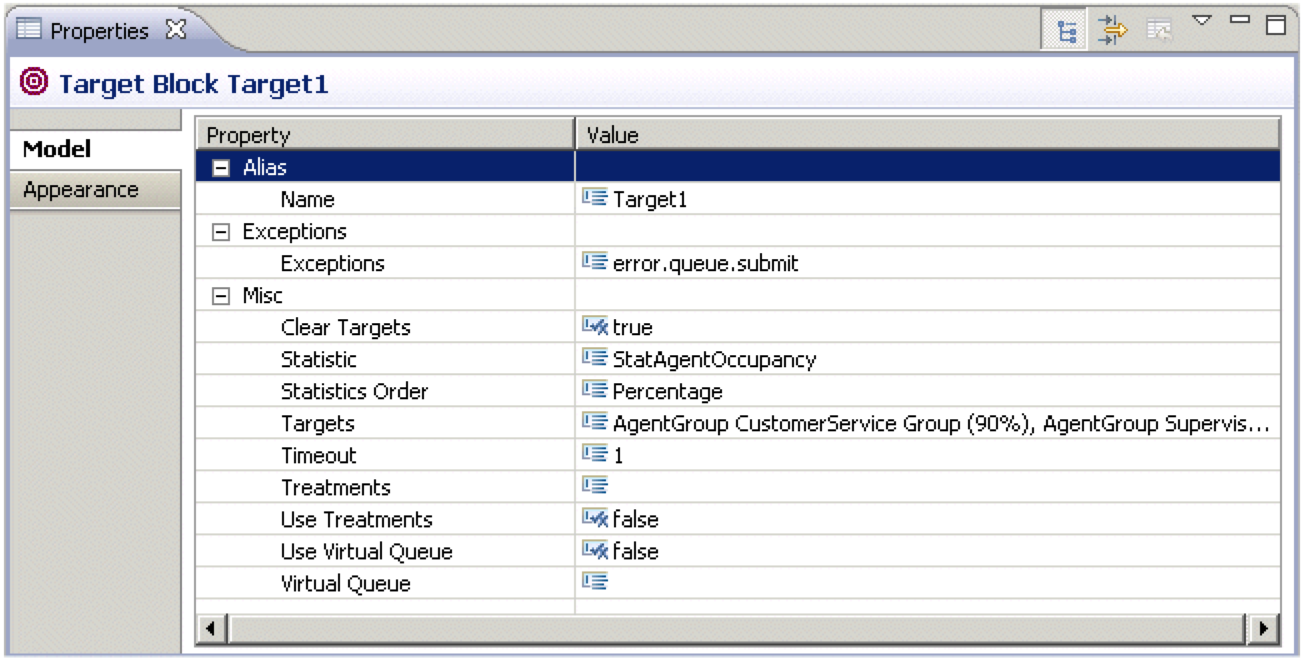
While not used in this sample, this property allows you to specify the time in seconds an interaction waits for an available target. If the timeout expires before one of the targets is available, the interaction is routed to the error port. For more information on this property, click Target Block Timeout Property. The remaining properties are left at their default values.

Routing Based on Percent Allocation

This application demonstrates distributing interactions to targets based on a set percentage for each target. The workflow diagram is shown below.



The Entry block does not contain any user-defined variables. The Target block Properties view is shown below.



Property	Value
⊟ Alias	
Name	Target1
⊟ Exceptions	
Exceptions	error.queue.submit
⊟ Misc	
Clear Targets	true
Statistic	StatAgentOccupancy
Statistics Order	Percentage
Targets	AgentGroup CustomerService Group (90%), AgentGroup Supervis...
Timeout	1
Treatments	
Use Treatments	false
Use Virtual Queue	false
Virtual Queue	

The Misc properties that were configured are described below.

Clear Targets

When this property is set to true, URS retains the targets listed in the block are after the interaction moves on through the strategy and encounters other Target blocks (not present in this simple sample). For more information on this property, see the Target Block Clear Targets Property.

Statistic

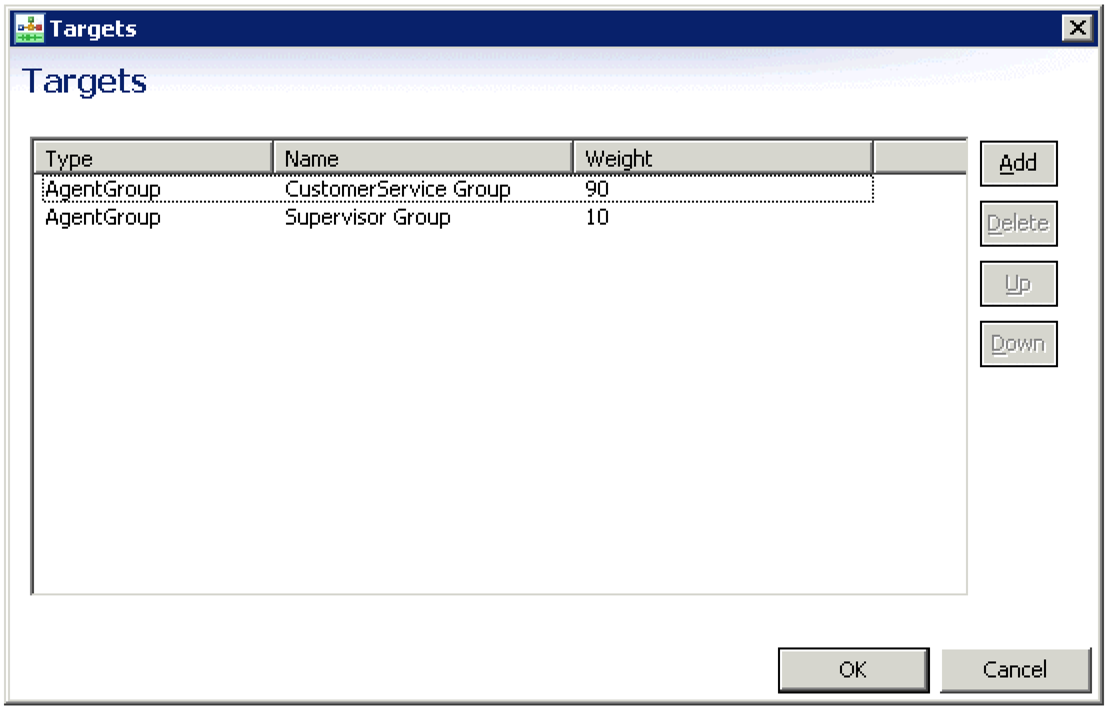
The selected statistic is StatAgentOccupancy. This statistic enables URS to route interactions to the least occupied agent, which is the agent with the lowest occupancy rate. Occupancy rate is the ratio between the time the agent has been busy since last login relative to the agent's total login time. StatAgentOccupancy enables URS to evaluate multiple available agents and select the least occupied agent so that the workload among available agents is balanced. For more information on this property, click Target Block Statistics Property.

Statistics Order

Percentage is selected in order to route interactions to targets based on a percentage allocation. Selecting Percentage causes the dialog box that opens from the Targets property to display a Weight column where you can specify a percentage for each target. For more information on this property, see the Target Block Statistics Order Property.

Targets

The figure below shows the entries in the Targets dialog box.



Targets

Type	Name	Weight
AgentGroup	CustomerService Group	90
AgentGroup	Supervisor Group	10

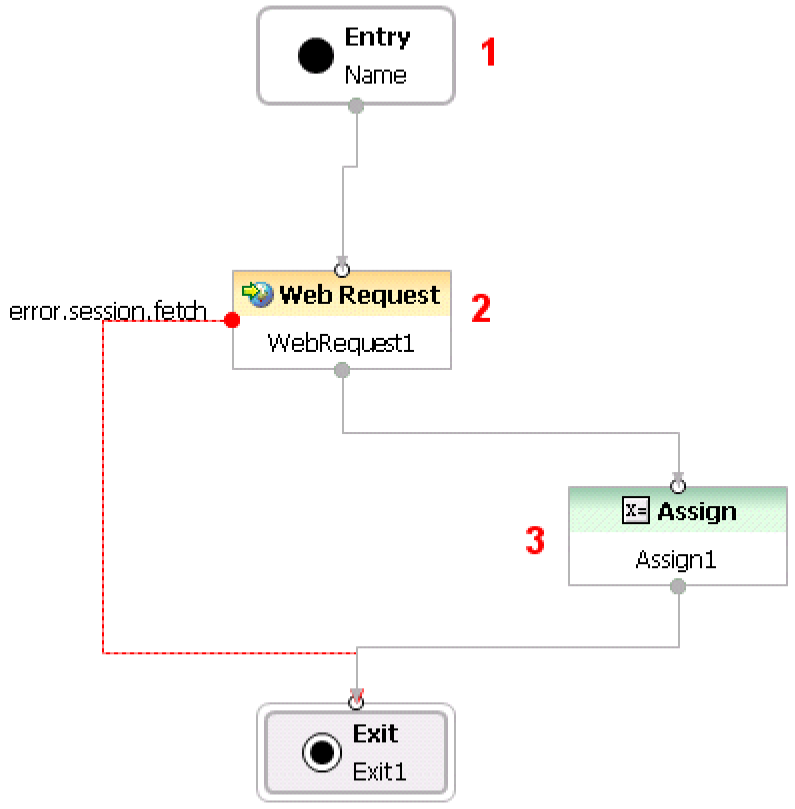
URS will route 90% of the calls to the CustomerService Group and 10% of the calls to the Supervisor Group. Using the StatAgentOccupancy statistic URS will select the least occupied agent among these based upon information from the specified Stat Server.

Timeout

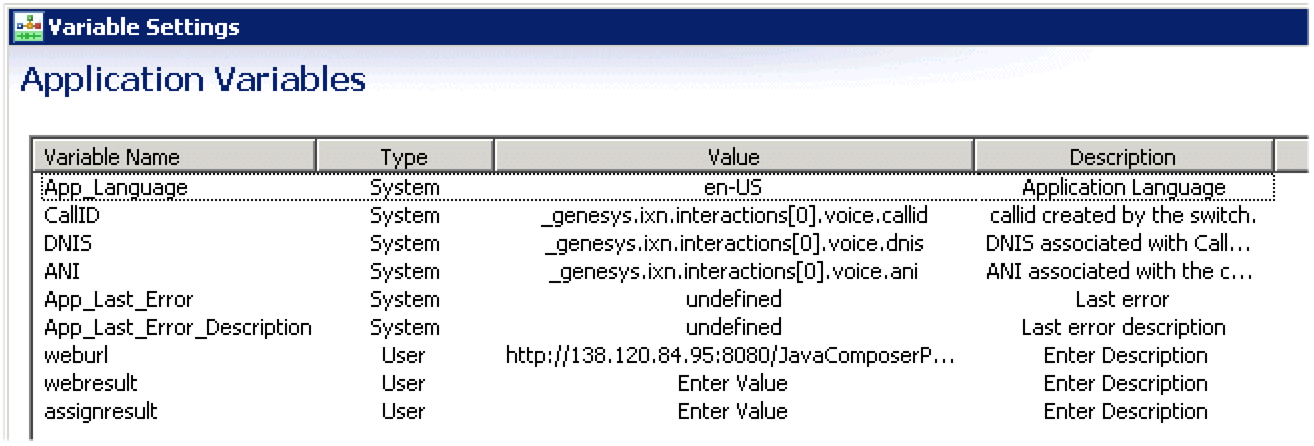
This property allows you to specify the time in seconds an interaction waits for an available target. If the timeout expires before one of the targets is available, the interaction is routed to the error port. For more information on this property, click Target Block Timeout Property. The remaining properties are left at their default values.

Routing Using Web Request Sample

This application demonstrates using the Web Request block to invoke an HTTP web request. The workflow diagram is shown below.



The above diagram is keyed to the numbers below. The Entry block Variable Settings dialog box is shown below.



Variable Settings

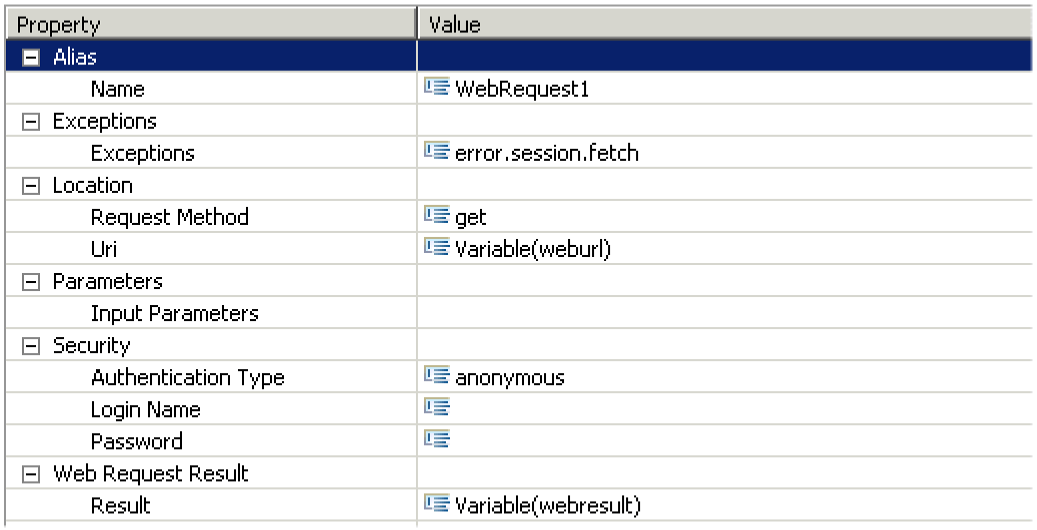
Application Variables

Variable Name	Type	Value	Description
App_Language	System	en-US	Application Language
CallID	System	_genesys.ixn.interactions[0].voice.callid	callid created by the switch.
DNIS	System	_genesys.ixn.interactions[0].voice.dnis	DNIS associated with Call...
ANI	System	_genesys.ixn.interactions[0].voice.ani	ANI associated with the c...
App_Last_Error	System	undefined	Last error
App_Last_Error_Description	System	undefined	Last error description
weburl	User	http://138.120.84.95:8080/JavaComposerP...	Enter Description
webresult	User	Enter Value	Enter Description
assignresult	User	Enter Value	Enter Description

Three user-defined variables are defined:

- weburl: Points to the getMemoryStatus.jsp application from the src dir. This represents an application hosted on an external application server.
- webresult: Holds the result of the get request.
- assignresult: The content of the webresult variable is transferred to this variable in the Assign block.

The Web Request block Properties view is shown below.

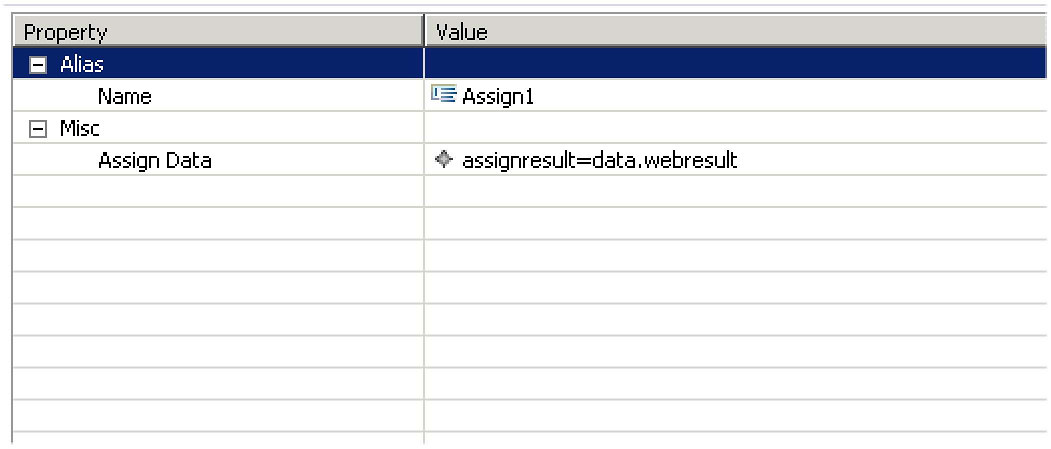


Property	Value
Alias	
Name	WebRequest1
Exceptions	
Exceptions	error.session.fetch
Location	
Request Method	get
Uri	Variable(weburl)
Parameters	
Input Parameters	
Security	
Authentication Type	anonymous
Login Name	
Password	
Web Request Result	
Result	Variable(webresult)

- The Exceptions property indicates support for the error.sesson.fetch event.
- The Request Method property has two choices: get or post with get selected indicating the type of request (HTTP Get) that Universal Routing Server will make.
- The Uri property indicates that the Uri is contained in a variable called weburl.
- The Authentication Type property has two choices: anonymous or basic with anonymous selected as a security setting. With this type of access, no user name/password is passed to the Web service for client authentication in order to get data.

- The Result property indicates that the data from the get request will initially be stored in a variable called webresult.

The Assign block properties view is shown below.



Property	Value
⊟ Alias	
Name	Assign1
⊟ Misc	
Assign Data	assignresult=data.webresult

The data received from the get request is assigned to a variable called assignresult.

Last Called Agent Routing

This topic summarizes how to use Composer to create a routing strategy that gives the calling customer the option to speak to the agent who handled their previous call or to wait for a callback from that agent.

Last Called Agent (LCA) Routing

You can use Composer to create an Integrated Voice and Route application that routes a customer call to the agent who last spoke with that customer (last called agent or LCA routing). The sample below summarizes one way to configure LCA routing. In this sample, an Interaction Workspace custom extension retrieves the last called agent information from the Universal Contact Server (UCS) Database.

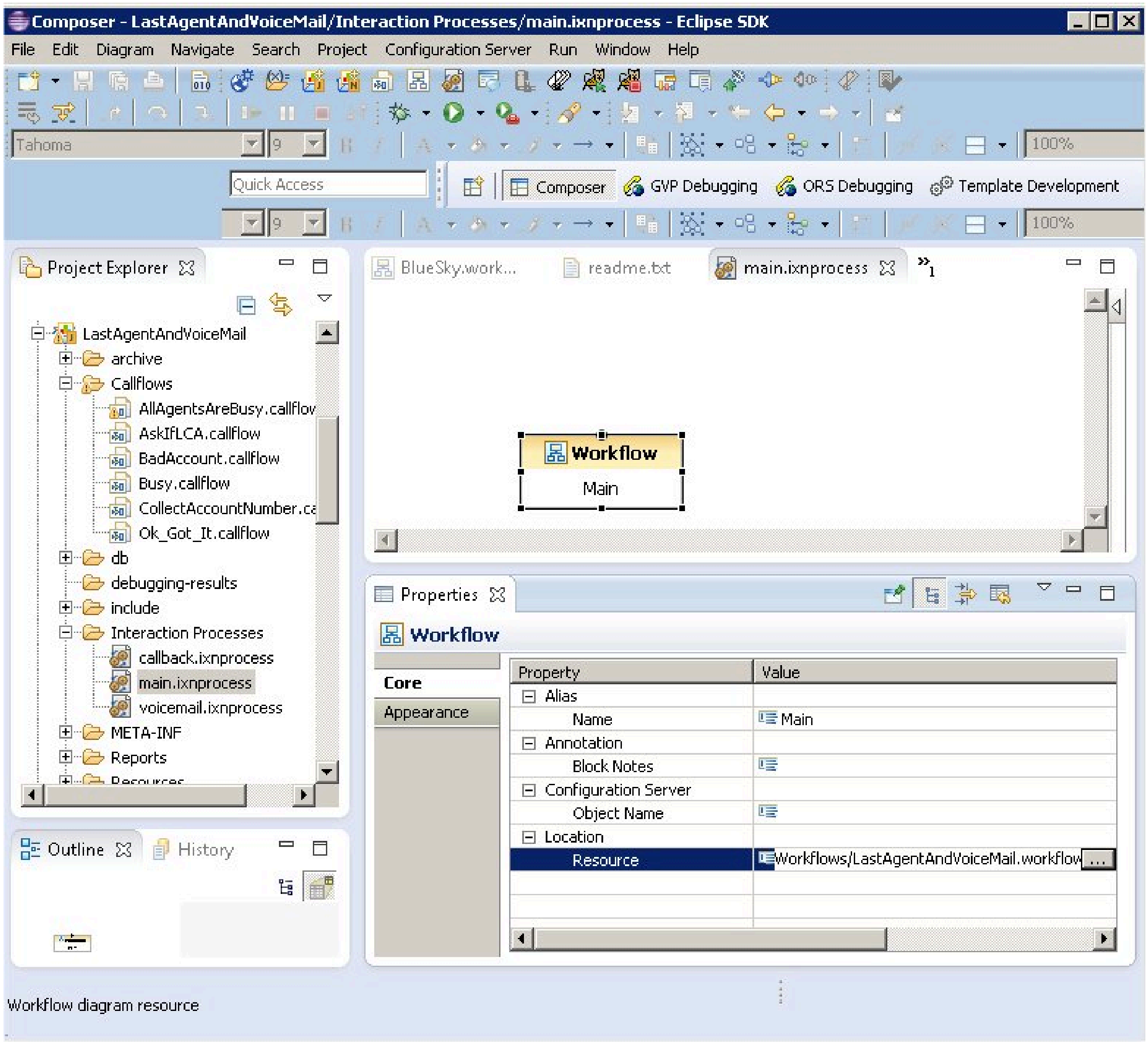
Note: You can set the Contact Server Interaction Workspace option contact.last-called-agent.enable to save the last called agent ID to the Universal Contact Server database. To use the agent ID in a routing workflow, you can use Expression Builder to create a custom extension (such as that shown in the sample below) that retrieves the agent ID from the UCS Database. Interaction Workspace installed "out-of-the-box" does not supply the extension used in the sample below.

Scenario Summary

- A customer calls the contact center and the call is routed to an agent.
- The agent ID and the call time is saved to the customer contact information in the UCS database. As a result, a future call from the customer can be routed to the same agent.
- If the customer calls again and requests the last called agent, the call is routed to that agent. If that agent is unavailable, the voice treatment offers three options: Request a callback, Wait, or Leave a voicemail.

Interaction Process Diagram

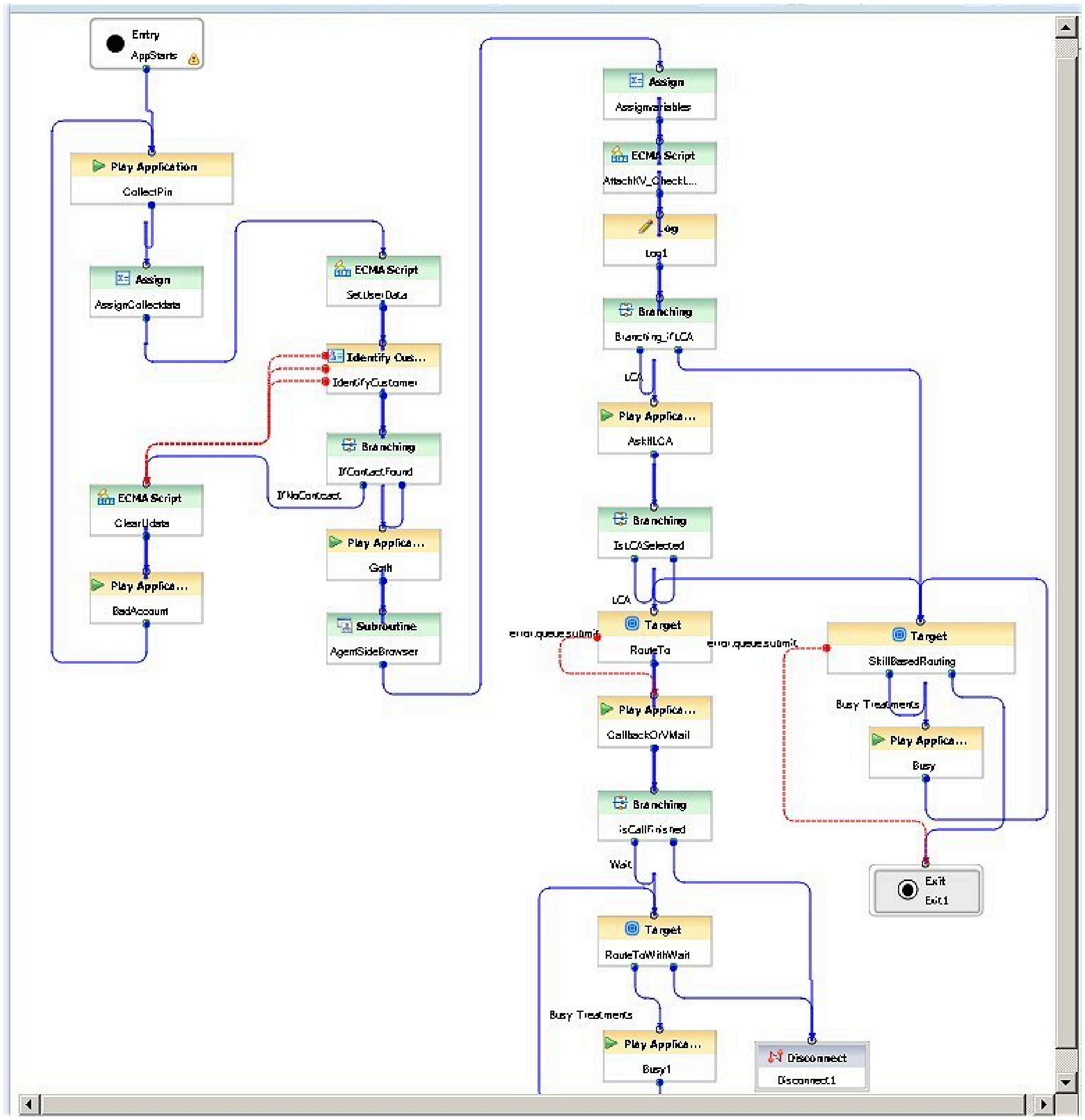
All Composer routing applications start with an interaction process diagram (IPD). For a routing application used for voice only interactions, an IPD Workflow block points to a workflow resource (diagram). In the Workflow block shown in the figure below, the resource is the `LastAgentAndVoiceMail.workflow` diagram.



Property	Value
Alias	
Name	Main
Annotation	
Block Notes	
Configuration Server	
Object Name	
Location	
Resource	Workflows/LastAgentAndVoiceMail.workflow

LastAgentAndVoiceMail.workflow

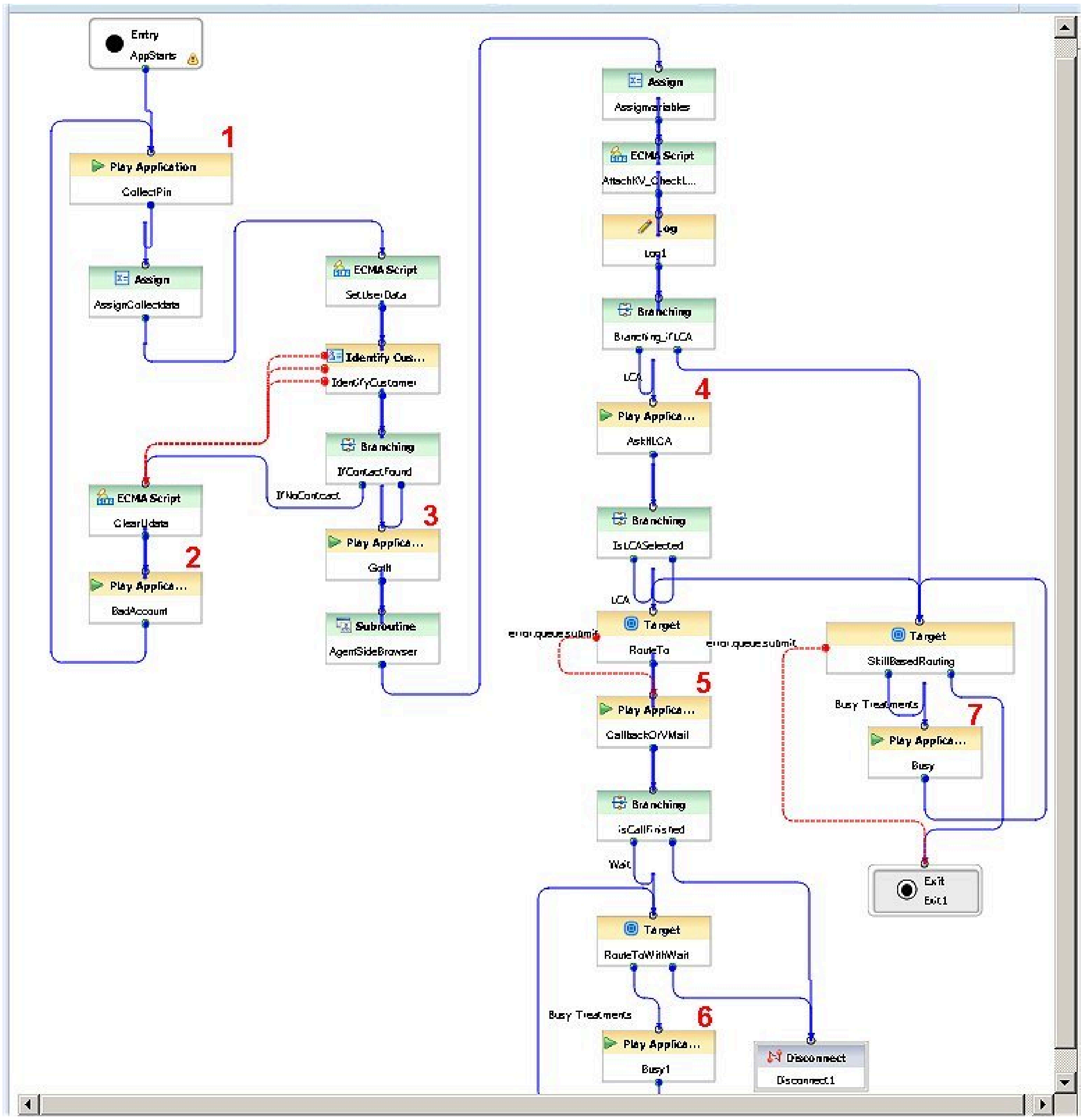
The sample workflow diagram that demonstrates LCA routing is shown below.



Each block in the workflow diagram is labeled with text that summarizes its function. This text appears below the block name. The next section discusses key blocks in the diagram.

Play Application Blocks

The above workflow diagram contain various Play Application blocks. These are numbered in the figure below.

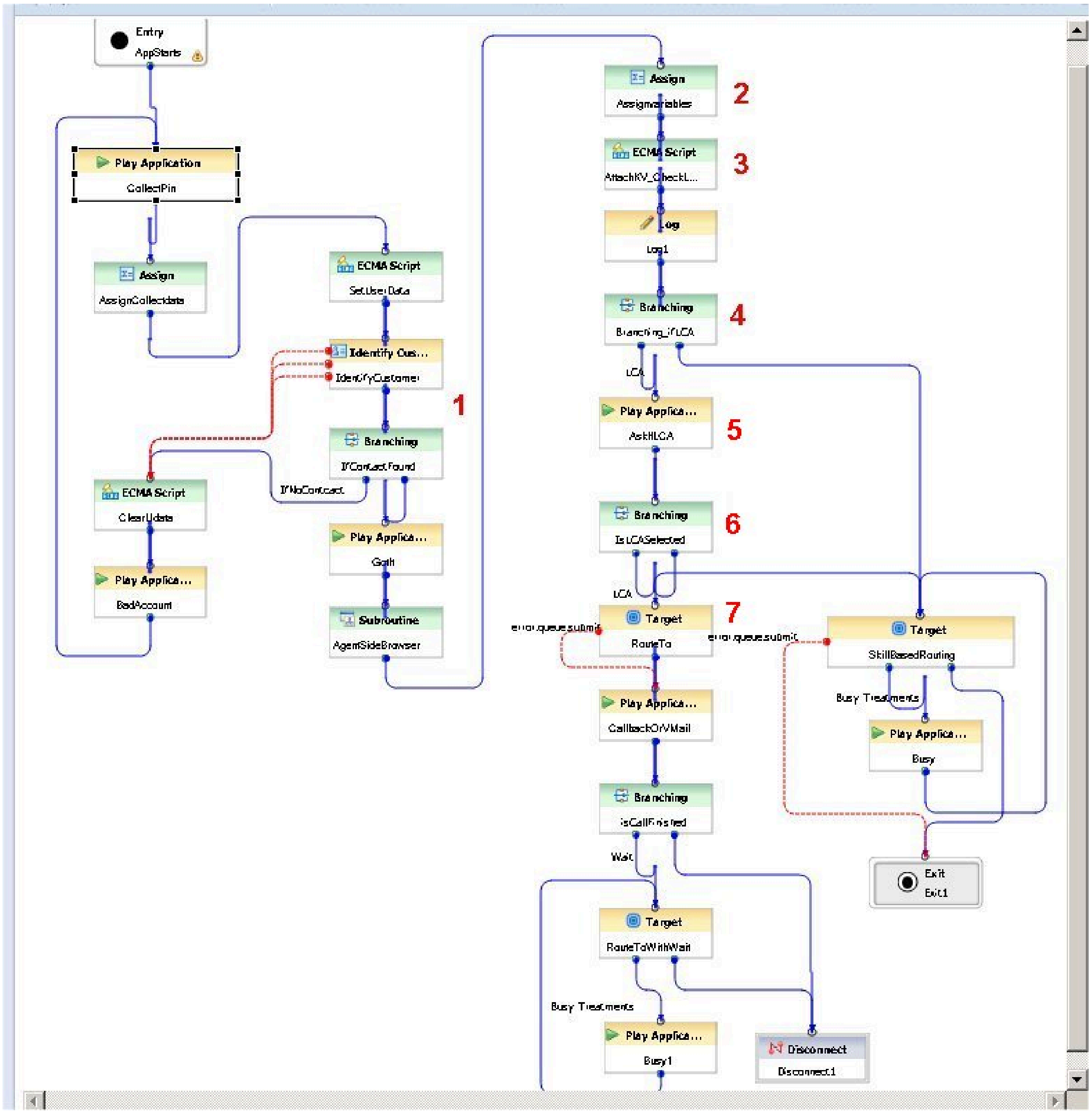


Reviewing the various voice prompts associated with these Play Application blocks helps you understand the flow.

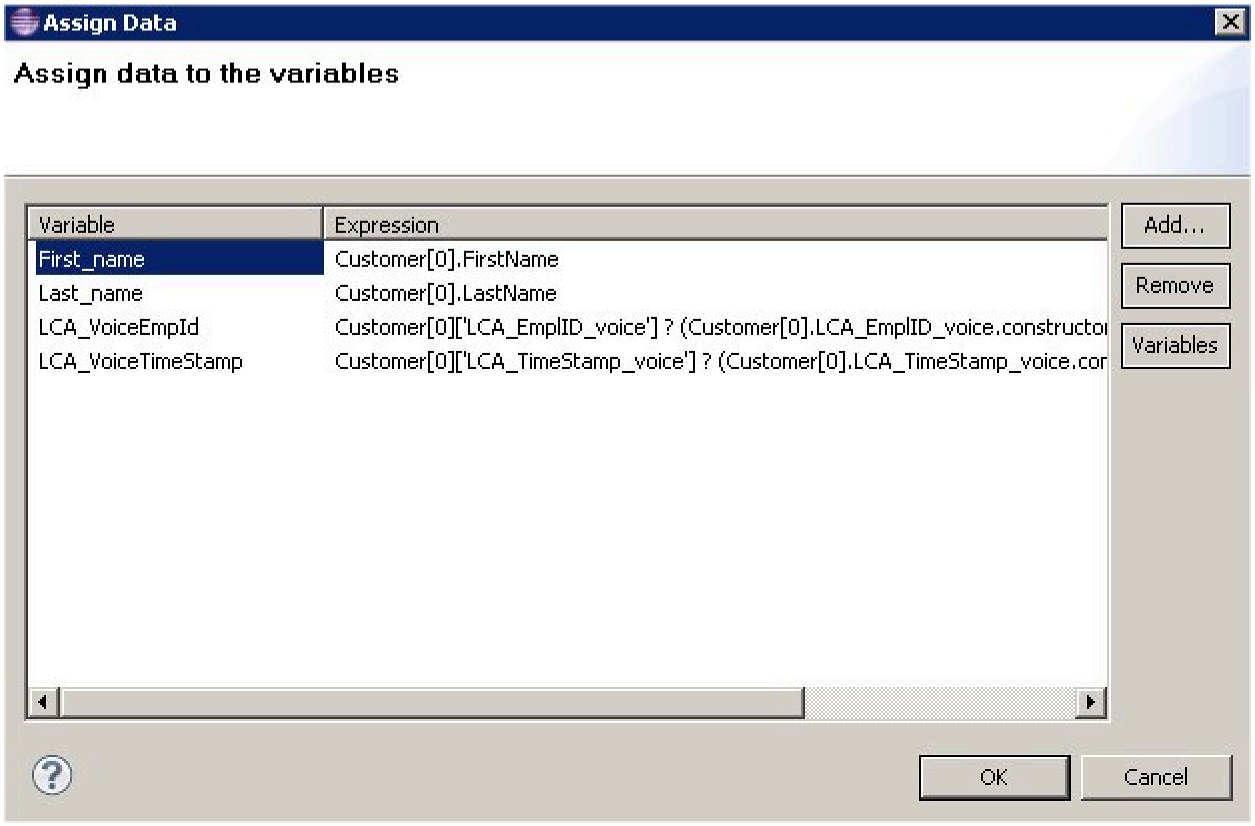
```
 last_agent_001.vox        Welcome.  Please enter your four digit account number
 last_agent_002.vox        Ok, got it.
 last_agent_003.vox        Please hold while I transfer your call to an agent ....
 last_agent_004.vox        Sorry, all of our agents are currently busy.  Please hold for the
next available agent.
 last_agent_005.vox        Would you like to be connected to the agent that you spoke with
previously?
                        For yes, press 1.  For no, press 2.
 last_agent_006.vox        Ok, checking agent availability
 last_agent_007.vox        Sorry, your agent is not available at the moment.  To have the
agent call you back,
                        press 1.  To leave a voicemail for the agent, press 2.
                        Or just wait on the line for the agent.
 last_agent_008.vox        Thank you, your callback request has been registered.  The agent
will call you
                        back soon. Goodbye.
 last_agent_009.vox        Ok, to leave a voicemail for the agent, being speaking at the
tone.
                        When you are finished, you can simply hang up or press pound.
 last_agent_010.vox        Your voicemail has been recorded and will be delivered to the
agent.
                        Thank you for calling, goodbye.
 last_agent_011.vox        Sorry, I didn't get that.
```

Blocks Used for LCA

This section summarizes the important blocks used for LCA routing in this particular sample. The figure below numbers the LCA-related block with the numbers keyed to descriptions further ahead.

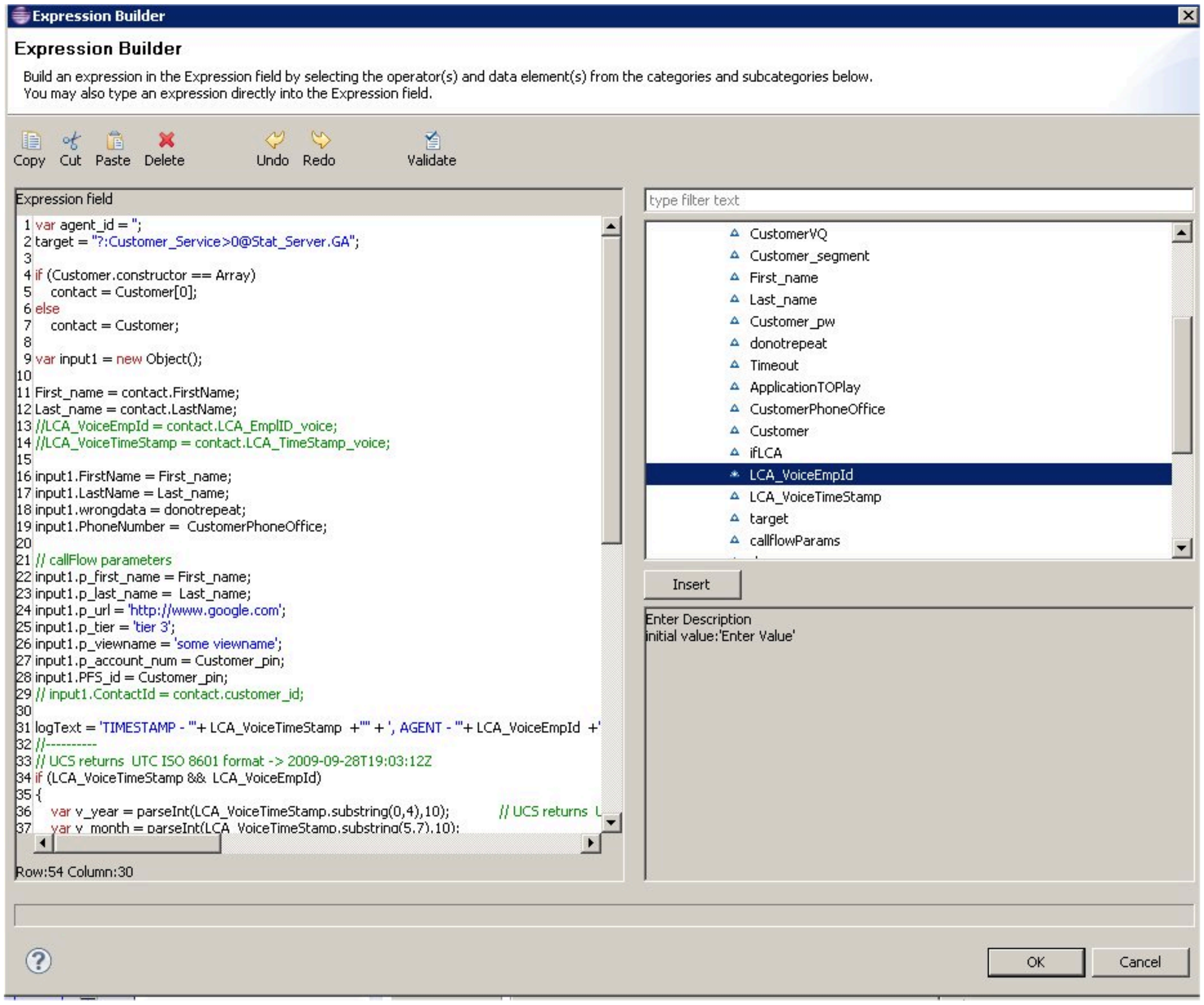


1. The Identify Customer block (**1**) uses the account numbered entered by the customer to search for contact information in the Universal Contact Server Database. A Branching block (1) causes the workflow to take different paths based on whether customer data is or is not found.
2. If found, an Assign block (**2**) assigns LCA information to variables.



Variable	Expression
First_name	Customer[0].FirstName
Last_name	Customer[0].LastName
LCA_VoiceEmpId	Customer[0]['LCA_EmplID_voice'] ? (Customer[0].LCA_EmplID_voice.constructo
LCA_VoiceTimeStamp	Customer[0]['LCA_TimeStamp_voice'] ? (Customer[0].LCA_TimeStamp_voice.cor

3. The ECMAScript block (**3**) calls a script created in Expression Builder to get the last called agent ID.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.
You may also type an expression directly into the Expression field.

```
var agent_id = '';
target = "?:Customer_Service>0@Stat_Server.GA";

if (Customer.constructor == Array)
   contact = Customer[0];
else
   contact = Customer;

var input1 = new Object();

First_name = contact.FirstName;
Last_name = contact.LastName;
//LCA_VoiceEmpId = contact.LCA_EmplID_voice;
//LCA_VoiceTimeStamp = contact.LCA_TimeStamp_voice;

input1.FirstName = First_name;
input1.LastName = Last_name;
input1.wrongdata = donotrepeat;
input1.PhoneNumber =  CustomerPhoneOffice;

// callFlow parameters
input1.p_first_name = First_name;
input1.p_last_name =  Last_name;
input1.p_url = 'http://www.google.com';
input1.p_tier = 'tier 3';
input1.p_viewname = 'some viewname';
input1.p_account_num = Customer_pin;
input1.PFS_id = Customer_pin;
// input1.ContactId = contact.customer_id;

logText = 'TIMESTAMP - "'+ LCA_VoiceTimeStamp  +'"' + ', AGENT - "'+ LCA_VoiceEmpId  +'
//----------
// UCS returns  UTC ISO 8601 format -> 2009-09-28T19:03:12Z
if (LCA_VoiceTimeStamp &&  LCA_VoiceEmpId)
{
   var v_year = parseInt(LCA_VoiceTimeStamp.substring(0,4),10);          // UCS returns  U
   var v_month = parseInt(LCA_VoiceTimeStamp.substring(5,7),10);
```

Row:54 Column:30

- CustomerVQ
- Customer_segment
- First_name
- Last_name
- Customer_pw
- donotrepeat
- Timeout
- ApplicationTOPlay
- CustomerPhoneOffice
- Customer
- ifLCA
- LCA_VoiceEmpId
- LCA_VoiceTimeStamp
- target
- callflowParams

Enter Description
initial value:'Enter Value'

4. If the last called agent is found, a Branching block (**4**) sends the interaction to another Play Application block.
5. This Play Application block (**5**) prompts the customer: "Would you like to be connected to the agent that you spoke with previously? For yes, press 1. For no, press 2."
6. If the customer presses 1, another Branching block (**6**) directs the interaction to a Target block (**7**), which routes the call to the last called agent. If the agent is unavailable, the customer is given the option of a callback or waiting for the agent.

For more information on this particular sample and last agent routing in general, please contact Genesys Technical Marketing.

Validation Debugging Deployment

This section contains information about the following:

- Validating your diagram files and other source files for completeness and accuracy
- Debugging voice applications
- Debugging routing applications
- Deploying Applications to a web server

Validation

Composer can validate your diagram files and other source files for completeness and accuracy.

Prompts Resource Validation

In Diagram Preferences, Global Settings, the Enable Validation for Prompt Resource preference enables diagram validation warnings where prompt audio resources no longer exist in the given file path. If the audio file is no longer present, the diagram block will show a warning icon.

Diagram Validation

This topic covers both callflow/workflow diagrams and interaction process diagrams (IPDs).

Callflow/Workfow Validation

You can initiate standalone callflow or workflow validation in a couple of ways. When the callflow or workflow is saved and selected:

- **Diagram** > **Validate** from the menu.
- Click the Validate icon ✓ on the upper-right of the Composer main window .

Note: In case of errors, the Problems view will become visible and error markers are put on the callflow or workflow blocks that contain errors. Double clicking on an error in the Problems view will take you to the corresponding blocks that contain the errors. Review each of the errors and do the fixes, then validate again. After validation, you can generate code.

Interaction Process Diagram Validation

The validation process is basically the same for IPDs. When the IPD is selected or in view :

- **Diagram** > **Validate** from the menu.
- Click the Validate icon ✓ on the upper-right of the Composer main window.

When invoked, validation checks for the existence of objects in Configuration Server and indicates the results. Validation does not make changes in Configuration Server as part of the process. Note: If Composer is not connected to Configuration Server, clicking the validation button brings up the

Connect to Configuration Server dialog.

Source File Validation

There are two types of validation that can occur when you are working with source files in Composer Rich editors:

- Source validation
- Batch validation

Source validation occurs as you type your code; this validation reflects the "unsaved" and "unbuilt" contents of the source you are editing. Note: To turn source validation on (or off) for all structured text editors, click **Window** > **Preferences** > **General**> **Editors**> **Structured Text Editors** and check (or uncheck) **Report problems as you type**. Batch validation occurs on saved files. Batch validations may catch build process errors and other types of non-source validation errors. Batch validation can uncover errors in multiple files at once and give you a comprehensive view of where problematic code can be found in your project. Moreover, you do not need to open files in an editor to run batch validation. To run batch validation on specific files, select and right click the files in the Project Explorer and then select Run Validation from the popup menu. Note: To set preferences for batch validation, click **Window** > **Preferences** > **Team** > **Validation**.

Project Level Validation

You can validate a Project using the integrated Composer Validator, which is enabled or disabled at global level using the Validation preferences (**Window** > **Preferences** > **Team** > **Validation** > **Composer** > **Composer Validator**).

- Invoke Composer Validator on a Project or individual file by right clicking and selecting **Validate** from the pop-up menu.

Note: Automatic validation does not occur during any change in resource contents, since this action is taken care by the Composer Project Builders. You can enable or disable Composer Project Builders using the **Project** > **Build Automatically** menu option.

Project Level Validation Use Cases

Here are some possible validation use case scenarios:

- Validate all the callflow and workflow diagram files within the Composer Project.
- Validate all the referenced subroutines and subdialog diagram files, which have the corresponding .scxml / .vxml files generated inside the src-gen folder.
- Validate all the GBuilder files, which have the corresponding .grxml file generated within the Grammars folder.
- Validate all ASP.NET related resources within a .NET Composer Project.
- Validate all JSP related resources within a Java Composer Project.

Validating a Single Flow Diagram

To configure the contents of the Problems view to view only warnings and errors associated with the currently validated callflow/workflow:

1. Open the Problems View by selecting **Window** > **Show View** > **Problems**.
2. On the top right side of the Problems view, click the down arrow to display the View menu.
3. Select **Configure Contents**.
4. On the left side of the Configure Contents dialog box under Configurations, select **Errors/Warnings on Selection**.
5. Under **Scope**, click **On selected element and its children**.
6. Click OK.

For more information on this dialog box:

1. Select **Help** > **Contents**.
2. Open the *Workbench User Guide*.
3. Expand **Concepts** > **View**s > **Problems**.
4. Review the Problems View topic.

Debugging Routing Applications

Composer's ORS_Debugger provides real-time debugging capabilities for SCXML-based Orchestration Server (ORS) applications. The ORS Debugger is integrated within the workflow designer for making test calls, creating breakpoints, viewing call traces, stepping through an SCXML document/workflow, and debugging applications. Debugging can be started on an existing session or it can wait for the next session that runs the application at a given URL. Prior to debugging, set preferences for the ORS Debugger, which supports both Run and Debug modes.

- Using a **Run As** > **Run Configurations** launch configuration, metrics (call traces) are displayed and the application continues without stopping at any breakpoints. When the SCXML application executes, these metrics can describe, for example, state transitions, ECMAScript executions, and execution warnings or errors.
- Using a Debug launch configuration, debugging pauses at breakpoints, single-step through the code, inspect variable and property values, and execute any ECMAScript from the query console.

You can debug:

- A workflow built with Composer, or
- Any SCXML application or set of SCXML pages whether or not they were created with Composer.

Notes:

- You can export a launch configuration. From the File menu, select Export. Expand Run/Debug and select **Launch Configurations** > **Next.** The dialog box lets you select or browse for a launch configuration.
- Composer 8.1 uses TCP to send SIP messages (previous releases used UDP). This is not a configurable option.
- Also see ORS Debugger Limitations.

Starting a Debugging Session

If using Context Services:

- Set Context Services parameters: **Window** > **Preferences** > **Composer** > **Context Services**. In Context Services Preferences, specify the Universal Contact Server host and port.

In order to debug, a launch configuration must exist. There are various ways to create a launch configuration:

- Right-click on the diagram/SCXML file in the Project Explorer. Select **Run As** > **Run Configuration**s or **Debug As** > **Debug Configurations**. (The difference between Run as and Debug as is explained at the start of this topic.) This opens a dialog box for creating a launch configuration for the workflow diagram or SCXML files.
- Use launch shortcuts. Note: To automatically fill in a debug or run launch configuration, use launch shortcuts. Right-click on the diagram/file and select **Debug As Workflow** or **Debug as SCXML File**.

Composer automatically fills in the launch configuration.

- On the main toolbar, there is a Debug button and a Run button. Clicking relaunches the most recently used launch configuration. You can also use the keyboard shortcuts **Ctrl+F11** (for Run) and **F11** (for Debug).
- If you click the down arrow on these buttons to drop down a menu, a history of recent launches appears.

All of the above are also available in the Run top-level menu.

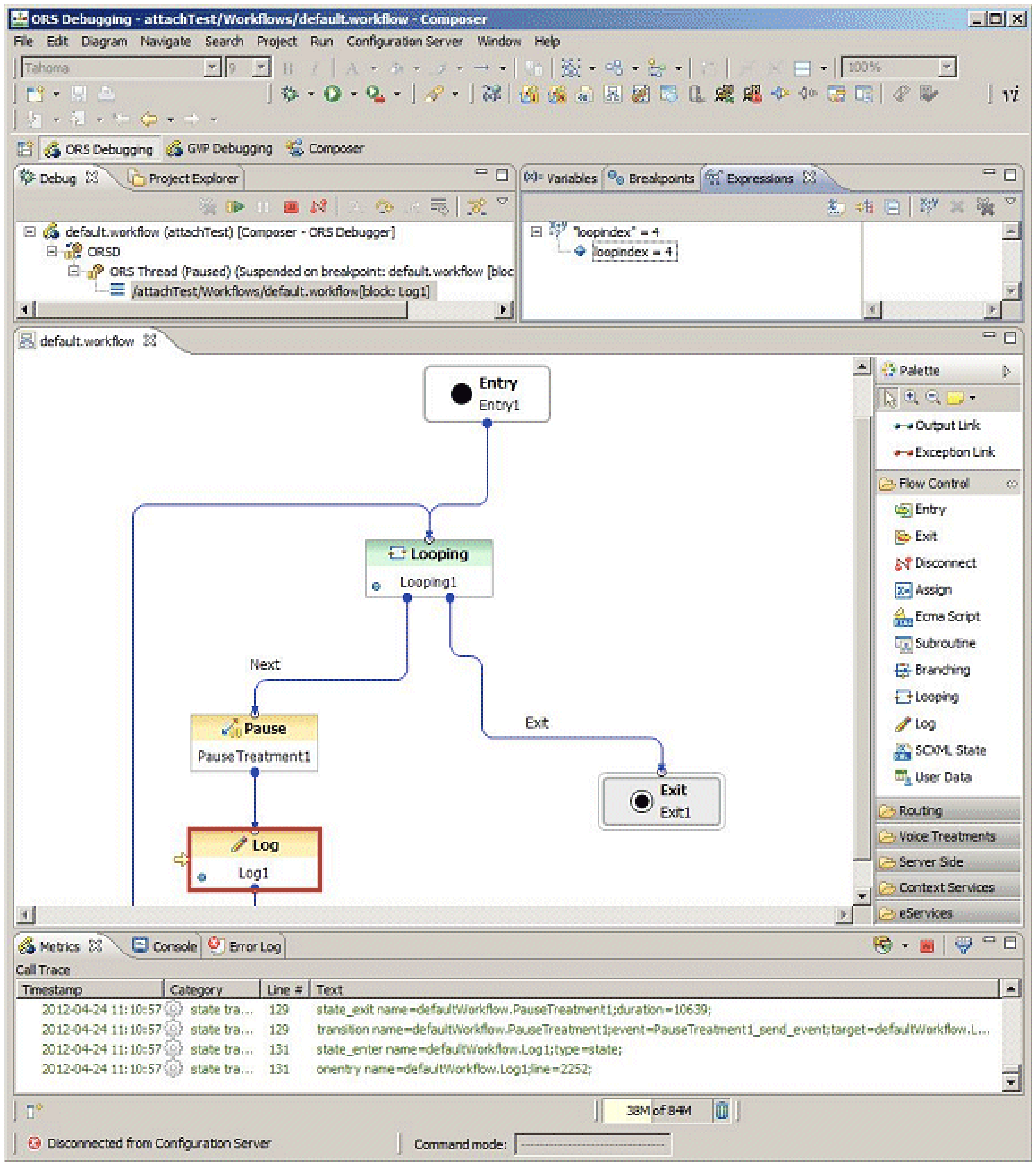
ORS Debugger Limitations

Limitations of the ORS Debugger are as follows:

- Interaction process diagram debugging is not supported. Code generated from an IPD can be debugged just like any other SCXML page.
- In SCXML debugging mode, the <invoke> tag will not step into the invoked SCXML page. Debugging will continue to the next element in the page currently being debugged.
- Debugging a Play Application block will not step into the associated Callflow diagram and will not launch a GVP debugging sessions. Instead debugging will continue on to the block after the Play Application block.
- Application variables are not displayed correctly in the Variables View in ORS Debugging Perspective if the value contains XML or variables that are of type E4X.

ORS Debugging Perspective

See Debugging Toolbars for information on the views and buttons.



Call Trace

Timestamp	Category	Line #	Text
2012-04-24 11:10:57	state tra...	129	state_exit name=defaultWorkflow.PauseTreatment1;duration=10639;
2012-04-24 11:10:57	state tra...	129	transition name=defaultWorkflow.PauseTreatment1;event=PauseTreatment1_send_event;target=defaultWorkflow.L...
2012-04-24 11:10:57	state tra...	131	state_enter name=defaultWorkflow.Log1;type=state;
2012-04-24 11:10:57	state tra...	131	onentry name=defaultWorkflow.Log1;line=2252;

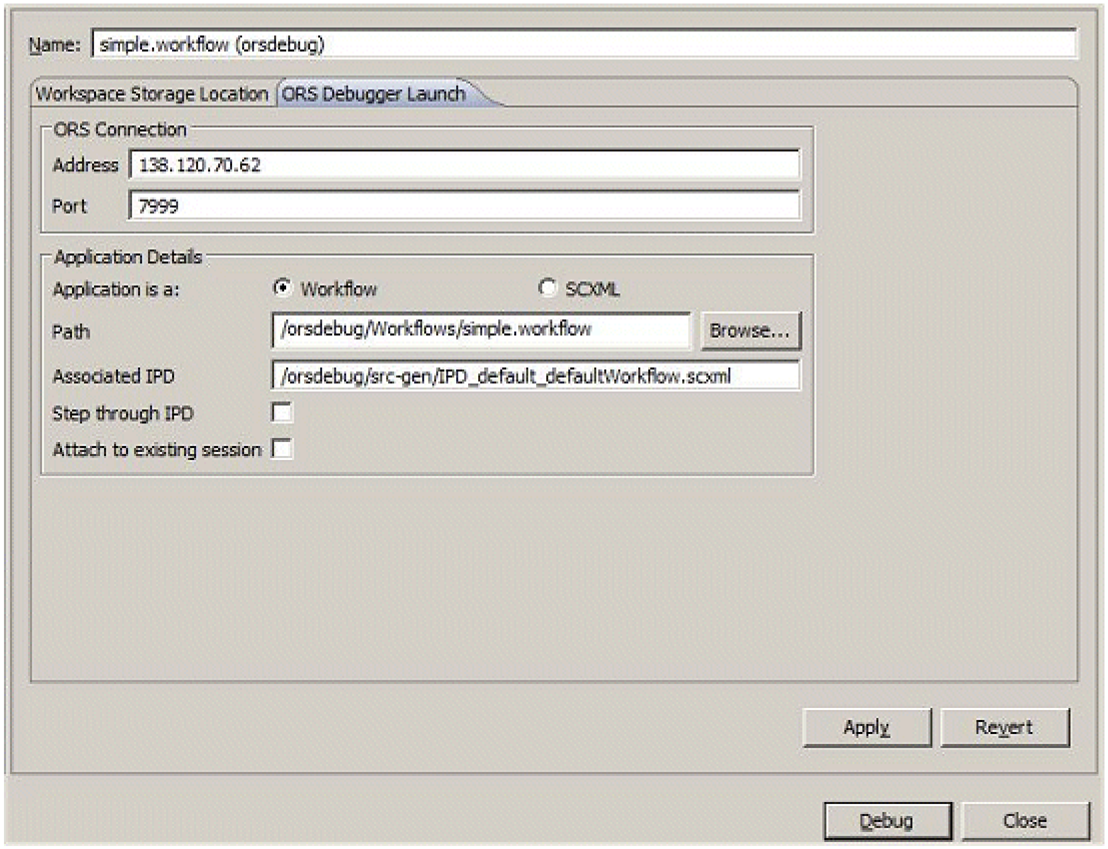
Debugging a Workflow Diagram

Workflow diagrams can be tested using the real-time ORS Debugger. Both Run and Debug launch configurations are supported, as well as code and diagram modes.

- In the Run mode, call traces are displayed and the workflow continues without stopping at any breakpoints.

- In the Debug mode, you can input breakpoints, single-step through the blocks, inspect variable and property values, and execute any ECMAScript from the query console.

Prior to debugging, you should have validated the workflow, generated the code, and deployed the Project for testing. Also, if you have not already done so, set ORS Debugger preferences. To start debugging, create a launch configuration for the file you want to debug. An example launch configuration is shown below.



Name: simple.workflow (orsdebug)

Workspace Storage Location | ORS Debugger Launch

ORS Connection

Address	138.120.70.62
Port	7999

Application Details

Application is a:	[✓ Workflow] SCXML
Path	/orsdebug/Workflows/simple.workflow
Associated IPD	/orsdebug/src-gen/IPD_default_defaultWorkflow.scxml
Step through IPD	☐
Attach to existing session	☐

To automatically fill in the debug launch configuration described below, use launch shortcuts. Right-click on the workflow file and select **Debug As Workflow.** Composer automatically fills in the launch configuration.

Creating a Debug Launch Configuration

Note: See Debugging a SCXML Files if you want to view only call traces and not use breakpoints to step through the file. There are various ways to start a debug session. To test your workflow by stepping through it, use Debug Configurations to first create a launch configuration:

1. In the Project Explorer, expand the Composer Project and its workflows subfolder.
2. Right-click on the workflow filename in the Project Explorer and select **Debug as** > **Debug Configurations**. The Debug Configuration dialog box opens.

3. Expand **Composer** - **ORS Debugger**.
4. Click the button for a new launch configuration or right-click and select **New**.
5. Name the configuration.
6. In the Workspace Storage Location tab, specify the Project name and location for saving SCXML pages executed by ORS. This folder appears in the **Location** field. Optionally, click **Create Automatically** to have the Debugger create a new Project folder to save the SCXML pages as the IPD is debugged. The files fetched may include SCXML pages, audio files, grammars, scripts, and SCXML data.
7. Click the **ORS Debugger Launch** tab.
8. Under **ORS Connection**, the **Address** and **Port** fields reflect the **ORS Server Host Name** and **ORS Server Port** previously entered as ORS Preferences, but can be changed.
9. **Address**. Enter the IP address or host name of the ORS server.
10. **Port**. Enter the debugger port of the ORS server. This is defined in ORS configuration as [scxml]:debug-port, and defaults to 7999. Make sure that ORS has debug-enabled set to true as well.
11. **Path**. Enter the workspace-relative path of the workflow diagram. For example, /MyProject/src-gen/IPD_default_defaultWorkflow.scxml.
12. **Application is a**. Select Workflow to step through the diagram or SCXML if code. If unchecked, it will step through the SCXML code.
13. **Associated IPD**. Enter the name of the interaction process diagram (IPD) associated with the workflow to be debugged. This field is optional because it is possible to run a stand-alone SCXML. Most of the time, you will use launch shortcuts (right-click on workflow or SCXML and select **Run/Debug As**). The fields in the launch configurations are filled in automatically.
14. **Step through IPD**. If enabled and debugging in code mode (as opposed to workflow mode), then the Debugger steps through the SCXML code that is generated from the IPD. Otherwise, it will "skip" through that code. The SCXML code generated from an IPD is generally setting up global variables and functions, so you might not want to go through that every time.
15. **Attach to existing session**. If enabled, ORS will start debugging on an existing session. When you launch the debugging session, Composer will prompt for a session ID in a dialog box. Once you enter the session ID, it will enter debugging mode for that session. If not enabled, ORS will wait for the next session that runs the application at a given URL. Note: The URL will point to the SCXML page that should be debugged. ORS will enter debug mode for the next session that is started for this URL.
16. Click **Apply** and **Debug** when ready.

After you launch, debugging doesn't start until ORS starts the session. You can start the session with a SIP call or multimedia interaction or using the ORS REST API (you need to send a POST request to ORS). The ORS Debugger skips over deactivated blocks **Note:** If the debugging session can't be started, a dialog box appears with an error message.

Stepping

Once debugging is initiated you will see a red box around the first block of the workflow. This indicates the current location where debugging is paused.

1. Step through the workflow. Click the Step Over button to step through the blocks. See Debug View. Application state can be seen in the Variables tab.

2. You can input breakpoints from the Breakpoints View/Toolbar or use the context menu on a block and select Toggle Breakpoint . When breakpoints are set, you can press F5 or click **Resume** to resume the call to the next breakpoint, instead of stepping block-by-block.
3. You can change values of variables in the middle of a workflow. This could be used to quickly change the execution path as the call is progressing. Right-click in the **Expressions** tab and select **Add Watch Expression**.
4. In the Add Watch Expression window, add a new expression to watch during debugging.

To change the value, expand the variable, right-click on the child item and select **Change value**. A popup window as shown above will open, and you can specify the new value. Click **OK**. Proceed with debugging of the application and see the changed value.

Debugging-results Folder

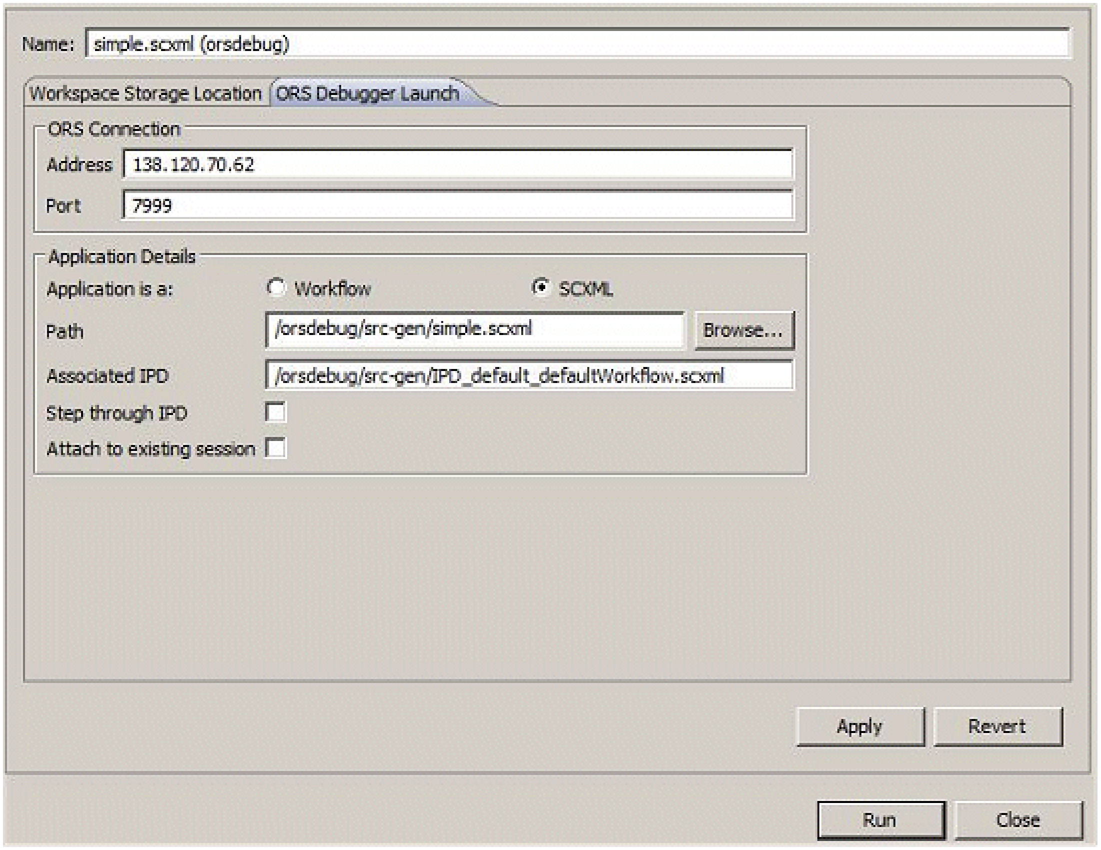
The ORS Debugger creates a `debugging-results` folder in the Project Explorer. Clean up the debugging results by deleting the `ors-debug.<timestamp>` folders from the Project Explorer. Each `ors-debug.<timestamp>` folder corresponds to a single debug call that was made at the time specified by the timestamp. It contains files downloaded by the debugger. The metrics.log file contains the Call Trace of the call.

Debugging SCXML Files

SCXML files can be tested using the real-time ORS Debugger. Both Run and Debug launch configurations are supported, as well as code mode.

- In the Run mode, call traces are provided and the application continues without any breakpoints.
- In the Debug mode, you can input breakpoints, single-step through the SCXML code, inspect variable and property values, and execute any ECMAScript from the query console.

Tomcat engine is bundled as part of Composer and the application can be auto deployed and auto-configured for testing. Also, if you have not already done so, set ORS Debugger preferences. To start debugging, create a launch configuration for the file you want to debug. An example launch configuration is shown below.



Name: simple.scxml (orsdebug)

Workspace Storage Location | ORS Debugger Launch

ORS Connection

Address: 138.120.70.62

Port: 7999

Application Details

Application is a: Workflow [✓ SCXML]

Path: /orsdebug/src-gen/simple.scxml

Associated IPD: /orsdebug/src-gen/IPD_default_defaultWorkflow.scxml

Step through IPD: ☐

Attach to existing session: ☐

To automatically fill in the run launch configuration described below, use launch shortcuts. Right-click on the SCXML file and select **Run As SCXML Page**. Composer automatically fills in the launch configuration.

Creating a Run Launch Configuration

Note: See Debugging a Workflow Diagram if you want to use breakpoints to step through the file by creating a Debug launch configuration. To test your SCXML Files without breakpoints, use **Run Configurations** to create a launch configuration:

1. In the Project Explorer, expand the Composer Project and its src subfolder.
2. Right-click on the SCXML filename in the Project Explorer and select **Run As** > **Run Configurations**. The Run Configuration dialog box opens.
3. Expand **Composer** - **ORS Debugger**.
4. Click the button for a new launch configuration or right-click and select **New**.
5. Name the configuration.
6. In the Workspace Storage Location tab, specify the Project name and location for saving SCXML pages executed by ORS. This folder appears in the **Location** field. Or click **Create Automatically** to have the Debugger create a new Project folder to save the metric traces and SCXML pages as the file is debugged. The files fetched may include SCXML pages, audio files, grammars, scripts, and SCXML data. Call traces are also saved in this location .

7. Click the **ORS Debugger Launch** tab.
8. Under **ORS Connection**, the Address and Port fields reflect the **ORS Server Host Name** and **ORS Server Port** previously entered as ORS Debugger Preferences, but can be changed.
9. **Application is a**. Select SCXML. Leave unchecked to step through the SCXML code. Select **Workflow** to step through a diagram.
10. **Path**. Enter the path for storing files downloaded during a debugging session. Specify the workspace-relative path of the SCXML file. For example, `/MyProject/src-gen/IPD_default_defaultWorkflow.scxml.`
11. **Associated IPD**. Enter the name of the interaction process diagram (IPD) associated with the SCXML file to be debugged. This field is optional because it is possible to run a stand-alone SCXML. Most of the time, you will use launch shortcuts (right-click on workflow or SCXML and select Run/Debug As). The fields in the launch configurations are filled in automatically.
12. Step through IPD. If enabled and debugging in code mode (as opposed to workflow mode), then the Debugger steps through the SCXML code that is generated from the IPD. Otherwise, it will "skip" through that code. The SCXML code generated from an IPD is generally setting up global variables and functions, so you might not want to go through that every time.
13. **Attach to existing session**. If enabled, ORS will start debugging on an existing session. When you launch the debugging session, Composer will prompt for a session ID in a dialog box. Once you enter the session ID, it will enter debugging mode for that session. If not enabled, ORS will wait for the next session that runs the application at a given URL. Note: The URL will point to the SCXML page that should be debugged. ORS will enter debug mode for the next session that is started for this URL.
14. Click **Apply** and **Run** when ready.

After you launch, debugging does not start until ORS starts the session. You can start the session with a SIP call or multimedia interaction or using the ORS REST API (you need to send a POST request to ORS). Note: If the debugging session cannot be started, a dialog box appears with an error message.

Debugging IPD SCXML Files

Interaction process diagram debugging is not supported. You can debug code generated from an IPD just like any other SCXML page (see Debugging Modes). Prior to debugging, you should have validated the workflow, generated the code, and deployed the Project for testing. Also, if you have not already done so, set ORS Debugger preferences. **Creating a Debug or Run Configuration** The focus is on stepping through the workflow diagrams called from the IPD as the bulk of routing logic and decision making is done at the workflow level. IPD flows are straightforward and in most cases should not require debugging. To debug/run an IPD, you first create a launch configuration.

1. In the Project Explorer, expand the Composer Project and its Interaction Processes subfolder.
2. Right-click on a workflow diagram, and select **Debug As** or **Run A**s. This will create a launch configuration automatically and start the debugging session.
3. Optionally, at a later time, you can go back to the launch configuration, and check Step through IPD if you want to step through the IPD as well.
4. Click the **ORS Debugger Launch** tab.
5. Under **ORS Connection**, the **Address** and **Port** fields reflect the **ORS Server Host Name** and **ORS Server Port** previously entered as ORS Debugger Preferences, but can be changed. **Address**. Enter

the IP address or host name of the ORS server. **Port**. Enter the debugger port of the ORS server. This is defined in ORS configuration as [scxml]:debug-port, and defaults to 7999. Make sure that ORS has debug-enabled set to true as well. **Path**. Enter the workspace-relative path of the IPD SCXML file. For example, `/MyProject/src-gen/IPD_default_defaultWorkflow.scxml`.

6. **Associated IP**D. Enter the name of the interaction process diagram (IPD) associated with the SCXML file to be debugged. This field is optional because it is possible to run a stand-alone SCXML. Most of the time, you will use launch shortcuts (right-click on workflow or SCXML and select **Run/Debug As**). The fields in the launch configurations are filled in automatically.
7. Step through IPD. See Step 3. If enabled and debugging in code mode (as opposed to workflow mode), then the Debugger will step through the SCXML code that is generated from the IPD. Otherwise, it will "skip" through that code. The SCXML code generated from an IPD is generally setting up global variables and functions, so you might not want to go through that every time.
8. **Attach to existing session**. If enabled, ORS will start debugging on an existing session. When you launch the debugging session, Composer will prompt for a session ID in a dialog box. Once you enter the session ID, it will enter debugging mode for that session. If not enabled, ORS will wait for the next session that runs the application at a given URL. Note: The URL will point to the SCXML page that should be debugged. ORS will enter debug mode for the next session that is started for this URL.
9. Click **Apply** and **Run** when ready.

After you launch, debugging does not start until ORS starts the session. You can start the session with a SIP call or multimedia interaction or using the ORS REST API (you need to send a POST request to ORS). **Note**: If the debugging session cannot be started, a dialog box appears with an error message.

Stepping Through a Routing Application

Note: Step Over on the Debugging Toolbar is the only way to step for both routing and voice applications. Stepping means executing the workflow one step at a time, suspending execution between steps, and using variables, breakpoints, and watch expressions. You can then examine the state of the application when it is suspended. At each step of execution, the Debugger displays metrics (call traces) received from ORS. This topic covers the following:

Suspending Execution

When execution is suspended:

- ORS displays the text of the SCXML page currently being executed.
- Highlights the line that will be executed in the next step.

When you choose to step again, the next step is executed and execution is suspended again. For information on how to step and suspend, see the Debugging Toolbars topic.

Using Variables

The ORS execution context contains ECMAScript variables. Composer access these variables by using the Eval message.

- When execution is suspended, Composer displays the workflow and project variables in the current scope of the ORS execution. The variables are displayed in the Variables tab in ORS Debugging Perspective.
- If the variable is a complex ECMAScript object, You can expand it to view the contents of the object.
- You can create a watch expression from any variable or sub-object of a variable.

For more information, see the Debugging Toolbars topic.

Using Breakpoints

Breakpoints allow you to select a position in the SCXML application to suspend execution. Instead of stepping through execution, You may wish to resume the application, which means to run it until the next breakpoint is reached. You create a breakpoint on a line of the current page. A line breakpoint is reached if the execution reaches the line number of the breakpoint. Line breakpoints are allowed only on SCXML elements which support suspending. The following elements support suspending: onentry, onexit, invoke, transition, script, log, if, assign, raise send, cancel. If you try to create a breakpoint on an unsupported element, the breakpoint is moved to the next valid line. From the Breakpoints View/ Toolbar, you can:

- Display a list of all current breakpoints and enable/disable them. If you disable a breakpoint by unchecking the box in the list of breakpoints, then execution is not suspended if that breakpoint is reached. You may also disable all breakpoints by clicking the Skip All Breakpoints action from the Eclipse Run menu.
- Remove a breakpoint.

Breakpoints are saved across sessions.

Using Watch Expressions

You create watch expressions to monitor the value of a given expression. The expression is an ECMAScript expression, e.g., a variable or function call, that is evaluated in the scope of current SCXML application. From the Expressions View/Toolbar, you can:

- Add a watch expression. The value of the expression is displayed immediately, and updated when the value changes.
- Modify the value of a variable in a watch expression. The change is then reflected in the SCXML application context.
- Disable a watch expression. When disabled, the watch expression value will not be updated.
- Remove a watch expression.

Debugging Modes

The ORS Debugger supports two debugging modes: Diagram and Code.

- Launching a workflow in Debug mode starts debugging in diagram mode.
- Launching a SCXML file in Debug mode starts debugging in code mode.

The table below summarizes the modes for different file types. Also see ORS Debugger Limitations.

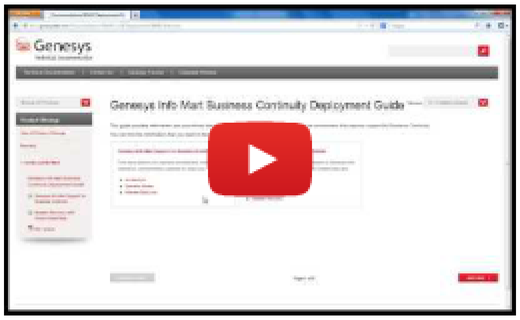
Debug File	Mode	Supported	Procedure to Launch
Interaction process diagram	As Code	Yes	Run/Debug as SCXML from an IPD SCXML. When debugging an SCXML file. Composer supports debugging of the SCXML document generated for the IPD diagram. It will not be a common use case to only debug the IPD.
Interaction process diagram	As Diagram	No	IPD debugging is not supported. Code generated from an IPD can be debugged just like any other SCXML page.
Workflow	As Code	Yes	Run/Debug as SCXML, from a workflow SCXML. This will basically still run the IPD SCXML but execution will pause at the <AppEntry> state of the workflow SCXML.
Workflow	As Diagram	Yes	Run/Debug as Workflow, from a workflow diagram. It will detect the associated IPD SCXML and run that.
Sub-Workflow	As Code	Yes	A subroutine cannot be debugged on its own. Put a breakpoint on the subroutine’s Entry block and debug its calling workflow.
Sub-Workflow	As Diagram	Yes	A subroutine cannot be debugged on its own. Put a breakpoint on the subroutine’s Entry block and debug its calling workflow.
Hand-coded SCXML	As Code	Yes	Should work identical to IPD SCXML debugging.

Debugging Voice Applications

Video Tutorial

Below is a video tutorial on debugging VoiceXML applications.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```



GVP Debugger

Composer's GVP Debugger provides real-time debugging capabilities for Genesys Voice Portal voice applications. The debugger is integrated with GVP for making test calls, viewing call traces, and debugging applications. It supports accessing SOAP and REST based Web Services. Database access is provided using server-side logic and a Web services interface. Prior to debugging, set preferences for the GVP Debugger, which supports both Run and Debug modes.

Run Versus Debug

- In the Run mode using **Run** > **Run Configuration**, call traces are provided and the application continues without any breakpoints.
- In the Debug mode, using **Debug as** > **Debug Configuration**, you can input breakpoints, single-step through the code, inspect variable and property values, and execute any ECMAScript from the query console.

Integration with a SIP Phone is provided with a click-to dial feature for making the test calls. You can debug:

- A callflow built with Composer, or
- Any VoiceXML application or set of VoiceXML pages whether or not they were created with Composer.

Notes:

- Previous to GVP 8.1.4, one instance of GVP's Media Control Platform (MCP) supported only one Composer debugging session. This limitation no longer exists in GVP 8.1.4.
- Debugging is supported only on Tomcat. VXML debugging is the same on any application server so debugging using Tomcat is sufficient.
- Composer 8.1 uses TCP to send SIP messages (previous releases used UDP). This is not a configurable option.

Starting a Debugging Session

If using Context Services:

- Set Context Services parameters: **Window** > **Preference**s > **Composer** > **Context Services**. In Context Services Preferences, specify the Universal Contact Server host and port.

You can start a debugging session in the following ways:

- Right-click on the diagram/VXML file. Select **Run As** > **Run Callflow** or **Debug As** > **Debug Callflow**.
- On the main toolbar, there is a Debug button and a Run button. Clicking relaunches the most recently used configuration. You can also use the keyboard shortcuts Ctrl+F11 (for Run) and F11 (for Debug).
- If you click the down arrow on these buttons to drop down a menu, a history of recent launches appears.

All of the above is also available in the Run top-level menu.

Debugging When Using Context Services

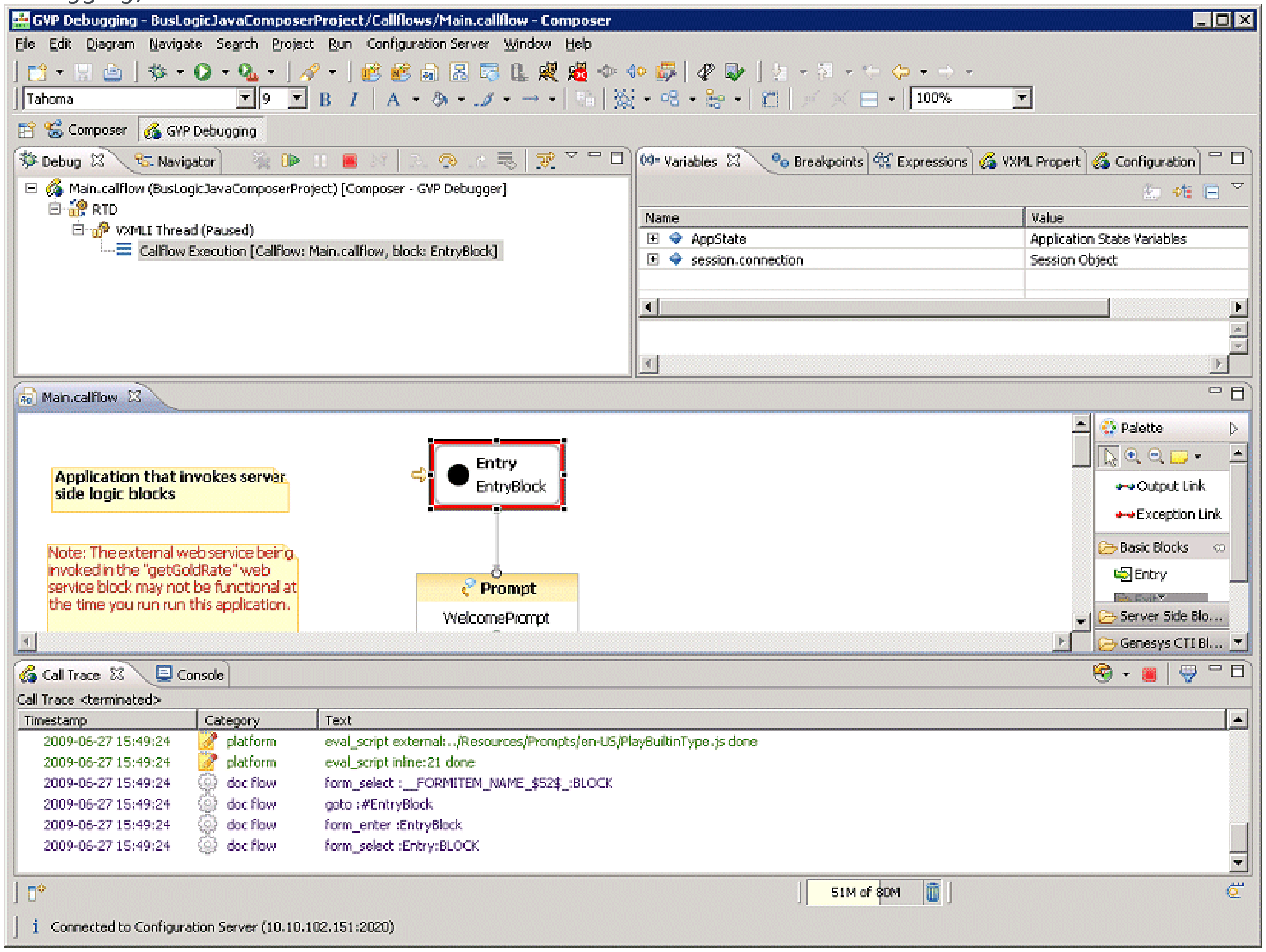
If using Context Services, you must do the following before debugging a callflow or a VXML application:

1. Check the **Connect to the Universal Contact Server when designing diagrams** preference option in the Context Services preference page.
2. Set the UCS parameters Context Services preference page.

Composer then automatically appends an extra `context_services_url` parameter to the SIP URI. This parameter is then read by the GVP application at runtime, enabling the GVP application to connect to the UCS.

GVP Debugging Perspective

The figure below shows Composer's elements for the GVP Debugging perspective (callflow debugging):



Main.callflow (BusLogicJavaComposerProject) [Composer - GVP Debugger]
RTD
VXMLI Thread (Paused)
Callflow Execution [Callflow: Main.callflow, block: EntryBlock]

Name	Value
AppState	Application State Variables
session.connection	Session Object

Application that invokes server side logic blocks

Note: The external web service being invoked in the "getGoldRate" web service block may not be functional at the time you run run this application.

Call Trace <terminated>

Timestamp	Category	Text
2009-06-27 15:49:24	platform	eval_script external:../Resources/Prompts/en-US/PlayBuiltinType.js done
2009-06-27 15:49:24	platform	eval_script inline:21 done
2009-06-27 15:49:24	doc flow	form_select :__FORMITEM_NAME_52_:BLOCK
2009-06-27 15:49:24	doc flow	goto :#EntryBlock
2009-06-27 15:49:24	doc flow	form_enter :EntryBlock
2009-06-27 15:49:24	doc flow	form_select :Entry:BLOCK

Connected to Configuration Server (10.10.102.151:2020)

- The **Debug** view shows the callflow diagram name being debugged, as well as the status of the debug progress or result.
- The Navigator view shows the same Project folder structure shown in the Project Explorer window of the Composer perspective.
- The callflow diagram is displayed below if you are debugging a callflow. At the beginning of the debug session, a red box surrounds the Entry block of the callflow to indicate the start point. The focus changes as the session progresses, and a red box displays wherever the call execution suspends, regardless of whether or not there’s a breakpoint.
- The **Call Trace** view displays metrics which describe the events occurring in the application, such as recognition events, audio playback, user input, errors and warnings, and application output. The history functionality of the call trace view shows the call traces from past calls.
- The **Console** is for executing ECMAScript commands on the interpreter.

Debugging Tools

See the Debugging Toolbars topic.

Debugging Views

The upper right of GVP Debugging Perspective contains the following views:

- **Variables** allows you to monitor the state and value of any variable used in the application, to see how the variable changes during execution. This shows all global variables of the application, and also the recognition results of the previous recognition, if any.
- **Breakpoints** allows you to select a position in the VoiceXML application to suspend execution instead of stepping through the application one command at a time.
- **Expressions** indicates watch expressions you can add and monitor during execution. You can change the value of these expressions to see how they impact the application.

Note: The VXML Properties and Configuration views are for information only; they do not perform any tasks.

- **VXML Properties** are the properties defined in the VXML application by <property> tags. This includes application-specific properties (set in the Entry block) as well as default properties defined by the platform.
- **Configuration parameters** are the configuration items of the VXMLI. Basically it is what you would see in the vxmli section of the MCP settings in MF.

Debugging a Callflow

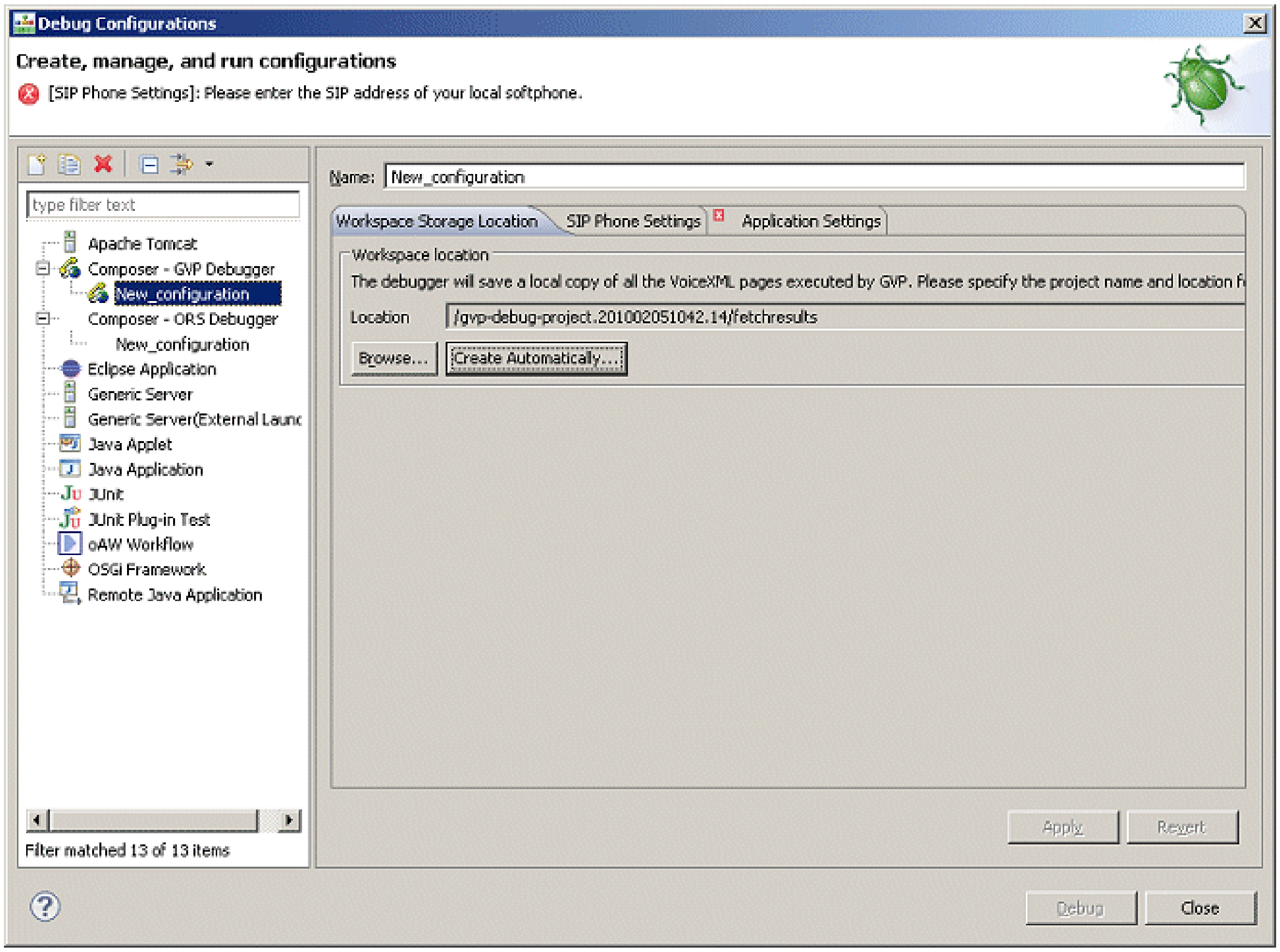
The GVP Debugger allows you to debug a callflow by single-stepping through the blocks. Prior to debugging, you should have validated the callflow, generated the code, and deployed the project for testing. Also, if you have not already done so, set GVP Debugger preferences. Select **Window** > **Preferences** > **Composer** > **Debugging** > **GVP Debugger** and configure the GVP Debugger.

Creating a Debug Launch Configuration

To test your callflow by stepping through it, use Debug Configurations to first create a launch configuration: To run your callflow for debugging use Debug Configurations:

1. In the Project Explorer, expand the Composer Project and its callflows subfolder.
2. Right-click on the callflow filename in the Project Explorer and select **Debug as** > **Debug Configurations**.
3. Expand **Composer** - **GVP Debugger**.

4. Select **New Configuration**. The Debug Configurations dialog box opens. An example is shown below.



Debug Configurations

Create, manage, and run configurations

[SIP Phone Settings]: Please enter the SIP address of your local softphone.

Name: New_configuration

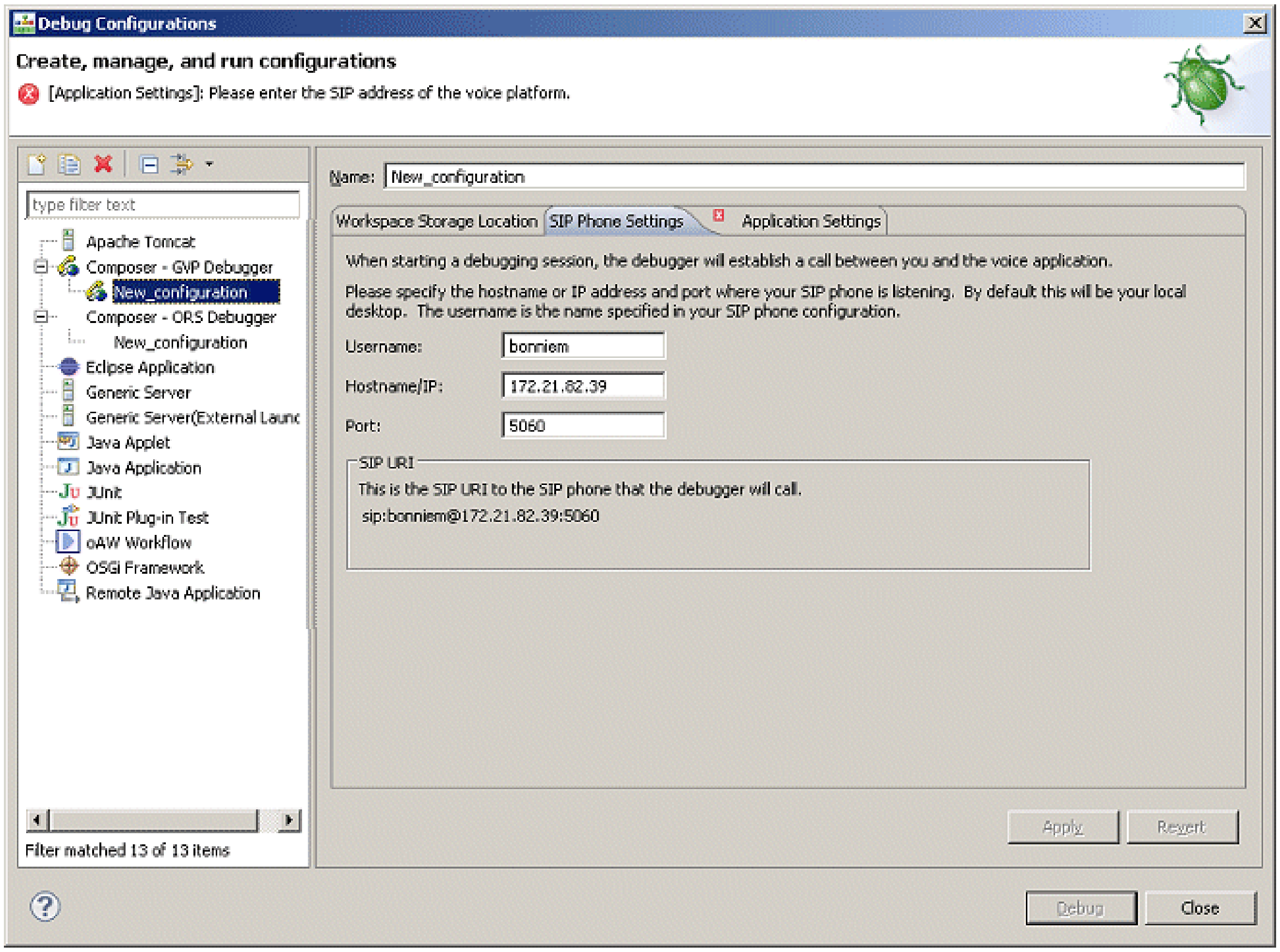
Workspace location

The debugger will save a local copy of all the VoiceXML pages executed by GVP. Please specify the project name and location f

Location /gvp-debug-project.201002051042.14/fetchresults

Filter matched 13 of 13 items

5. Name the configuration.
6. Click the **Create Automatically** button to create a new project folder to save the metric traces and VXML pages as the calls are being executed. This folder appears in the Location field as shown above.
7. Click the **SIP Phone Settings** tab and provide your SIP Phone information if not already there from GVP Debugger Preferences. An example is shown below:



[Application Settings]: Please enter the SIP address of the voice platform.

Name: New_configuration

When starting a debugging session, the debugger will establish a call between you and the voice application.

Please specify the hostname or IP address and port where your SIP phone is listening. By default this will be your local desktop. The username is the name specified in your SIP phone configuration.

Username: bonniem

Hostname/IP: 172.21.82.39

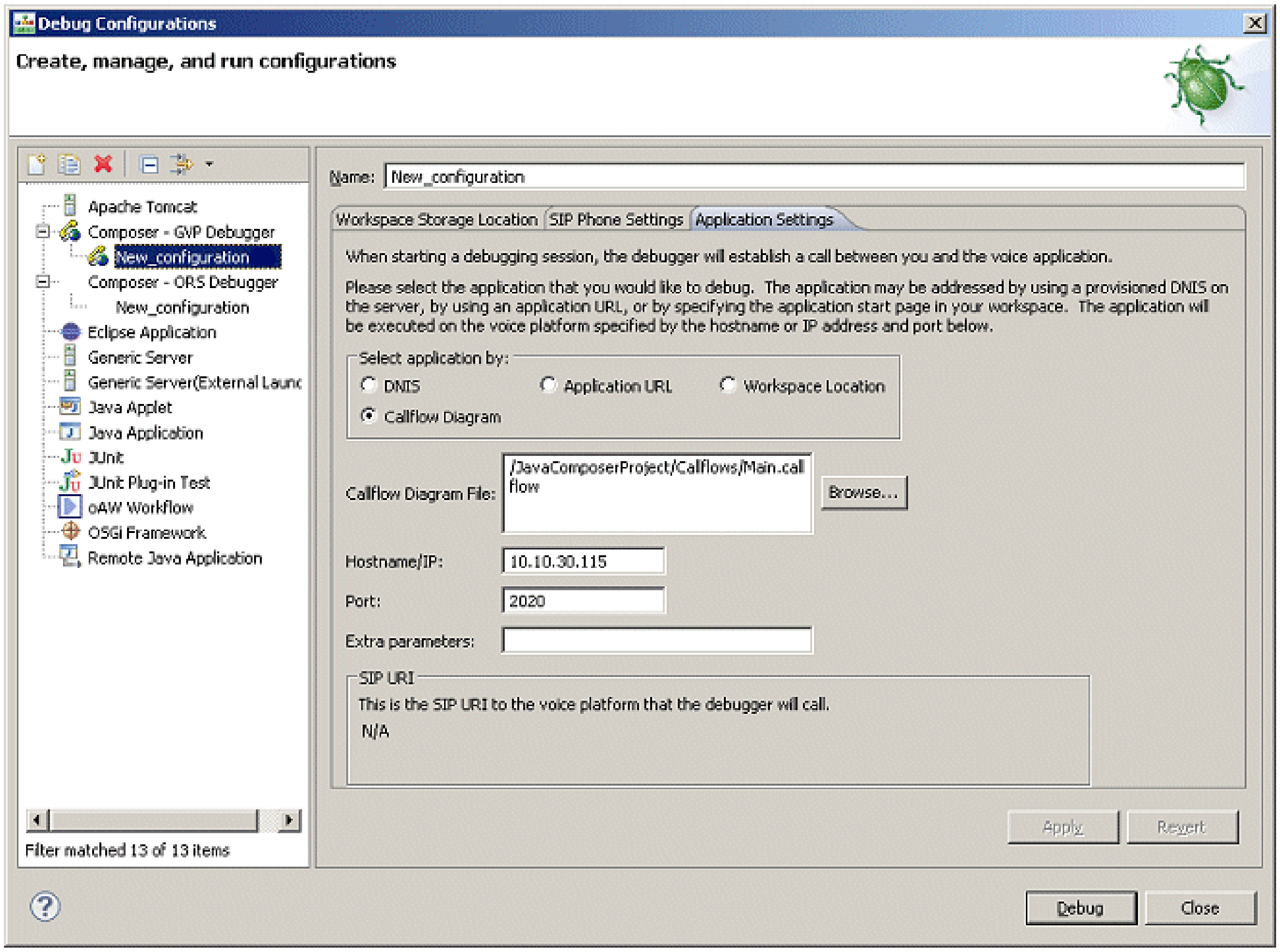
Port: 5060

SIP URI

This is the SIP URI to the SIP phone that the debugger will call.

sip:bonniem@172.21.82.39:5060

8. Click the **Application Settings** tab and select **Callflow Diagram**. An example is shown below:



Name: New_configuration

When starting a debugging session, the debugger will establish a call between you and the voice application.

Please select the application that you would like to debug. The application may be addressed by using a provisioned DNIS on the server, by using an application URL, or by specifying the application start page in your workspace. The application will be executed on the voice platform specified by the hostname or IP address and port below.

Select application by: DNIS Application URL Workspace Location [✓ Callflow Diagram]

Callflow Diagram File: /JavaComposerProject/Callflows/Main.callflow

Hostname/IP: 10.10.30.115

Port: 2020

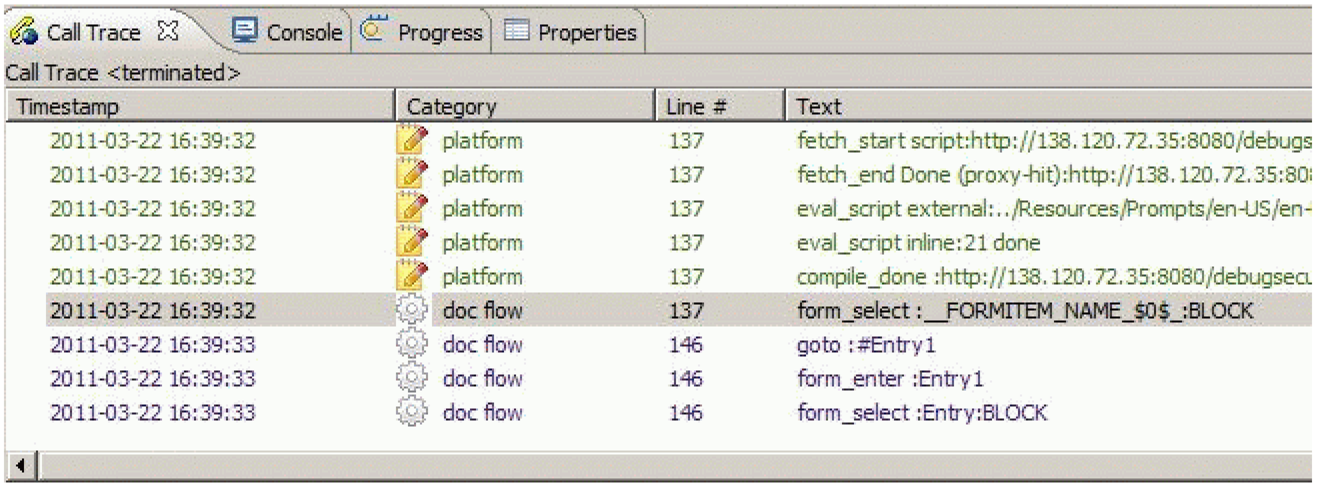
Extra parameters:

SIP URI
This is the SIP URI to the voice platform that the debugger will call.
N/A

9. You can pass CTI Input variables in a Debugger call. Input variables in a callflow diagram can be initialized in a Debugger call using the **Extra Parameters** field in the Run / Debug Configurations > **Application Settings** tab. The Parameter names should match the "Input" variable defined in the Entry Block of the Callflow diagram.
10. Click **Apply**.
11. Click **Debug**. This will automatically dial out your SIP Phone.
12. Accept the call and you will be connected to the application on GVP. Composer switches to the GVP Debugging perspective.

Once the call is initiated you will see a red box around the first block of the application. This indicates the current location where the call is paused. Note: The GVP Debugger skips over deactivated blocks.

12. Click the **Step Over** button to single step through the blocks. **Note:** Step Over on the Debugging Toolbar is the only way to step for both routing and voice applications. Blocks in the diagram correspond to <form> elements in the generated VXML. When stepping through a callflow diagram, the debugger is stepping through <form> elements in the underlying VXML. The call traces will become visible in the Call Trace view at the bottom. An example is shown below.



Call Trace <terminated>

Timestamp	Category	Line #	Text
2011-03-22 16:39:32	platform	137	fetch_start script:http://138.120.72.35:8080/debugs
2011-03-22 16:39:32	platform	137	fetch_end Done (proxy-hit):http://138.120.72.35:80
2011-03-22 16:39:32	platform	137	eval_script external:../Resources/Prompts/en-US/en-
2011-03-22 16:39:32	platform	137	eval_script inline:21 done
2011-03-22 16:39:32	platform	137	compile_done :http://138.120.72.35:8080/debugsec
2011-03-22 16:39:32	doc flow	137	form_select :__FORMITEM_NAME_0_:BLOCK
2011-03-22 16:39:33	doc flow	146	goto :#Entry1
2011-03-22 16:39:33	doc flow	146	form_enter :Entry1
2011-03-22 16:39:33	doc flow	146	form_select :Entry:BLOCK

Application state and last user input values can be seen in the Variables tab.

13. You can input breakpoints from the context menu on a block and select **Toggle Breakpoint** . When breakpoints are set, you can press F5 to resume the call to the next breakpoint, instead of single stepping block-by-block.
14. You can change values of variables in the middle of the callflow. This could be used to quickly change the execution path as the call is progressing. Right-click in the Expressions tab and select **Add Watch Expression**. In the Add Watch Expression window, add a new expression to watch during debugging. For example, give the name as `AppState.<actual variable name>`.
15. To change the value, expand the variable, right-click on the child item and select **Change value**. A popup window as shown above will open, and you can specify the new value. Click **OK**. Proceed with debugging of the application and see the changed value.The value of the watch expression can be refreshed at any time by right-clicking on it and selecting **Reevaluate Watch Expression**.

Debugging-results Folder

The GVP debugger creates a `debugging-results` folder in the Project Explorer. There is currently no automatic cleanup so the number of files can become large. Clean up the debugging results by deleting the `gvp-debug.<timestamp>` folders from the Project Explorer. Each `gvp-debug.<timestamp>` folder corresponds to a single debug call that was made at the time specified by the timestamp. It contains files downloaded by the debugger. The metrics.log file contains the Call Trace of the call.

Code Generation of Multiple Callflows

When using the **Run Callflow** or **Debug Callflow** functions, Composer automatically generates the VXML files from the diagram file that you want to run. In the case of a Java Composer Project that has multiple callflows, Composer attempts to generate the VXML for all the callflows before running (because the application might move between multiple callflows for subdialogs). However, if one of the callflows has an error, Composer provides the option to continue running the application anyway, because the erroneous callflow may be a callflow that's not used by the one being run (if there are two or more main callflows, for example). When this happens, the VXML files are basically out of sync with the diagram files and this may affect execution. Genesys recommends that you fix all errors before running the application.

Debugging VoiceXML Files

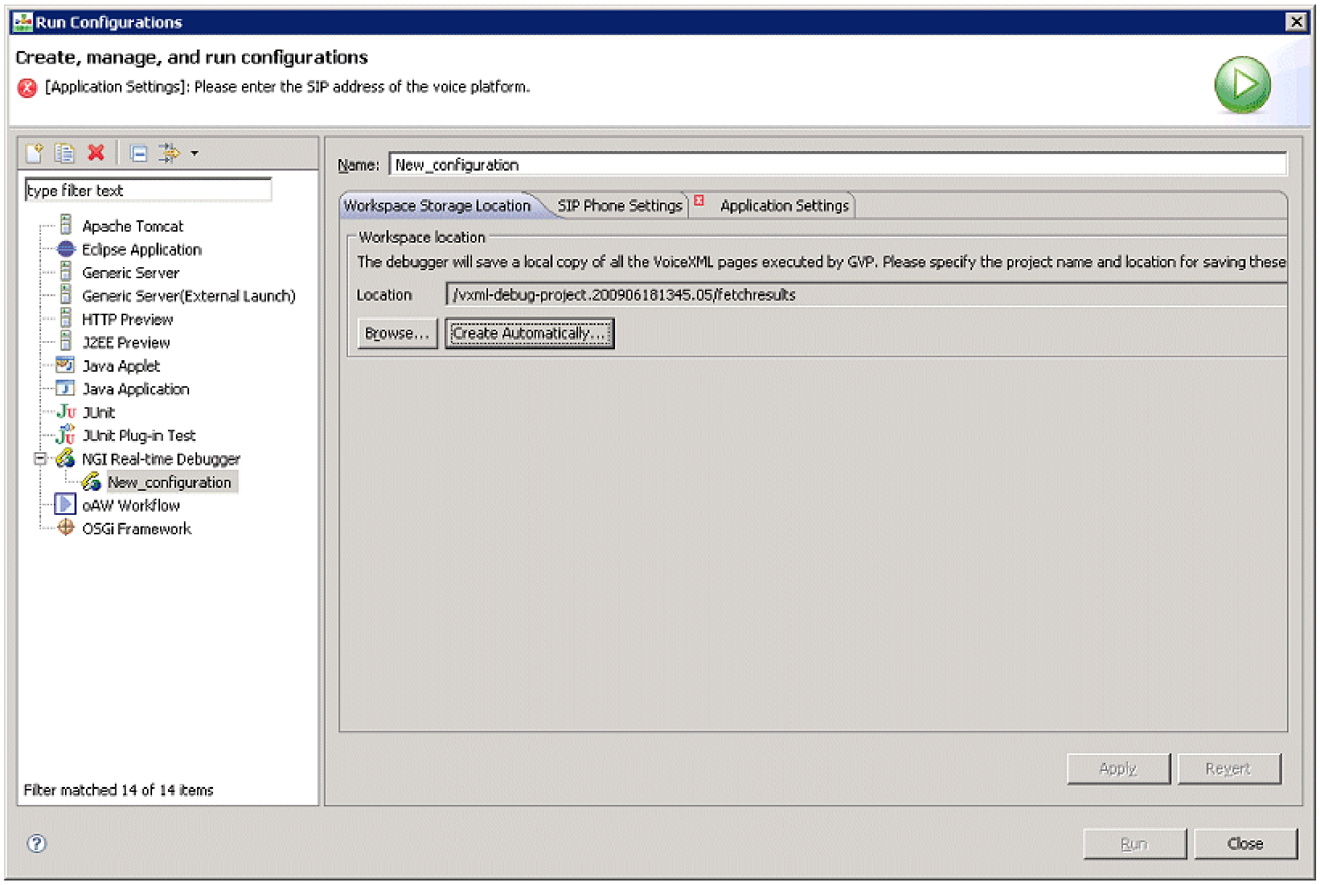
- VXML debugging does not work on 64-bit operating systems when Transport Layer Security if TLS is enabled in Context Services Preferences. If TLS is not enabled, debugging works as expected.
- Debugging is supported only on Tomcat. VXML debugging is the same on any application server so debugging using Tomcat is sufficient.
- When subdialog calls external VXML page in debug mode, Composer throws the following error: An internal error occurred during: Debug Source Lookup. This is a known limitation. The problem occurs when stepping through a callflow with a subdialog block that links to a VXML page. A "mode switch" between debugging a callflow diagram and debugging a VXML page is not supported. Workaround is to start debugging the generated code instead as a VXML page, which will step into the hand written VXML page when it is called.
- VoiceXML applications can be tested using the real-time GVP Debugger. Support for both Run and Debug mode is provided.
- In the Run mode, the call traces are provided and the application continues without any breakpoints.
- In the Debug mode, you can input breakpoints, single-step through the VoiceXML code, inspect variable and property values, and execute any ECMAScript from the query console. Integration with a SIP Phone is provided and click to dial feature is provided for making the test calls.
- Tomcat engine is bundled as part of Composer and the application is auto deployed and auto-configured for testing. Programmers can test by specifying the DNIS of the application already provisioned in MF or provide the URL of the application or let Composer auto-configure the application for testing.

There are various ways to start a debug session. The next section describes Run Mode and Run Launch Configurations.

Creating a Run Launch Configuration

In order to test and debug applications configured in the Genesys Administrator Console as an IVR Profile, you will have to create a launch configuration for making test calls.

1. From the **Run** menu select **Run Configurations**.
2. In the dialog box, right-select **NGI Real Time Debugger** and select **New** from the menu.
3. Define the launch configuration. The figure below shows an example completed Workspace Storage Configuration tab.


Run Configurations

Create, manage, and run configurations

[Application Settings]: Please enter the SIP address of the voice platform.

type filter text

Apache Tomcat
Eclipse Application
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Test
NGI Real-time Debugger
New_configuration
oAW Workflow
OSGi Framework

Filter matched 14 of 14 items

Name: New_configuration

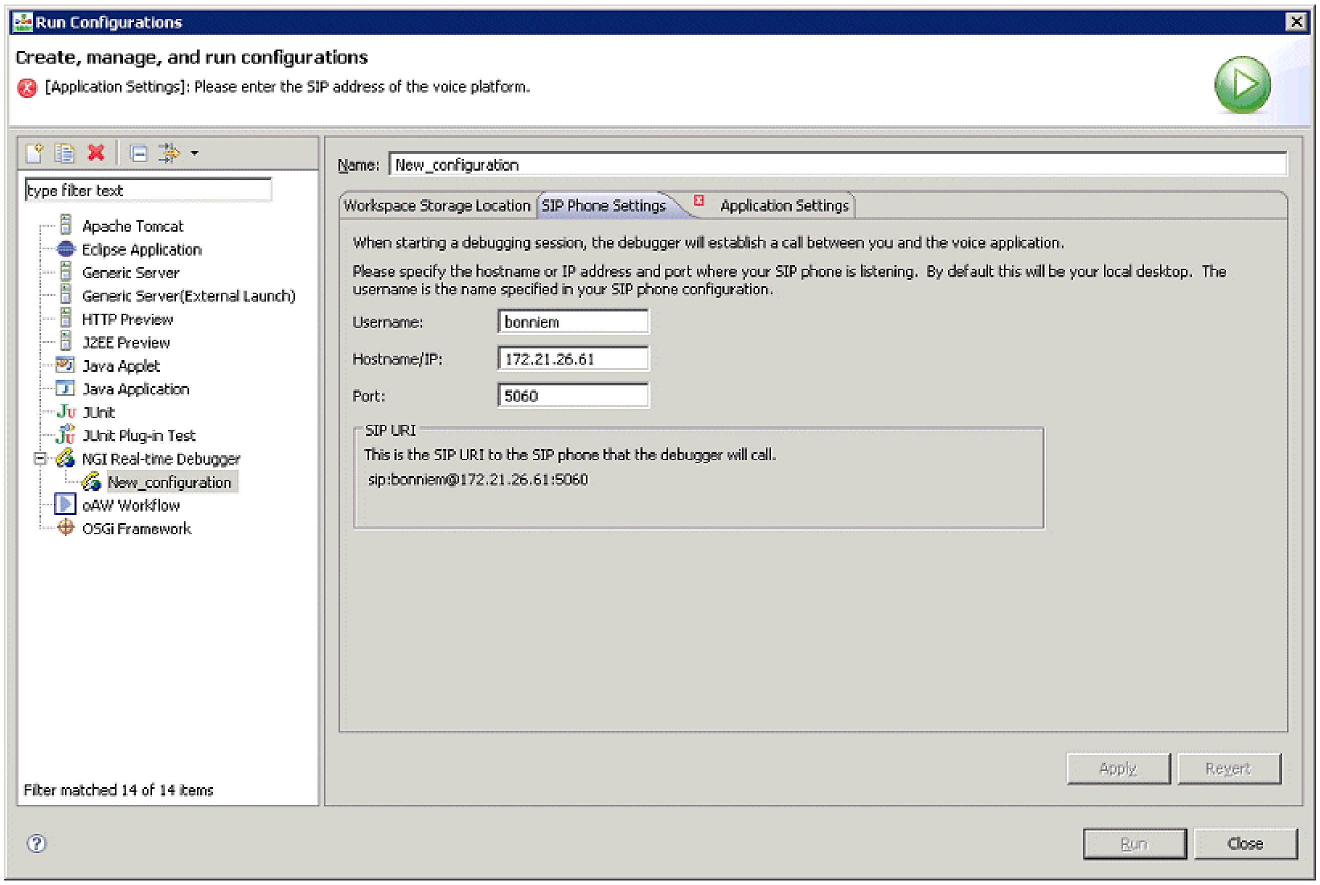
Workspace Storage Location | SIP Phone Settings | Application Settings

Workspace location

The debugger will save a local copy of all the VoiceXML pages executed by GVP. Please specify the project name and location for saving these

Location /vxml-debug-project.200906181345.05/fetchresults

4. Click the **Create Automatically** button to create a new Project folder to save the metric traces and VXML pages as the calls are being executed. This folder appears in the Location field as shown above.
5. Click the **SIP Phone Settings** tab and provide your SIP Phone information if not already there. An example is shown below:



[Application Settings]: Please enter the SIP address of the voice platform.

Name: New_configuration

When starting a debugging session, the debugger will establish a call between you and the voice application.

Please specify the hostname or IP address and port where your SIP phone is listening. By default this will be your local desktop. The username is the name specified in your SIP phone configuration.

Username: bonniem

Hostname/IP: 172.21.26.61

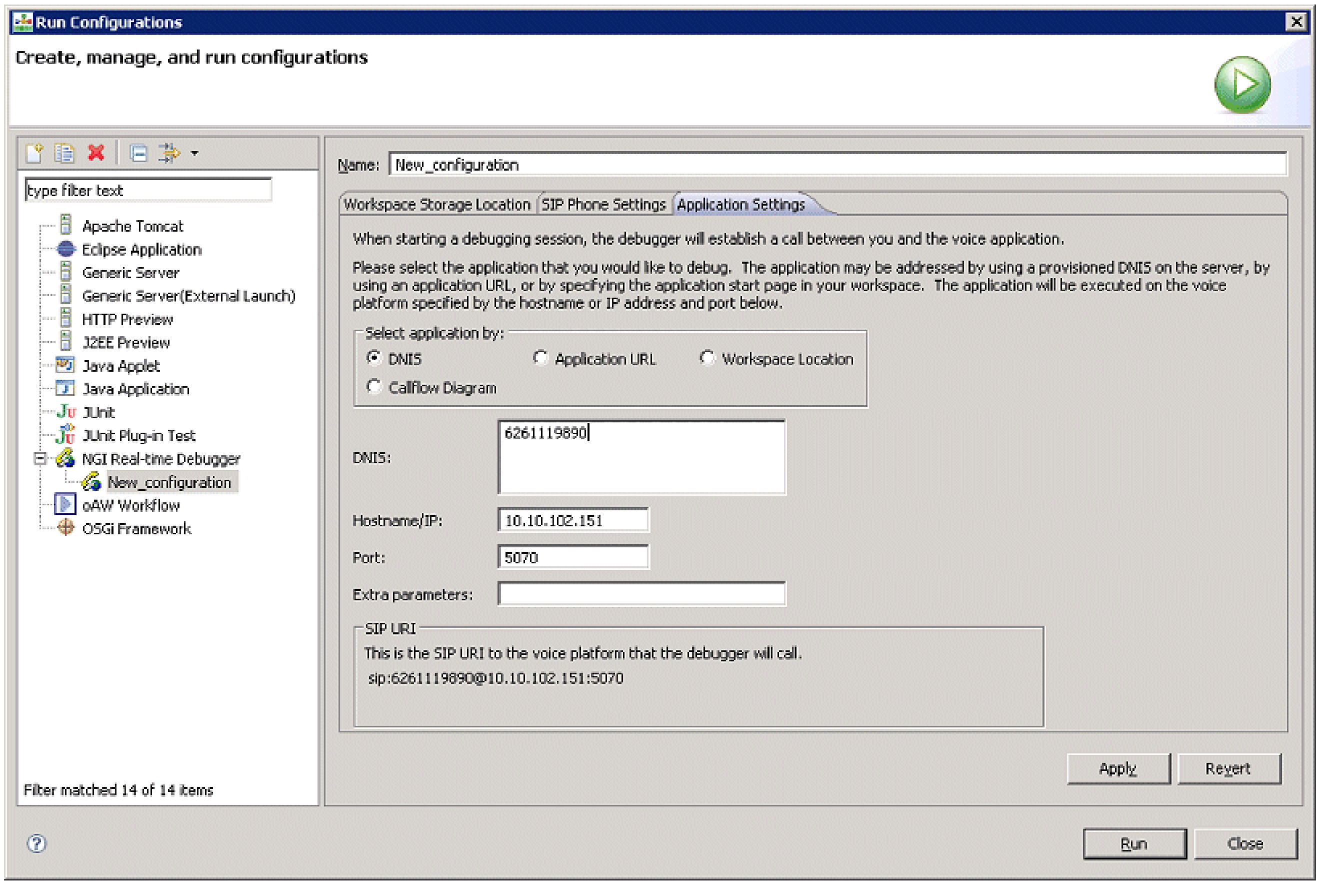
Port: 5060

SIP URI

This is the SIP URI to the SIP phone that the debugger will call.

sip:bonniem@172.21.26.61:5060

6. Click the **Application Settings** tab and select the **DNIS** option. Specify the DNIS of your application and the IP Address of your GVP (MCP / RM), as well as the port. An example is shown below:


Run Configurations

Create, manage, and run configurations

type filter text

- Apache Tomcat
- Eclipse Application
- Generic Server
- Generic Server(External Launch)
- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- NGI Real-time Debugger
  - New_configuration
- oAW Workflow
- OSGi Framework

Filter matched 14 of 14 items

Name: New_configuration

Workspace Storage Location | SIP Phone Settings | Application Settings

When starting a debugging session, the debugger will establish a call between you and the voice application.

Please select the application that you would like to debug. The application may be addressed by using a provisioned DNIS on the server, by using an application URL, or by specifying the application start page in your workspace. The application will be executed on the voice platform specified by the hostname or IP address and port below.

Select application by:
[✓ DNIS] Application URL Workspace Location
Callflow Diagram

DNIS: 6261119890

Hostname/IP: 10.10.102.151

Port: 5070

Extra parameters:

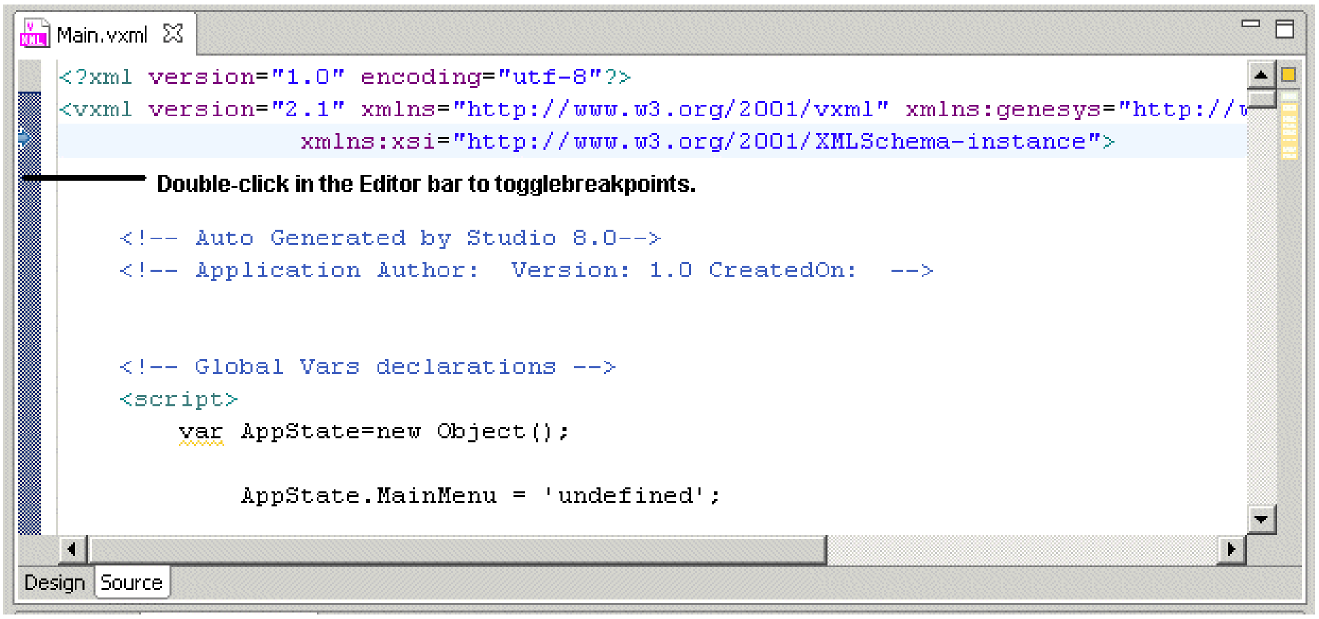
SIP URI
This is the SIP URI to the voice platform that the debugger will call.
sip:6261119890@10.10.102.151:5070

- The **Application URL** option is for an application that is not provisioned, but is hosted at an HTTP URL.
- The **Workspace Location** and **Callflow Diagram** options are generally not used to create launch configurations. Those launch configurations are automatically created using **Run As Callflow** or **Run As VXML file**.
- You can pass CTI Input variables in a Debugger call. Input variables in a callflow diagram can be initialized in a Debugger call using the **Extra Parameters** field in the Run / Debug Configurations > **Application Settings** tab. The Parameter names should match the "Input" variable defined in the Entry Block of the Callflow diagram.

7. Click the **Apply** button and then click **Run**.
8. Your SIP Phone should get dialed. Accept the call in your SIP Phone and then the Debugger will dial out to GVP and connect the call.
9. You should then see call traces in the Call Trace view.

Adding Breakpoints

To toggle breakpoints, double-click the side area of the Editor window as shown in the figure below:

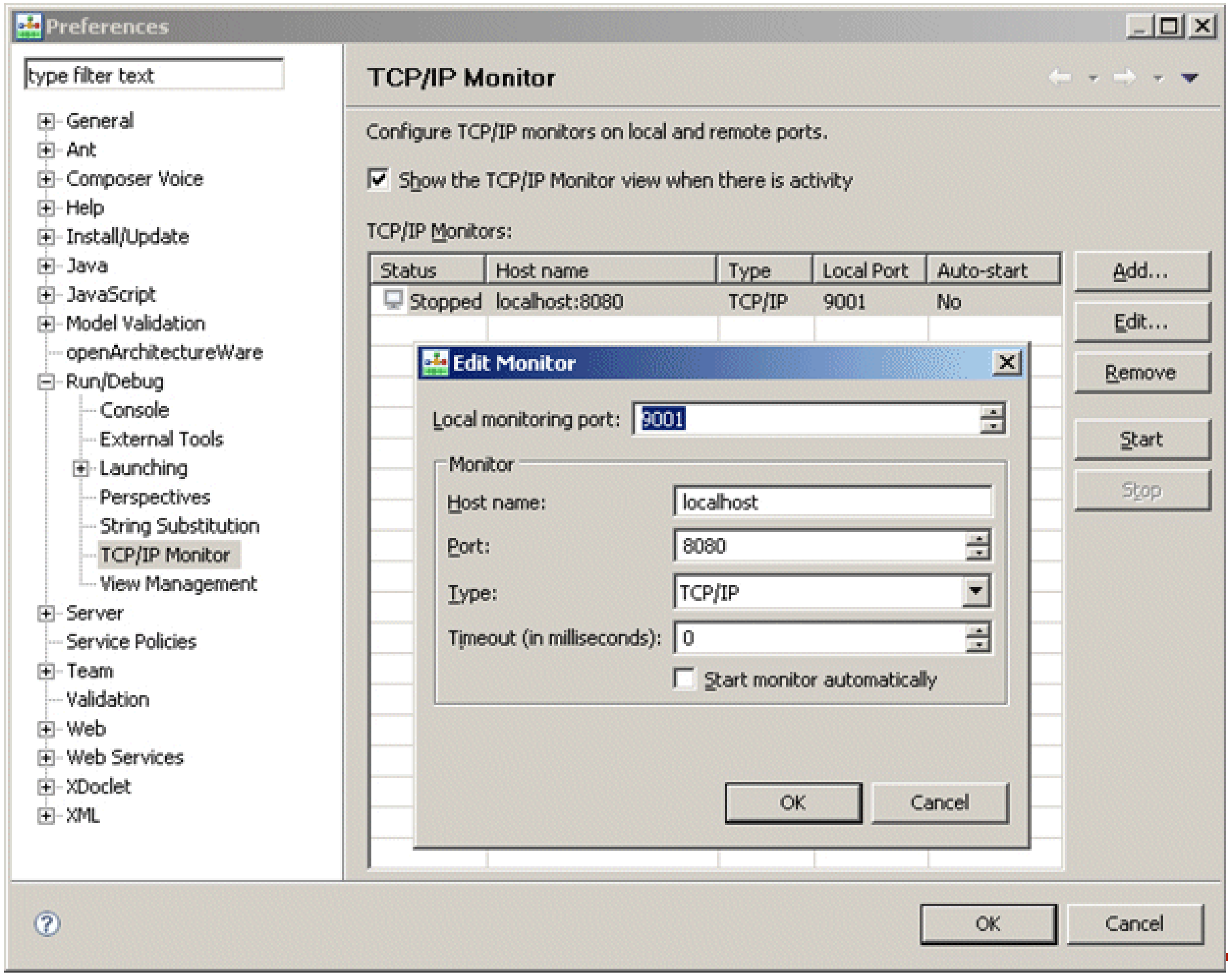


```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml" xmlns:genesys="http://w
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <!-- Auto Generated by Studio 8.0-->
    <!-- Application Author:  Version: 1.0 CreatedOn:   -->


    <!-- Global Vars declarations -->
    <script>
        var AppState=new Object();

            AppState.MainMenu = 'undefined';
```

Debugging Server-Side Pages

This section covers server-side debugging with the TCP/IP monitor. Composer includes a TCP/IP monitor which can be used to debug server-side code such as Server Side blocks and Database blocks. To monitor TCP traffic on the Tomcat or IIS port, follow these steps:

1. Check the Composer Preferences page to determine the port on which your bundled Tomcat is running. If you are using IIS, see the IIS port configured for Composer in Preferences.
2. Enable the TCP/IP monitor in Preferences (**Window** > **Preferences**, expand **Run/Debug** and select **TCP/IP Monitor**. Select the **Show the TCP/IP Monitor view when there is activity** check box.
3. Add a new monitor entry in TCP/IP monitor preferences. The Monitor section refers to the target being monitored. Use the configured port in the previous step as the port to monitor. For example, use 8080 and start the monitor.



TCP/IP Monitor

Configure TCP/IP monitors on local and remote ports.

☑ Show the TCP/IP Monitor view when there is activity

TCP/IP Monitors:

Status	Host name	Type	Local Port	Auto-start
Stopped	localhost:8080	TCP/IP	9001	No

Edit Monitor

Local monitoring port: 9001

Monitor

Host name: localhost

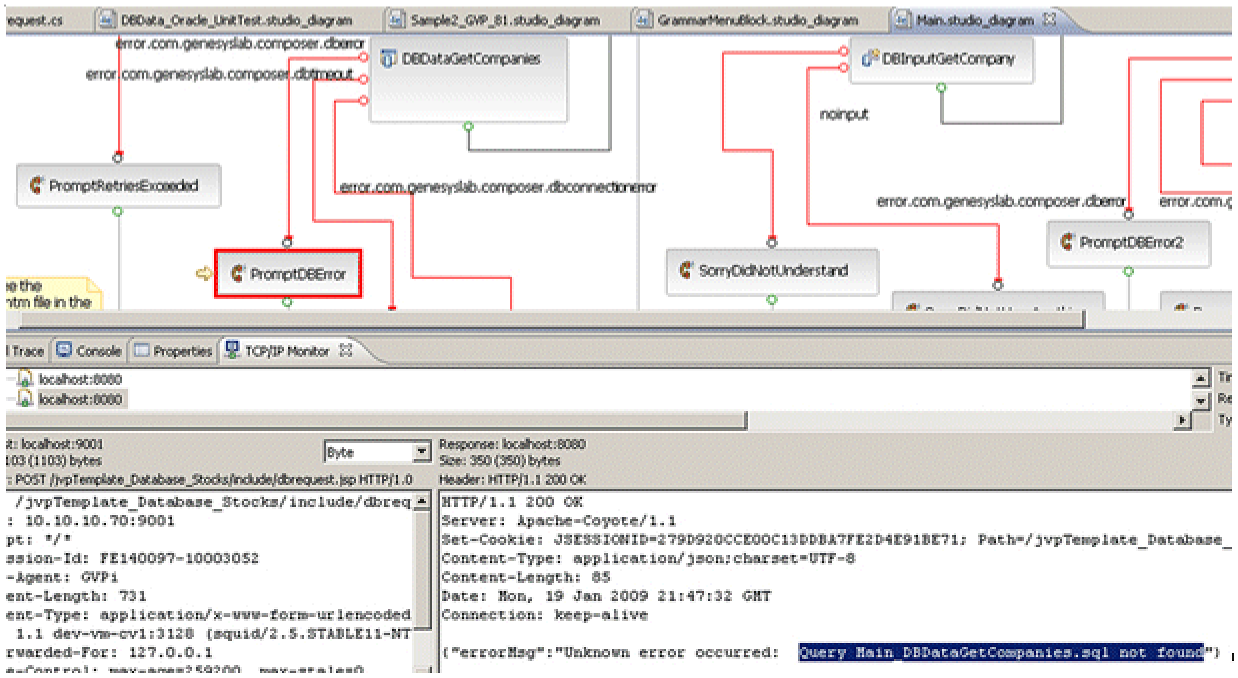
Port: 8080

Type: TCP/IP

Timeout (in milliseconds): 0

☐ Start monitor automatically

4. Update your Tomcat port / IIS port to the Local Monitoring Port.
5. Start debugging your application and put breakpoints on appropriate blocks. The debug perspective will start showing the TCP/IP Monitor view. If the view is not visible, access it from **Window** > **Show View**. The image below shows the bundled Database Stock Application being debugged. The TCP/IP Monitor view shows the error returned by server-side pages that handle database interactions.


equest.cs
DBData_Oracle_UnitTest.studio_diagram
Sample2_GVP_81.studio_diagram
GrammarMenuBlock.studio_diagram
Main.studio_diagram
error.com.genesyslab.composer.dberror
error.com.genesyslab.composer.dbtimeout
DBDataGetCompanies
DBInputGetCompany
noinput
PromptRetriesExceeded
error.com.genesyslab.composer.dbconnectionerror
error.com.genesyslab.composer.dberror
PromptDBError2
PromptDBError
SorryDidNotUnderstand
Trace
Console
Properties
TCP/IP Monitor
localhost:8080
localhost:8080
Byte

```
t: localhost:9001
103 (1103) bytes
: POST /jvpTemplate_Database_Stocks/include/dbrequest.jsp HTTP/1.0
 /jvpTemplate_Database_Stocks/include/dbreq
: 10.10.10.70:9001
pt: */*
ssion-Id: FE140097-10003052
-Agent: GVPi
ent-Length: 731
ent-Type: application/x-www-form-urlencoded
 1.1 dev-vm-cv1:3128 (squid/2.5.STABLE11-NT
rwarded-For: 127.0.0.1
```

```
Response: localhost:8080
Size: 350 (350) bytes
Header: HTTP/1.1 200 OK
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=279D920CCE00C13DDBA7FE2D4E91BE71; Path=/jvpTemplate_Database_
Content-Type: application/json;charset=UTF-8
Content-Length: 85
Date: Mon, 19 Jan 2009 21:47:32 GMT
Connection: keep-alive

{"errorMsg":"Unknown error occurred:  Query Main_DBDataGetCompanies.sql not found"}
```

' For additional information, see the following sections in the Eclipse *Workbench User Guide* Help available from within Composer (**Help** > **Help Contents**):

- TCP/IP Monitor view
- Defining TCP/IP Monitor preference
- Using the TCP/IP monitor to test web services

Debugging TLS Support

The instructions below describe how to (optionally):

- Configure a secure connection (Use Secure Connection) to GVP's Media Control Platform (MCP) during voice application debugging when using SIPS.
- Use a Transport Layer Security (TLS) connection for the Debugger control channel.

For additional information, see the *Genesys 8.1 Security Deployment Guide*. Note: MCP version 8.1.401.07 or later is required to use the TLS feature for debugging. 1. Import the certificate for MCP as follows:

a. In the MCP's configuration, in section `vxmli`, there is a `debug.server.tlscert` parameter. (By default, this is `$InstallationRoot$/config/x509_certificate.pem.`) Copy this file from the MCP installation to a location on Composer's local machine.

b. In Composer, go to **Window** > **Preferences**. Select **Composer** > **Security** in the tree. Click **Import Certificate** and navigate to the x509_certificate.pem file.

c. Restart Composer.

2. Enable debugging from MCP configuration as follows:

   a. Set `[vxmli]:debug.enabled` to **true**.
   b. Ensure that `[vxmli]:debug.server.tlsport and [vxmli]:debugserver.tlsport.public` are set. (By default, it is set to port 27668.)
   c. In Composer, go to **Window** > **Preferences** and select **Composer** > **Debugging** > **GVP Debugger**. Check **Use Secure Connection** to enable security. The Platform Port setting should match the MCP configuration `[sip]:transport.2`.

Limitations

The GVP Debugger has the following limitations:

- CTI calls are not supported.
- Transfers of type blind and consultation do not work when a test call is made using the GVP debugger. The call will reach the Transfer block, but then it will hang and not proceed so you will need to terminate the call or debugging session manually.

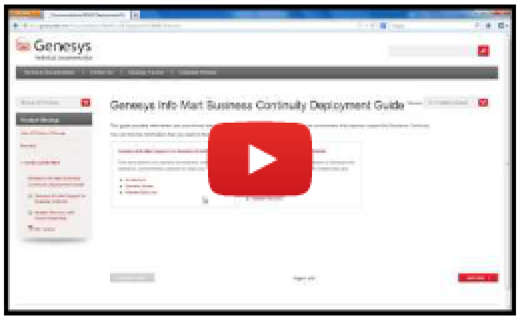
A partial workaround is to set the Transfer Method property to bridge (i.e.. Transfer Type = blind/consultation, Method = bridge) before debugging. The transfer will proceed, but the debugging session and the call will terminate immediately afterwards. Genesys recommends that you do full testing for Transfer applications by provisioning the application in Genesys Administrator and making test calls directly from the SIP phone. These limitations apply to **Run Callflow** (in Run mode) as well.

Deploying Composer Applications

Video Tutorial

Below is a video tutorial on exporting and deploying a Composer application to a web server.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```



Deploying to Apache Tomcat Server

For testing purposes, Composer supports automated deployment of routing applications to the bundled Tomcat server or to a local IIS server. For more information, see Testing Your Application.

Migrating a Composer App From Lab to Production

Automated deployment of a routing application to application servers from within Composer, such as those that would be used in a production environment (JBoss, Websphere, and IIS) is not supported. For more information, see the section on application server requirements in the Composer 8.1.3 Deployment Guide.

Deployment to a Web Application Server

Deployment to a web server depends on which type of Project you are working with:

- Java Composer Project
- .NET Composer Project

Once your application has been unit tested you will need to deploy it to a web server. The deployment process involves:

1. Exporting your Project
2. Transferring the files to your web/application server
3. Executing any necessary configuration steps required to make your application work.

Combination Routing and Voice Projects

A single Composer Project can contain both routing and voice elements. If this is the case, the application will get deployed on a single application server (such has IIS or Tomcat), but must be provisioned in Genesys Administrator in two places:

- In GVP for the voice elements. See the chapter, Post Installation Activities on the GVP Hosts, Provisioning the Components section, in the *Genesys Voice Platform 8.1 Deployment Guide*.
- In URS for the routing elements. See the chapter, SCXML Strategy Support, in the *Universal Routing 8.1 Deployment Guide*.

Microsoft IIS Application Servers

Deploying an Composer application to a Microsoft IIS application server requires Administrative privileges when running the Microsoft Windows 7 and Microsoft Windows Server 2008, 32-bit operating systems.

Deploying a Java Composer Project

Java Composer Projects can be deployed to any web application server that meets the following minimum pre-requisites:

- Must be J2EE 5 compliant.
- Must support the JSP 2.1/Servelet 2.5 specification.

Deploying to J2EE-Compliant Web Application Servers

To deploy a voice application on your Tomcat or JBOSS Production Web Application Server, you will need to export the Project as a `.war` file.

1. Select **File** > **Export**, expand the **Composer** folder, then select **Java Composer Project as WAR file** and click **Next.**
2. Select the Java Composer Project that you wish to export, the display name, and the destination location. A file with the same name as the name of your Java Composer Project and file extension as .war will be created in the destination location. The `.war` file name is always set to the Project name.

The display name is not the file name. It is the application display name used by an application server when the `.war` file is deployed in it. See the web.xml file in `WEB-INF` folder in the generated .war file. It will contain the specified display name. When exporting a `.war` file, the display name is saved on a per-Project basis. Subsequently, when exporting the same Project, the saved name is pre-populated.

2. Select the Generate VXML for the callflow files check box if you would like to auto-generate the code for all the callflows before exporting the `.war` file.
3. Deploy the `.war` file in your Production server or JBOSS server. For example, typically, you will drop the file into the webapps folder of your Tomcat server. If your server is configured to auto-expand the files, you will see a folder created with the same name as your voice Project. If auto-expand is not configured you will have to stop and start your web server in order to expand the `.war` file.

Specifying the URL

Your application will run on a URL of the following type: `[http:// http://]<ip_of_application_server>:<port>/<voice_project_name>/src-gen/<callflow_name>.vxml` Here is an example: http://192.168.1.1:8080/HelloWorld/src-gen/main.vxml Use this URL to provision your application in the Genesys Administrator Console as an IVR Profile. You can then make direct calls via the DNIS associated with the IVR Profile. Also see the topic: Deploying Applications that Use Context Services.

Deploying a NET Composer Project

.NET Composer Projects developed with Composer can be deployed on the IIS web application server.

- If you rename a .NET Composer Project, the new Project is not automatically deployed to IIS. The workaround is to undeploy the Project before renaming and then deploy manually after renaming.
- To deploy Composer .NET Projects to IIS, the IIS 6 Metabase Compatibility must be installed.

To deploy a voice application on IIS:

1. Generate the code for your project.
2. Right-click on your project and click **Export**.
3. In the Export dialog box, select **General** > **File System** and click **Next**.
4. Select all folders except `simulation`, `callflows`, and `debugging-results`.
5. In the To directory box, select the location in your file system where you want to export the application. Select the option for **Create only selected directories** then click **Finish** to export.

All your Project files should be exported to the location that you specified.

6. You can copy this folder to your final deployment machine.
7. Create a virtual directory for this application in IIS and point it to this folder:
    - In **Virtual Directory Alias**, specify the name that will be used to access this virtual directory from HTTP, then click **Next**.
    - Browse to the folder that has your application's exported contents, then click **Next**.

- Give the following permissions: **Read**, **Run Scripts**.
- Configure the Mime types for the deployed .NET Composer Projects manually. The following mime types should be added: `.grx`
- Open IIS and select the website you want to use. Right-click it and select **New Virtual Directory**. A wizard dialog will be displayed. Click **Next** to start it.
- `ml` and `.vxml`

To add a mime-type, open Internet Services Manager and follow these steps:

- Right-click your website (such as Default Web Site) and select properties.
- Click the HTTP Headers tab.
- Click the **MIME Types** button to display the MIME Types dialog box.
- Add these MIME types:
    - `.grxml  application/srgs+xml`
    - `vxml   text/xml`
- Make sure that ASP.NET extensions are enabled in your IIS.
- Make sure that ASP.NET is enabled on your virtual directory and set to the correct version.
- Make sure that scripts have execute permissions on your virtual directory.

8. Right-click on the main vxml page in your `src-gen` folder and select **Browse**. If all settings are correct, a browser window will open and show you the VXML page. The address in the browser will be the URL at which your VXML application will be available.

Deploying a Routing Application

Deploying routing applications involves two main tasks.

1. Creating the appropriate objects in Configuration Server that are required by the Universal Routing platform. These objects are needed so that the platform understands how to direct interactions (voice or multimedia) as well as how to process them.
2. Generating SCXML pages accessible to the platform so they may be retrieved and processed by the platform.

You can handle both tasks in Composer using its integrated development environment.

- Publishing interaction process diagrams (IPDs) creates most necessary Configuration Server objects.
- Deploying Composer Projects to local application servers (Tomcat for Java Composer Projects and IIS for .NET Composer Projects) makes application SCXML pages available to the platform. However, Composer does not support deploying applications to a production environment. The steps documented below can be used to do that.

The term object is used (e.g., Interaction Queue object) when referencing Configuration Server objects. When blocks in Composer diagrams are referenced, the term block is used (e.g., Interaction Queue block). To deploy a routing application:

1. Deploy your Composer-generated SCXML pages to an application server. Note the URL of the starting SCXML page.
2. Using Genesys Administrator or Configuration Manager, log into the Configuration Database and connect to the Configuration Server that is being used for the environment in which you wish to deploy this application.
3. Create the Configuration objects listed below. All objects must be created under the appropriate Tenant that owns Configuration objects being referenced in your Workflow blocks like Queues, Standard Responses, and so on.

    a. One `Script` object for each Workflow block in your IPDs. The object type should be `EnhancedRouting`. This applies to workflows that process voice or multimedia interactions.

    b. One `Script` object for each Interaction Queue block in your IPDs. The object type should be Interaction Queue. This applies only to IPDs/ workflows that process multimedia interactions.

    c. One `Script` object for each View defined in Interaction Queue blocks in your IPDs. The object type should be Interaction Queue View. This applies only to IPDs/ workflows that process multimedia interactions.

    d. A media server Endpoint for each newly added Endpoint in any Media Server block in your IPDs. This applies only to IPDs/ workflows that process multimedia interactions.

    e. For any E-mail or SMS servers being referenced in Media Server blocks in your IPDs, configure the appropriate POP accounts to route multimedia interactions to the appropriate media server endpoints. This applies only to IPDs/ workflows that process multimedia interactions.

    f. Configure DNs to point to an `EnhancedRouting` object so that voice calls on those DNs invoke the application that is referenced in the Enhanced Routing object. This applies only to IPDs/ workflows that process voice interactions.

Detailed Steps for Creating Configuration Server Objects

The steps are listed below.

1. Common steps to create a`Script` object of a specific type:
    - Create a new `Script` object.
    - Ensure that the correct object type is set e.g. `InteractionQueue` or `EnhancedRouting`.
    - Set the object state to `Enabled`.
    - Check that the object is being created under the correct Tenant object.
2. Creating `Script` objects for Workflow blocks.
    - In Genesys Administrator, navigate to **Provisioning** > **Routing** / **eServices** > **Orchestration**.
    - Create a Script object of type Enhanced Routing. The name can be the name of your Workflow block. If another object already exists by this name, you can use a different name.
    - Specify the URI. It should be the URL of the starting SCXML page of your application.

- Add a parameter: context_management_services_url. Its value should be in the format: [http:// http://]<UCS application host>:<Context Services port>.
- Additional parameters can be specified here. If the names match the names of any Project level variables, those variables will be initialized with values specified here.

3. Creating the Script object of type `InteractionQueue`.

- In Genesys Administrator, navigate to **Provisioning** > **Environment** > **Scripts**.
- Create a `Script` object of type Interaction Queue. The name can be the name of your Interaction Queue block. If another object already exists by this name, you can use a different name.
- In its Annex, create the following sections and keys:

Annex Section	Property	Equivalent Composer Block	Value	Notes
Namespace	Name	Name	<name of the queue>	
Namespace	Description	Queue Description	<descriptive text for the queue>	
Orchestration	Application	--	script:<name of the Enhanced Routing object to which interactions from this queue should be sent	Connects an interaction queue to an Enhanced Routing object. Equivalent to linking an Interaction Queue block to a Workflow block in an IPD.

4. Creating the `Script` object of type Interaction Queue View.

- In Genesys Administrator, navigate to **Provisioning** > **Environment** > **Scripts**.
- Create a `Script` object of type `InteractionQueueView.` The name can be the name of your view defined in an Interaction Queue block. If another object already exists by this name, you can use a different name.
- In its Annex, create the following sections and keys:

Annex Section	Property	Equivalent Composer Block	Value	Notes
View	Name	Name	<name of the queue>	
View	Queue	Parent Interaction Queue block	<name of the Interaction Queue object in Configuration Server	Connects the Interaction Queue View object with its parent Interaction Queue object
View	Description	Description	<descriptive text>	

View	Freeze Interval	Check Interval		
View	Condition	Condition		
View	Order	Order		
View	scheduling mode	Scheduling		
View	"Condition.<Name>"	Parameterized Conditions	<value>	
View	sql-hint	Database Hints		
View	segment-by	Configured Segments	Value will be "value of segment 1, value of segment 2, ..., value of segment n"	Segment names are not used
View	segment-check-interval	Segment Interval		
View	segment-total-limit	Segment Limit		

5. Creating Media Server Endpoints

    - In Genesys Administrator, navigate to the correct Application object representing your media server application of type EmailServer or SMSServer.
    - For EmailServer and SMSServer applications, Endpoints are created in the endpoints:<tenant dbid> section which is specific to a Tenant.
    - For each Endpoint, add a key in the above section with

        1. Key name = <Endpoint name>
        2. Value = <name of the Interaction Queue object to which interactions coming from this Endpoint will be submitted>

This defines an Endpoint and connects it to an interaction queue.

- Once an Endpoint is hooked up to an Interaction Queue object, all interactions coming in from that Endpoint are directed to the connected interaction queue. This object, in turn, is linked to an `EnhancedRouting` object from where the URL to the application is picked up. This completes the flow from the media server to the application SCXML pages.

6. Configuring POP accounts (EmailServer).

    - In Genesys Administrator, navigate to the correct application object representing your media server application of type EmailServer.
    - Accounts defined in EmailServer need to be configured to send e-mail interactions to specific Endpoints. For this, locate the pop-client<some number> section in the application's options that represents a POP account.
    - In this section, set the value of the endpoint property to the name of the Endpoint to which e-mails should be redirected.

7. Configuring DNs. To connect DNs with SCXML applications for voice interactions, specify the following in the Annex of the DN:

Annex Section	Property	Equivalent Composer Block	Value	Notes
Orchestration	application	-	script:<name of the Enhanced Routing object to which interactions from this queue should be submitted>	Connects a DN to an Enhanced Routing object

a. For more details, see the chapter on configuring Orchestration Server in the Orchestration Server 8.1 Deployment Guide.

Deploying Applications That Use Context Services

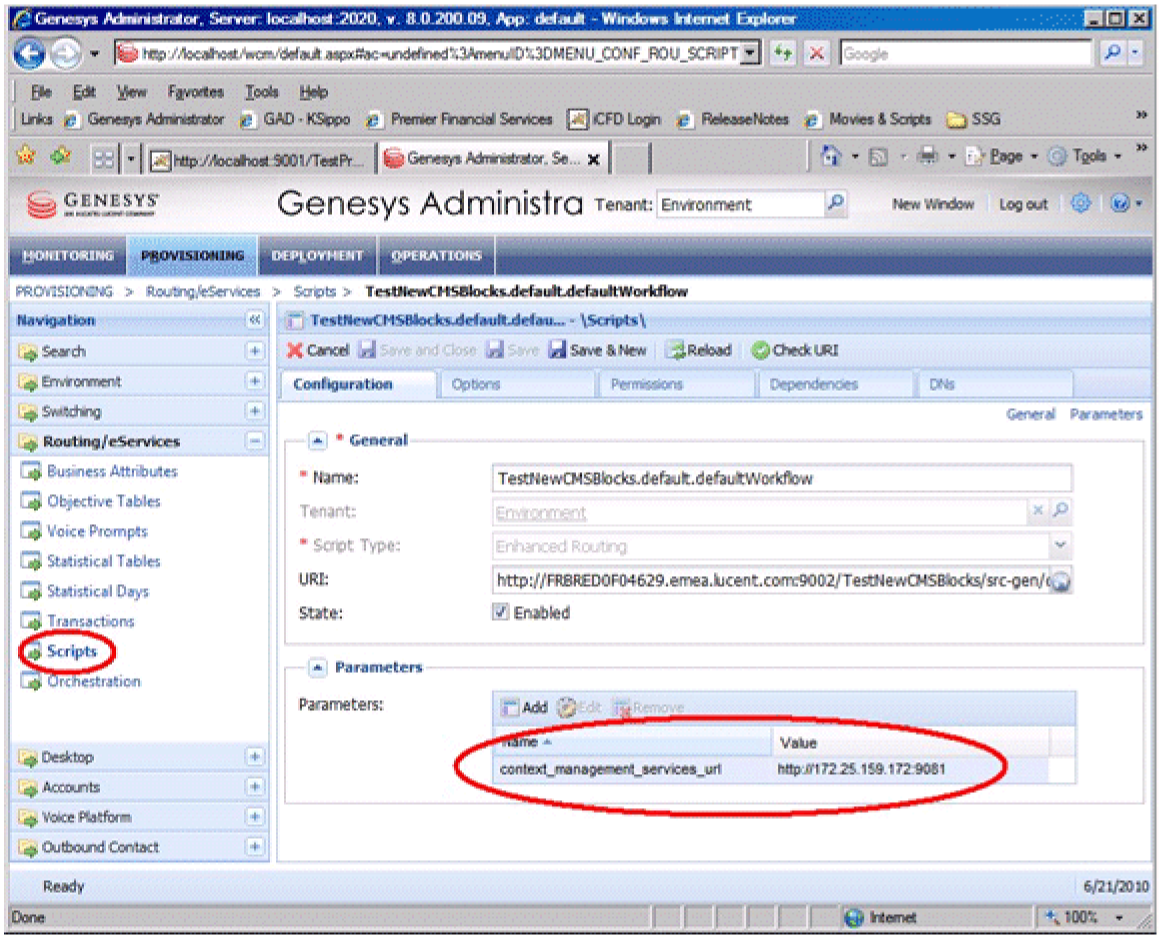
This topic discusses the following types of applications that use Context Services:

- SCXML Applications
- VXML Applications

SCXML Applications

When you publish an interaction processing diagram, Composer creates an enhanced Script record in the Configuration Database. This Script record has a `context_management_services_url` parameter, which is initialized with the UCS server parameters configured in Context Services Preferences. When the SCXML application is run, this parameter is read to enable Orchestration Server to connect to Universal Contact Server (UCS). If you want to point to another UCS, you must either:

- Update the UCS parameters in the Context Services Preferences and re-publish, or
- Manually update the `context_management_services_url` parameter using either Configuration Manager or Genesys Administrator. The example below shows the configuration in Genesys Administrator.


Genesys Administrator, Server: localhost:2020, v. 8.0.200.09, App: default - Windows Internet Explorer
http://localhost/wcm/default.aspx#ac=undefined%3AmenuID%3DMENU_CONF_ROU_SCRIPT
File Edit View Favorites Tools Help
Genesys Administra
Tenant: Environment
New Window Log out
MONITORING PROVISIONING DEPLOYMENT OPERATIONS
PROVISIONING > Routing/eServices > Scripts > TestNewCMSBlocks.default.defaultWorkflow
Navigation
Search
Environment
Switching
Routing/eServices
Business Attributes
Objective Tables
Voice Prompts
Statistical Tables
Statistical Days
Transactions
Scripts
Orchestration
Desktop
Accounts
Voice Platform
Outbound Contact
TestNewCMSBlocks.default.defau... - \Scripts\
Cancel Save and Close Save Save & New Reload Check URI
Configuration Options Permissions Dependencies DNs
General Parameters
Ready
6/21/2010
Done
Internet
100%

General

* Name:	TestNewCMSBlocks.default.defaultWorkflow
Tenant:	Environment
* Script Type:	Enhanced Routing
URI:	http://FRBRED0F04629.emea.lucent.com:9002/TestNewCMSBlocks/src-gen/
State:	☑ Enabled

Parameters

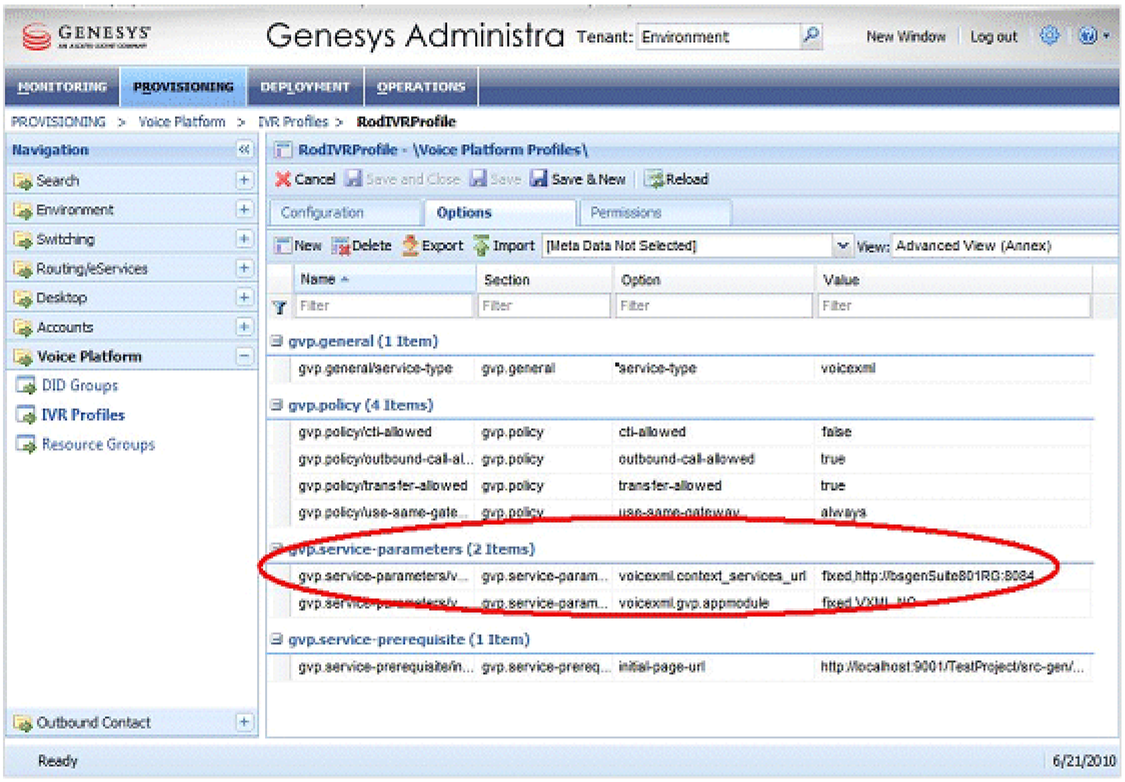
Parameters: Add Edit Remove

Name	Value
context_management_services_url	http://172.25.159.172:9081

MoreImages/contextServicesurl.gif

VXML Applications

When running a callflow from a RM Direct call, you must update the IVRProfile to define an additional `context_services_url` parameter whose value points to the Context Services (UCS) URL.



Name	Section	Option	Value
gvp.general (1 Item)			
gvp.general/service-type	gvp.general	'service-type	voicexml
gvp.policy (4 Items)			
gvp.policy/cti-allowed	gvp.policy	cti-allowed	false
gvp.policy/outbound-call-al...	gvp.policy	outbound-call-allowed	true
gvp.policy/transfer-allowed	gvp.policy	transfer-allowed	true
gvp.policy/use-same-gate...	gvp.policy	use-same-gateway	always
gvp.service-parameters (2 Items)			
gvp.service-parameters/v...	gvp.service-param...	voicexml.context_services_url	fixed,http://bsgenSuite801RG:8084
gvp.service-parameters/v...	gvp.service-param...	voicexml.gvp.appmodule	fixed,VXML_NG
gvp.service-prerequisite (1 Item)			
gvp.service-prerequisite/in...	gvp.service-prereq...	initial-page-url	http://localhost:9001/TestProject/src-gen/...

Running a Callflow from a PlayApplication Workflow Block In this case, you must configure the `context_services_url` parameter in RM's default IVR Profile, which RM passes on to the VXML application. Configuration details are as follows:

1. In the Sip Switch/DN/VOIP Services/MSML_Service DN (if the msml-support option is true in Sip Server) or in the standard VoipService DN (if the `msml-support` option is false in SipServer): change the option contact from `sip:host_MCP:port_MCP` to `sip:host _RM:port_RM`
2. In the Tenant object, designate a default profile for `RM: gvp.general` section, option `default-application`=<name of some IVR Profile object under that tenant>, Default Application for instance.
3. In the IVR Profile/Default Application specified above, in the Annex, add the section `gvp.service-parameters`.
4. In the `gvp.service-parameters` section, add the option `msml.context_services_url=` fixed,http://demosrv8:908 (host:port of Context Management Server aka UCS' customer view port).
5. In the `gvp.service-parameters` section, add the option `voicexml.context_services_url=` fixed,http://demosrv8:9080 (host:port of Context Management Server aka UCS customer view port).

Testing Your Application

After you have saved your files and generated code for your application, test the application as follows:

1. Deploy the project for testing.

- If deploying a Java Composer Project, Composer bundles Tomcat 6.0 for running test applications, such as routing applications. If you configured the Tomcat settings prior to creating your Project, it will be auto-deployed on the Tomcat Server. You can double check this by clicking on the name of the Project in the Project Explorer, then right-click and select **Project Properties**. Select the Tomcat deployment category and verify that the project is deployed. If not, click **Deploy**.

Note: If deploying a .NET Composer Project, deploy your project on an IIS Server. Be sure you have configured the IIS settings. Click on the name of the Project in the Project Explorer, then right-click and select **Project Properties**. Select the IIS deployment category and verify that the Project is deployed. If not, click **Deploy**.

2. For Voice Projects, use Run mode to run the application by selecting **Run** > **Run As** > **Run Callflow**, or by right-clicking on the callflow file name in the Project Explorer and selecting **Run As** > **Run Callflow**. The code is generated in the `src-gen` folder and the debugger sends the call to your SIP Phone.

3. Accept the call and you will be connected to the application on GVP. The call traces will become visible in the Call Trace window, and you should hear the voice application run.

Deploying Updates

This topic summarizes how to deploy updates to a Composer GVP voice application. For example, you may wish to deploy an updated version of a voice application with some new prompts and different DNIS numbers, but want to test it while the old one is still running. To deploy a new version of a Project without affecting the previously deployed one, try the following:

1. Export the existing Project (doesn’t have to be a new Project unless it is needed in cases where SCM tools are not used).
2. Rename the `.war` file before deploying to the application server. The deployed URL will depend on this war file name and therefore result in a different URL for the updated Project. You can control the display name of the application from the export wizard, however, this does not affect the URL.

Notes:

- The above procedure works on Tomcat, but has not been tested on other application servers.
- if the `.war` file is renamed before deploying to the application server, the application URL will reflect the .war file name instead of the Project name. This behavior may be application server-specific. The URL will look like:

```
[http:// http://]<ip_of_application_server>:<port>/<war_file_name>/src-
gen/<callflow_name>.vxml
```

- See the information on specifying the URL in the Deploying a Java Composer Project topic.
- Direct deployment to remote application servers is currently not supported in Composer.

Best Practices

This section discusses the best practices to use when developing efficient Composer applications. It contains the following topics: Also see:

- Deploying Update.
- Best Practices in the *GVP 8.1 VoiceXML Reference Help* available within Composer (**Help** > **Help Contents**).

Pooling Reusable Subflows

This topic discusses how to share a pool of reusable subcallflows between multiple Composer Projects.

Runtime Solution

You can have the shared pool exist in a single Project, which contains the set of subcallflows that is the shared pool.   Each Project that wants to use these subcallflows uses the Subdialog block. The Uri property can be used to specify the location of the subcallflow VXML, which is deployed on an application server. For example, you could enter something like `http://appserver/SharedModules/src-gen/SubFlow1.vxml` `http://appserver/SharedModules/src-gen/SubFlow1.vxml` in the URI field of the Subdialog block (or have the value contained in a variable).   This solution works best if you want to keep a shared pool of subroutines at runtime. If you update a subroutine at the shared location on the application server, all applications that reference it immediately start using the updated subroutine.

Design Time Solution

If you need shared subroutines at design time, but want to include a copy in each application and avoid a global update to all deployed applications, you could try the following:

1. Identify a folder where shared subroutines will be stored. This can be inside a Project or a folder outside your workspace on your hard drive.
2. In any Project that needs to reference subroutines, create a new folder and link it to this shared subroutines folder. This will allow access to all shared subroutines in the referencing Project as if they are a part of the Project. Any changes made to these subroutines will update the master copy and propagate to all other Projects.
3. When you generate code and export a .war file, subroutine code will be included in the export allowing a more controlled deployment of shared subroutines. The drawback of this approach is that you will need to update each application individually.

You can also use an SCM tool to create these linked folders, which may allow other features, such as providing read-only access to shared subroutines from referencing Projects.

Multiple Developer Access to Single Project

At times, you may need to have more than one Developer working on different modules of the same Composer Project. For example, you could have a "modularized" Composer application with each module being its own Subdialog/Subcallflow. In this case, you could have the Composer workspace on a shared network drive with multiple workstations accessing the Project without corrupting the Project metadata. In a such a team development scenario, Genesys recommends using a source code management system. Third party plugins specific to the source control system (ClearCase, Subclipse Team Plugin, etc.) can be installed on top of Composer to enable this functionality. The application files need to be structured to allow individual developers to work on different diagrams. Note: Merging updates to the same diagram from different sources has not been tested so currently Genesys recommends not doing that. If a source code control system is not an option, a shared location could possibly be used simultaneously by more than one developer, but this has also not been tested. The Project could remain on the shared drive and imported into the workspace on each developer’s machine. The import is initiated from **File** > **Import** > **General** > **Existing Projects into Workspace**. Uncheck the option **Copy Projects into Workspace** so that files remain on the shared drive but can be used from the workspace.

Dynamic Web Projects

After you install Java EE Developer Tools plugins, you can create a Dynamic Web Project containing pages with active content. Unlike with static Web Projects, dynamic Web Projects enable you to create resources such as JavaServer Pages and servlets. Here’s how to get started:

1. **Composer Help** >> **Install New Software**.
2. Click **Add**. In the resulting box, enter http://download.eclipse.org/releases/galileo/
3. Select it to see the available package.
4. Select the **Web, XML, and Java EE Development Eclipse Java EE Developer Tools** entry.
5. Install the plugins.
6. Restart Composer.
7. Create a Dynamic Web Project.

Note: Other missing project types can be similarly enabled.

Troubleshooting

For more extensive troubleshooting information for Genesys Voice Platform 8.1 components, please refer to the *Genesys Voice Platform 8.1 Troubleshooting Guide*. The present troubleshooting section considers issues specific to Composer.

General Troubleshooting

If you encounter any of the following symptoms, try increasing the memory allocated to Composer:

- You get `OutOfMemory` Exceptions
- The perspective layout changes on its own
- The system hangs while loading a Composer Project
- The Composer UI appears slow to respond

1. Edit the Composer.ini file present in the installation folder. Open it in a text editor. It will look like this:

```
-vm
j2sdk\jre\bin
-vmargs
-Xms40m
-Xmx256m
-XX:MaxPermSize=256m
```

2. Try increasing the value of `--XX:MaxPermSize` and`--Xmx` to a higher value so that the new value may look like:

```
-vm
j2sdk\jre\bin
-vmargs
-Xms40m
-Xmx512m
-XX:MaxPermSize=512m
```

Block Names & Multi-byte Characters

Composer block names can contain only alphanumeric characters. If multiple-byte characters are used in block names, the code generation step fails and no SCXML or VXML file is generated from the Composer diagram.

Chat Messages in Queue

When a chat server terminates unexpectedly or the server goes offline, the chats that were hosted on that chat server are terminated at the client end. However the chats remain in the system and continue to route to Workspace Desktop Edition (previously Interaction Workspace (IWS)).

IWS presents the chat, but displays a message that it cannot connect to the chat server.

When the agent can eventually mark the chat as done, IWS closes the chat in the agent IWS session, but it routes again, landing on any available agent IWS. This can occur for up to an hour.

The following is an example log entry:

```
14-03-28 12:17:10.138 [ChannelDefault] WARN ESDK - Channel tcp://ctizgesmipr002:4125/ closed
14-03-28 12:17:10.139 [ChannelDefault] WARN ESDK - Channel closing [Name] ChatSessionChannel.
Id001_90f30c5c-63c4-438b-a1b7-72c2dd7a8fc9 [Uri] tcp://ctizgesmipr002:
4125/ has 1 requests pending. They are lost
14-03-28 12:17:10.139 [ChannelDefault] DEBUG ESDK - Chat strategy 'ResyncStrategy'
Processing msg [Name] undefined <check channel evenr> [EndPoint] tcp://ctizgesmipr002:4125/
14-03-28 12:17:10.141 [ChannelDefault] DEBUG ESDK - Chat strategy 'ResyncStrategy'
Found 1 related channel for Event undefined <check channel event>
14-03-28 12:17:10.147 [ChannelDefault] WARN  edia.InteractionChat - Protocol Closed  message:
'Genesyslab.Platform.Commons.Protocols.ProtocolException: Exception occured
during channel opening ---> System.Net.Sockets.SocketException: No connection could be made
because the target machine actively refused it 172.203.217.149:4125
   at System.Net.Sockets.Socket.EndConnect(IAsyncResult asyncResult)
   at Genesyslab.Platform.Commons.Connection.CommonConnection.AsyncConnect(IAsyncResult res)
   – End of inner exception stack trace ---' Previous Channel State:'Opening,
[InteractionChat: Id001/0006Ka9HFHRR0T2B]'
```

Resolution

Use the workflow diagram to stop chat interactions from being routed to an agent. The workflow must determine if a chat session is still accessible before sending the interaction to an agent. One of the possible solutions could be to use "dummy" chat ESP messages. If such a message could not be delivered to Chat Server (see below details about errors), this will indicate not to route the interaction to the agent (and so the interaction could be stopped in workflow). The workflow could make several attempts (in the case where chat High Availability mode will be enabled) before finally stopping the attempt to route.

In order to hide these dummy messages from chat parties, the following could be done:

1. Use some special keywords in the message and modify the web chat application not to show those to the customer. Agent still will be seeing them.
2. Use the "Visibility" parameter of the chat ESP message request. It could have values ALL (default), INT (only agents) or VIP (only supervisors). Setting Visibility=VIP will hide the message from customer and agent (only supervisor will see those and it will be saved in final transcript in Universal Contact Server).

Errors (in the Interaction Server log) when trying to deliver chat ESP message are as follows:

If ChatServer is not running (was not restarted)

```
00:04:08.485 Trc 24112 Cannot find appropriate 3rd-party server: name: [any], type: ChatServer
00:04:08.485 Trc 24102 Sending to Universal Routing Server: URServer: 'EventError' (52)
message:
…
                AttributeErrorCode [int] = 1
                AttributeErrorMessage [str] = "Not found by type"
…
```

After ChatServer was restarted and running

```
00:18:32.532 Trc 26015 Received message 'ExternalServiceFault' ('502') from client
'ChatServer' -
Third party server:0:920, message attributes:
                attr_envelope [list, size (unpacked)=488] =
                   'Parameters' [list] = (size=80)
                      'FaultCode' [str] = "100"
                      'FaultString' [str] = "interaction with specified id was not found"
                   'Service' [str] = "Chat"
                   'Method' [str] = “Message""
```

Checkin Error During Source Code Integration

If you are using Source Control tools, checking in Composer Projects contents after the Project Upgrade process may result in an error.   To solve errors during checkin, refer to the table below and delete the listed files manually in the Source Control.

Composer From Version	Composer To Version	.NET Project, Files/Directories to Delete in Source Control	.Java Project Files/Directories to Delete in Source Control	File Location in the Project
8.0.30*	8.0.40*		lib	.
PlayBuiltinType.js	PlayBuiltinType.js	.\Resources\Prompts\en-US		
8.0.40*	8.1.00*	getWebRequestDataWorkflow.aspx	getWebRequestDataWorkflow.jsp	.\include
		CVDBCommand.cs		.\App_Code
		BackEndLogic.cs		.\App_Code
		CVDBConnProfile.cs		.\App_Code
		dbrequest.cs		.\App_Code
		WebServcClient.cs		.\App_Code
		SPHelper.cs		.\App_Code
		CVDBQueryFile.cs		.\App_Code
		upgradeReports		.\App_Code
8.1.00*	8.1.10*			

Composer Project Not Deployed on Tomcat

In the Project Settings, if you try to deploy the Project and it fails each time, check the following:

1. Are the Tomcat preferences configured correctly?
    - Make sure the post-installation configuration steps have been accomplished. Verify the Tomcat port, user name, and password values.
2. Is the Tomcat service running?

Open **Control Panel** > **Administrative Tools** > **Services**, and look to see if the `ComposerTomcat` service is listed as `Started`.

If not, follow the steps in Tomcat Service Failed to Start to verify that no port clash is present.

Composer Project Not Currently Deployed

If you attempt to run a callflow or workflow and receive a message indicating the following:

```
Could not start the debug target. The Composer Project containing <diagram> is not
currently deployed.
```

this indicates that the Project is not currently deployed. This most likely has occurred because the Project was either imported or renamed.

Do the following to deploy a Java Composer Project on Tomcat:

1. From the Project Explorer, right-click on the Composer Project and select **Properties**.
2. Select **Tomcat Deployment** and click the **Deploy** button.

Do the following to deploy a .NET Composer Project on IIS:

1. From the Project Explorer, right-click on the Composer Project and select **Properties**.
2. Select **IIS Deployment** and click the **Deploy** button.

Connection Profile and ASCII Characters

Database Connection Profiles do not support non-ASCII characters. Use only ASCII characters when creating connection profiles.

Chinese Characters Do Not Display

If Chinese UTF-8 characters cannot be displayed/used in the GRXML/VXML editor, install files for East Asian languages in Windows.

1. Select **Start** > **Control Panel** > **Regional Settings**.
2. Select the **Languages** tab.
3. Select **Install files for East Asian languages**.
4. Windows installs the necessary files for Chinese characters to be displayed correctly in most applications including Composer.

Connection to a Database Fails

If your application or Composer is not able to connect to your database, you can:

- Check connection parameters in the Connection editor UI and verify they are correct.
- Verify that the database is up and running and accepting new connections.
- Check the error message that is stored in the Composer logs. It will contain the database-specific error message returned by the JDBC driver.
- For SQLServer, you can create a new ODBC connection to verify whether the information is correct. Access it from **Control Panel** > **Administrative Tools** > **Data Source (ODBC)**. Provide the same information that you provided in the connection and test the connection.
- For Oracle, run SQLPlus and try connecting to your database with the same information you provided in the Connection Profiles editor.

Oracle and .NET Composer Projects

While working with .NET Composer Projects and Oracle, you may get this error at runtime: Oracle client and networking components were not found. These components are supplied by Oracle Corporation and are part of the Oracle Version 7.3.3 or later client software installation. Provider is unable to function until these components are installed. If so, try the following steps:

1. Log on to Windows as a user with Administrator privileges.
2. Launch Windows Explorer from the Start Menu and navigate to the `ORACLE_HOME` folder. To find this value, look up this variable in your machine environment variables through **My Compute**r > **Propertie**s > **Advanced** > **Environment Variables** > System Variables.
3. Right-click the `ORACLE_HOME` folder in Windows Explorer and select the **Properties** option from the drop down list. A Properties window appears.
4. Click the **Security** tab of the Properties window. Click the **Authenticated Users** item in the Name list (on Windows XP the Name list is called Group or user names).
5. Uncheck the **Read and Execute** box in the Permissions list under the Allow column (on Windows XP the Permissions list is called Permissions for Authenticated Users).
6. Recheck the **Read and Execute** box under the Allow column (this is the box you just unchecked).
7. Click the **Advanced** button and in the Permission Entries list make sure you see the Authenticated Users listed there with:

`Permission = Read & Execute Apply To = This folder, subfolders and files` If this is not the case, edit that line and make sure the **Apply onto dropdown** box is set to This folder, subfolders and files. This should already be set properly, but it is important to verify it.

1. Click the **OK** button until you close out all of the security properties windows. This may take a few seconds as permissions are applied.
2. Restart your computer to assure that these changes have taken effect.

3. Retest your application.

Additional Troubleshooting Steps

If dbrequest.aspx returns a 500 error (with ASP debugging enabled) and if the event viewer shows the following exception message: `Failed to access IIS metabase` try granting access using the following at the command line (sample version number): **Start** > **Run** > `C:\WINDOWS\Microsoft.NET\Framework\v2.0xyz\aspnet_regiis -i`

Context Services URL Message

`session.connection.protocol.sip.requesturi['context_services_url']`

If you get the above message when debugging VXML, you must update your IVRProfile to define the `context_services_url` parameter.

See IPD Runtime Configuration for details.

CTI Block Issues

Potential Issues are as follows:

1. A CTI block throws an error.unsupported.send exception:

    - Check if external messaging is enabled in MCP. CTI blocks use <send> and <receive> tags and these can be disabled in MCP configuration. Please refer to the *Genesys Voice Platform Deployment Guide*.

2. A CTI block throws an `error.com.genesyslab.composer.unsupported exception`:

    - Check that the CTI functionality your application is trying to use is supported in the scenario that is getting executed at runtime -- CTI via SIPServer or CTI via CTIConnector. Consult the Working with CTI Applications and CTI Blocks topics to see which features are supported in the two scenarios in each CTI block.

Debugging Failure

While debugging diagrams or code, if there is a error, check the following:

1. The Project is deployed on Tomcat or IIS.
2. Debugger preferences are setup correctly. Select **Window** > **Preferences** > **Composer** > **Debugging** > **GVP Debugger** and configure the GVP Debugger. Make sure the SIP phone IP address and MCP's IP address and ports are correct.
3. Delete any previous debug configurations by right-clicking the file > **Debug As** > **Debug Configuration**s. Delete existing debug configurations for the file. A new one will be created when debugging is initiated.
4. In the Project Explorer, check that the debugging-results folder exists. If not, create the folder and try debugging again.
5. While creating a .NET Composer project, if a non-default location is selected, debugging workflows or callflows may fail. Workaround: Give read permissions to all files in the Project from Windows Explorer so that they are accessible to the Debugger.

Deployment Failure on IIS

In a .NET Composer Project, if you try to deploy the Composer Project and it fails each time, check the following:

1. Is IIS installed on the machine?
    - Make sure IIS is installed and running on the machine that is running Composer.
2. If running on IIS 7.5 and Windows 7, does the account used for login have full administrative permissions? If not, the deployment will fail to access metadata with error: `location IIS://LocalHost/w3svc Failed`. This error appears either in the logs when you try to debug or in the **Project Properties** > **IIS Deployment** tab. Ensuring full administrative permissions is best accomplished by using one of two methods:
    - Log into your computer by using the local administrator account.
    - If logging in using an account with administrative permissions, but the account is not the local administrator account, open Composer by using the "Run as Administrator" option.
3. If running on IIS 7 (Windows Vista, Windows 2008) or IIS 7.5 (Windows 7), during the installation of IIS, IIS Metabase and IIS 6 configuration compatibility must be installed. See the section "Configuring in IIS Manager" in the Composer 8.1 Deployment Guide.
4. Are the IIS preferences configured correctly?
    - Make sure the post-installation configuration steps have been accomplished. Verify the IIS website port.
5. To address any potential deployment failures when using IIS, Genesys recommends disabling the User Account Control (UAC) for all Composer supported Windows operating systems (**Control Panel** -> **User Accounts** -> **Use User Account Control**).
6. Make sure IIS does not have any existing virtual directory of the .NET Composer Project name in the website where you want to deploy to.
7. Make sure the files in the Composer Project have read permissions for all users. The permissions can be set by navigating to the Project in Windows Explorer and using the Security tab in the Properties dialog of the folder.

DOTNet (.NET) Project Issues

```
1. If requests to backend pages in a .NET Composer Project are failing, it may be due to IIS
validation. IIS validates all requests for security reasons.
```

It is possible to disable validation. You can do this for a specific page by adding the following directive in the ASPX file: <%@ Page validateRequest="false" %> You can also disable validation for the entire project by adding this configuration to the `web.config` file: `<configuration> <system.web> <pages validateRequest="false" /> </system.web> </configuration>` Note that request validation is a security feature provided by ASP. Disabling it is at your own risk. See for details.

2. Error 500.24 - Internal Server Error If you are using IIS 7 or IIS 7.5 with Integrated Pipeline Mode, requests to backend pages in a .NET Composer Project may fail with 500.24 Internal Server Error. This occurs because ASP.NET Integrated mode is unable to impersonate the request identity in the pipeline stages. It is possible to ignore or workaround the 500.24 error.

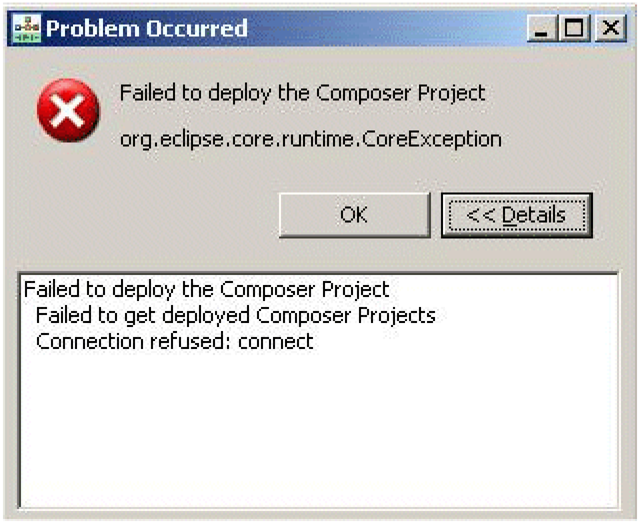
a. If your application does not rely on impersonation, edit the predefined `web.config` file within the DotNet Composer Project to set impersonate to false, `<system.web> <identity impersonate="false"/>`

b. If your application relies on impersonation, configure Impersonate in IIS 7 as suggested in this link. http://technet.microsoft.com/en-us/library/cc730708(v=WS.10).aspx c. If you want to ignore these errors edit the predefined `web.config` file to set `ValidateIntegratedModeConfiguration` property to false. http://msdn.microsoft.com/en-us/library/bb422433(v=vs.90).aspxhttp://msdn.microsoft.com/en-us/library/aa965174(v=vs.90).aspx

d. If you want to use the Composer predefined `web.config` file without making any new changes, set the Request-Processing mode of the Application pool to `classic` in your IIS. http://technet.microsoft.com/en-us/library/cc725564(v=ws.10).aspx

Failed to Deploy Message

The message below appears if trying to deploy a project and Tomcat is down or there is no connection to Tomcat. Check Tomcat preferences and start the service.



```
Failed to deploy the Composer Project

org.eclipse.core.runtime.CoreException
```

```
Failed to deploy the Composer Project
  Failed to get deployed Composer Projects
  Connection refused: connect
```

Installation and Uninstallation

When installing or uninstalling Composer in a Windows XP Professional or Windows Server 2003 Standard Edition environment, the prompt reboot may appear. After the reboot, the following message appears: `There are some pending operations and the system needs a reboot. The target computer does not meet some mandatory requirements.` Subsequent reboots result in the same behavior and you cannot install Composer on this machine. This is a result of pending reboots from other installations. To correct this situation:

1. Open the Registry editor.
2. Remove the following key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations`
3. Install Composer.

JSON objects and JavaScript keywords

If the JSON Objects used within the VXML document contain JavaScript reserved keywords (for example: double), NGI will throw an error.semantic exception.

For example, if the following occurs:

```
{'double':{'content':'40.015','XMLNS':'HTTP://WWW.WEBSERVICEX.NET/'}}
```

then accessing double.content in the above JSON Object will result in the following error:

```
exec_error SyntaxError: missing name after . operator JSONType.double.toSource()
```

```
event error.semantic:1
```

Therefore, responses from the requested URLs in Web Service and Web Request blocks should not contain any JavaScript reserved keywords.

ORS Compile Errors and Non-Escaped Characters

The condition expression for event-related properties in interaction process (IPD) and workflow diagrams are not XML-escaped when generating the SCXML code. Non-escaped XML special characters in the condition expression field will cause Orchestration Server compile errors at runtime. As a result, the following XML special characters in condition expressions should be escaped:

CHARACTER	REPLACEMENT
"	"
<	<
>	>
&	&

For an existing 8.1.3 or 8.1.4 diagram:

1. Go to the IPD diagram `Events` property or workflow diagram block `Exceptions` property. In the case of an IPD, when you select the IPD in the Project Explorer, a Properties view shows the Events property.
2. Check for XML special characters in the Condition expression field and escape them as shown in the list above. If the Body field SCXML expression includes some conditions (i.e. <if cond="somecondition">), the condition expression needs to be XML-escaped as well.
3. Save the diagram and generate code.

Note: Condition expressions provided by Composer (like for default IPD event handlers) are already XML-encoded.

Plugin Installation

When installing plugins into Composer, for example, the Genesys Rules Development Tools plugin, you may have to add an update site to Composer's installation preferences. Go to **Window** > **Preferences**, and select **Install/Update** > **Available Software Sites**. Ensure that the Galileo update site is present, and if not, add it. The URL for location should be http://download.eclipse.org/releases/galileo.

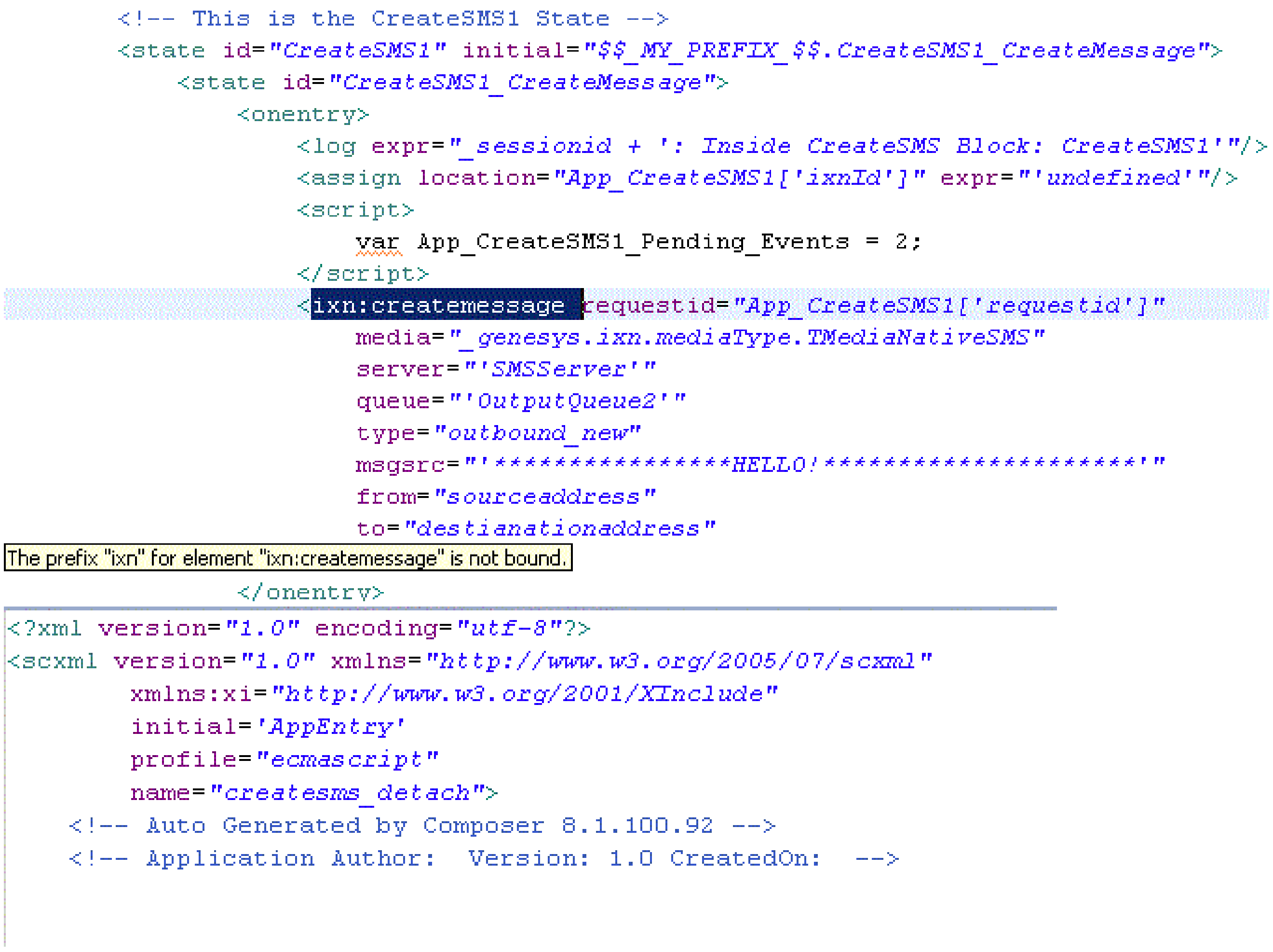
Proxy Configurations .NET Composer Projects

Request Form Error Message

When running a .NET project, you may encounter the error message: `A potentially dangerous Request.Form value was detected from the client` .NET has a built-in security check that will reject values that look like a potential attack. You can disable the validation using the configuration described in this page: http://www.asp.net/learn/whitepapers/request-validation.

SCXML Editor Element Not Bound Message

The SCXML editor may show element not bound errors for an SCXML tag while editing a generated workflow or sub-workflow SCXML file. The is caused by missing namespace declarations in the root <scxml> element of the page. For workflow and sub-workflow-generated SCXML files, this error can be safely ignored as namespace declarations for the enclosing IPD SCXML file will take effect at runtime and the current page's <scxml> element will be ignored by ORS.



```
        <!-- This is the CreateSMS1 State -->
        <state id="CreateSMS1" initial="$$_MY_PREFIX_$$.CreateSMS1_CreateMessage">
            <state id="CreateSMS1_CreateMessage">
                <onentry>
                    <log expr="_sessionid + ': Inside CreateSMS Block: CreateSMS1'"/>
                    <assign location="App_CreateSMS1['ixnId']" expr="'undefined'"/>
                    <script>
                        var App_CreateSMS1_Pending_Events = 2;
                    </script>
                    <ixn:createmessage requestid="App_CreateSMS1['requestid']"
                        media="_genesys.ixn.mediaType.TMediaNativeSMS"
                        server="'SMSServer'"
                        queue="'OutputQueue2'"
                        type="outbound_new"
                        msgsrc="'***************HELLO!*********************'"
                        from="sourceaddress"
                        to="destianationaddress"
The prefix "ixn" for element "ixn:createmessage" is not bound.
                </onentry>
```

```
<?xml version="1.0" encoding="utf-8"?>
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:xi="http://www.w3.org/2001/XInclude"
        initial='AppEntry'
        profile="ecmascript"
        name="createsms_detach">
    <!-- Auto Generated by Composer 8.1.100.92 -->
    <!-- Application Author:  Version: 1.0 CreatedOn:  -->
```

Slow Response Time

If Composer response time is slow, stop/disable anti-virus software that might be running on the system.

Stored Procedure Helper and DB Data Block

When **StoredProcedure** is selected for the Type Property Query Type in the DB Data block, you can open the Stored Procedure Helper dialog box.

If using the one of the database Project Templates, after selecting a stored procedure and clicking Execute, you may get a Problem Occurred exception message and a null exception may occur in the logs.

This may occur, for example, if the correct parameters are not passed.

Tomcat Service Failed to Start

Start Tomcat and **Stop Tomcat** toolbar buttons can be used to control the bundled Tomcat service from within the Integrated Development Environment. In some cases, using these buttons may display error messages.

```
Failure in Starting Tomcat Service. Composer Tomcat could not be started

Failure in Stopping Tomcat Service. Composer Tomcat could not be stopped
```

If you receive these error messages, close Eclipse (or Composer) and run `Eclipse.exe` (`Composer.exe`) as administrator by right clicking the file and selecting **Run as Administrator**. This provides administrator permission to the integrated development environment and enables it to start/stop the Tomcat service.

If the **Start Tomcat** button is clicked when the service is already started, this message is displayed:

```
Failure in starting Tomcat Service
Reason:
Tomcat Service could not be started.
Please check if Tomcat is already running.
```

Tomcat Windows Service

Composer bundles Tomcat 6.0 and deploys it as a Windows Service called ComposerTomcat. If you receive a message indicating that the Tomcat Service failed to start, please check the following:

1. From **Control Panel** > **Administrative Tools** > **Services**, check to see if the `ComposerTomcat` service is started.
2. If the service is not started, open up the log files in `${ComposerInstalledPath}/tomcat/logs` and look for an error that looks like: `java.net.BindException: Address already in use: JVM_Bind.`
3. If you see this error, it means that the port specified in the Tomcat configuration screen in the installer wizard is already in use. Uninstall Composer and reinstall using a different value for the Tomcat port.

Another place to check is the `composer_global.properties` in the installed location `<program files/GCTI/Composer 8.1/>`. It should show the correct Tomcat port number e.g. `TOMCAT_PORT=8082`

Note: The port number for Tomcat entered during Composer installation and the port number in Composer preferences should match.

Test Calls Do Not Work

If the test calls from Composer keep failing, check the following:

1. Do you keep getting Received 200 OK without receiving debugging control info for every test call made from Composer?
    - Make sure that the `vxmli.debug.enabled` parameter on the MCP is set to true. Follow the instructions in Post-Installation Configuration to verify the values.
2. Is the Tomcat service or IIS service running?
    - If you are using Tomcat, take the actions described in Tomcat Service Failed to Start. If no port conflict exists, try restarting Tomcat from Windows Services.
    - If you are using IIS, take the actions described in Deployment Failure on IIS. Try restarting IIS from Windows Services.
3. Has Debugger configuration been set as described in the Configure Tomcat and Debugger Settings cheat sheet, or has IIS been configured according to the Setting IIS Preferences cheat sheet?
    - Check the instructions in Post-Installation Configuration.
4. Is there a SIP Phone running on the configured host and port number?
    - Check your installation for a SIP Phone, and check the SIP Phone values in Post-Installation Configuration.
5. Can you make a test call directly from the SIP Phone?
    - If so, this confirms that the MCP and SIP Phone are working fine.
6. Is your SIP Phone running on the dedicated port?
    - If not, check your SIP Phone documentation to see how to configure it to run on a dedicated port. If the SIP Phone does not support this, switch to one of the SIP Phones like X-Lite which provides this capability.
7. Do you have a local firewall on the development server (for example, Windows Firewall on Windows XP/2003)?

Make sure that the following TCP ports have been opened:

- Tomcat port (this is generally set to port 8080). If you installed Tomcat on a different port, open its corresponding port in the firewall. IIS port (this is generally set to port 80). If you installed IIS on a different port, open its corresponding port in the firewall.
- UDP port on which your SIP Phone is running (by default this will be 5060 or 5070). Check your SIP Phone settings for the exact port number.
- RTP ports on which your SIP Phone will get the audio stream. Check your SIP Phone Help file

for details on this. Some SIP phones will auto-configure this during installation.

If you continue to run into problems with the firewall and calls are not successful, try turning off the firewall temporarily when making the test calls.

See GVP Debugging Limitations.

Upgrade Error Messages

Tomcat version mismatch errors

One or more of these errors may be observed:

- **Compilation errors in Eclipse console**

Errors seen on building the Composer Project.

```
 [javac] warning: C:\Program Files (x86)\Sun\JRE\7u5\lib\rt.jar(java/net/
ProtocolException.class):
 major version 51 is  newer than 50, the highest major version supported by this compiler.
 [javac] It is recommended that the compiler be upgraded.
 [javac] warning: C:\Program Files (x86)\Sun\JRE\7u5\lib\rt.jar(javax/net/ssl/
KeyManager.class):
 major version 51 is newer than 50, the highest major version supported by this compiler.
 [javac] It is recommended that the compiler be upgraded.
 [javac] Note: Some input files use or override a deprecated API.
 [javac] Note: Recompile with -Xlint:deprecation for details.
 [javac] Note: <your project path>\WEB-INF\src\org\apache\jsp\include\
getWebRequestData_jsp.java
 uses unchecked or unsafe operations.
 [javac] Note: Recompile with -Xlint:unchecked for details.
```

- **Tomcat logs**

```
 Tomcat logs error:-
 Tomcat localhost log file we get the following error:
 15-mei-2014 15:04:17 org.apache.catalina.core.StandardWrapperValve invoke SEVERE:
 Servlet.service() for servlet jsp threw exception
 java.lang.UnsupportedClassVersionError: com/genesyslab/studio/backendlogic/db/
CVDBBackendHandler :
 Unsupported major.minor version 51.0
 at java.lang.ClassLoader.defineClass1(Native Method)
 at java.lang.ClassLoader.defineClassCond(ClassLoader.java:632)
 at java.lang.ClassLoader.defineClass(ClassLoader.java:616)
 at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:141) at
org.apache.catalina.loader.WebappClassLoader.findClassInternal(WebappClassLoader.java:1819)
 at org.apache.catalina.loader.WebappClassLoader.findClass(WebappClassLoader.java:872)
 at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1327)
 at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1206)
 at org.apache.jasper.servlet.JasperLoader.loadClass(JasperLoader.java:128)
 at org.apache.jasper.servlet.JasperLoader.loadClass(JasperLoader.java:66)
```

Root Cause:

Composer 8.1.3 requires Tomcat 6 with JVM 1.7 (please check the *Composer Deployment Guide* for the most updated information). This is required both

- At design time while working on your application in the Composer/Eclipse Integrated Development Environment.

- At runtime when the exported application is deployed on production or test servers where Composer is not installed.

Resolution:

1. Ensure Tomcat JAVA version is set to the correct version as mentioned above or as per the latest information in the *Composer Deployment Guide*.
2. Restart Composer Tomcat
3. Clean the Project in Composer from the Project menu.
4. Build the project.

Note: If you continue to see errors, please ensure that the JVM version set as the default in the operating system (JAVA_HOME) is the same as the version being used in Tomcat (check version from Tomcat scripts/utilities).

java version mismatch and unsupported class version errors

After upgrading to Composer 8.1.3 from prior versions, java version mismatch and unsupported class version errors may appear in the console window. For example:

```
 15-mei-2014 15:04:17 org.apache.catalina.core.StandardWrapperValve invoke SEVERE:
Servlet.service()
 for servlet jsp threw exception java.lang.UnsupportedClassVersionError:
 com/genesyslab/studio/backendlogic/db/CVDBBackendHandler : Unsupported major.minor version
51.0
```

As a workaround, use the Composer workbench **Project Clean** to clean all the Projects. This will remove all the temporary jsp compiler-related files inside the WEB-INF folder files and do a new Composer Project build.

Validation Error upon publishing IPD

Sometimes, when publishing an IPD to the Config server for the first time, Composer shows a Validation Error. After a second publish, no error appears, and the publish completes successfully. The likely cause is that the Config Server does not like the data being written to it. Most likely the underlying SQL Server isn't configured to accept non-ascii input.

Web Service Block Issues

If you have problems using the Web Service block, follow the procedure below. Also see the Errors in WSDL Parsing section in the Web Service block topic.

Web Services Explorer

Before proceeding to the Web Service block, use the Web Services Explorer to run and test your Web Service.

- To access the Web Services Explorer from the diagram, right-click on the Web Service block and select Test with Web Services Explorer. The Web Services Explorer view will open and provide support for browsing and invoking Web services natively from the WSDL supplied in the Web Service block.

Setting Properties in the Web Service Block

When you provide the WSDL URL in the Service URL property, Composer will try to access the URL and parse it to populate the drop-down fields for the remaining properties. If the following error occurs, there is a problem in parsing the WSDL: An error occurred while parsing the WSDL url WSDLException: faultCode=OTHER_ERROR: Unable to resolve imported document at 'null".

1. Verify that the WSDL URL entered is valid and prefixed with http://.
2. If you are behind a proxy server, configure the proxy settings by going to Window > Preferences, then expand General and select Network Connections.
3. Select Manual proxy configuration and add values for HTTP proxy and Port.
4. After you have chosen the available Service and operations which you want to invoke, set the required input Parameters, if any, in the Input Parameters dialog box.
5. You can use the Output Result dialog box to map the Web Service response keys in to AppState variables by setting the Map Output Values to Variables property to true; otherwise the entire Web Service response will be assigned to a variable.
6. If required, set the Web Service authentications (only basic authentication is supported).

Execution Errors when the Web Service Block is Executed in Your Call

If you see the error subdialog_return :|event|com.genesys.studio.webservice.badFetch in the call traces while executing the Web Service block, check the following:

1. If your Tomcat Web Server is behind a proxy server, configure Proxy settings in Tomcat.

Proxy settings have to be configured in Tomcat for the backend pages to access the Web when Web Request and Web Service blocks are used. To configure proxy settings in Tomcat, add the following lines into the catalina.properties file under the $ComposerInstalledDir$\tomcat\conf\ folder: http.proxyHost=hostip http.proxyPort=portofProxy http.proxyUser=username http.proxyPassword=password

1. Finally, restart the CV80Tomcat service.
2. Use the Web Services Explorer to test whether the Web Service is a valid one.
3. Verify the Service URL and Service End Point values.
4. Verify that the supplied Input Parameters and the data type of the values are valid.
5. Verify whether the Web Service requires any basic authentications and if needed, configure these authentications in the Security category of the Web Service block.

How to Configure Connection Timeout and Read Timeout See Connection and Read Timeout Configuration.</nowiki>

Custom SOAP Envelope Feature - Fetch Failed in Weblogic

Weblogic returns null for the `ServletContext.getRealPath()` method when web applications are deployed as WAR files. You must manually enable this in Weblogic since the Composer Custom SOAP Envelope property uses the `getRealPath()` method.

Here are the steps to enable RealPath in a Weblogic application server:

1. Go to the server admin console->**Domain**-> Web applications.
2. Click the check box for **Archived Real Path Enabled**. This should make an entry into the domain `config.xml` as below.

```
<web-app-container>
<show-archived-real-path-enabled>true</show-archived-real-path-enabled>
</web-app-container>
```

A second option is at the web application level by updating `weblogic.xml` as below:

```
<container-descriptor>
<show-archived-real-path-enabled>true</show-archived-real-path-enabled>
</container-descriptor>
```

The value of `<show-archived-real-path-enabled>` set in the web application has precedence over the value set at the domain level. The default value of this property is `false`.

Workspace in Use or Cannot be Created

The above message may display if multiple Composer instances are sharing the same workspace, causing an access violation. Use a different workspace or ensure that only single instance of Composer accesses a workspace.

Workspace Files Not in Sync

When running into files in your Project that are out of sync, there is a Refresh automatically preferences option to avoid manually pressing F5 on the Workspace resource.

Refresh Automatically

To enable the preference:

1. Go to **Window** > **Preferences**.
2. Expand the **General** tree item and click on the **Workspace** item.
3. Mark the check box **Refresh automatically**.
4. Click **OK** to close the Preferences dialog.

To avoid refresh issues Composer recommends the following when dealing with Project resource files:

- 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 from the file system onto Composer's Project Explorer.

Links to Useful Resources

You may find information and resources at the following locations to be useful as you use Eclipse and Composer. For Genesys Product Documentation Each product has its own documentation for online viewing at the Genesys Technical Support website or on the Documentation Library DVD, which is available from Genesys upon request.

- Universal Routing Server (URS)—which enables intelligent distribution of voice and multimedia interactions throughout the enterprise. You may need this link if transitioning from IRD routing strategies to Composer routing workflows.

- Orchestration Server (ORS)—an open standards-based platform with an SCXML engine, which enables the customer service process. See the *Orchestration Server Developer's Guide* for information on ORS Functional Modules and Extensions.

For Eclipse

- Eclipse website: http://www.eclipse.org/
- Ganymede Documentation: http://help.eclipse.org/ganymede/index.jsp
- Workbench User Guide: http://help.eclipse.org/ganymede/nav/0

For VXML

- VXML 2.1 Draft Specification: http://www.w3.org/TR/voicexml21/
- VoiceXML website: http://www.voicexml.org
- For information on the ECMAscript functions that can be used in Expression Builder:http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

For IBM WebSphere

- Installing WebSphere Application Server 6.1: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.express.doc/info/exp/ae/tins_custome_61.html
- Installing application files with the console: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.express.iseries.doc/info/iseriesexp/ae/trun_app_instwiz.html
- Configuring HTTP Proxy Information: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.express.doc/info/exp/ae/twbs_configaddhttppropertiesadmin.html

For Web Services Web Services Description Language (WSDL) website: http://www.w3.org/TR/wsdl SCXML Reference Documents State Chart XML (SCXML): State Machine Notation for Control Abstraction: http://www.w3.org/TR/scxml/ ECMAScript Language Specification: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf Standard ECMA-327: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-327.pdf

Composer Product Videos

This page contains Composer product videos. Stay tuned for more videos to come. To request a video, please email Techpubs.webadmin@genesyslab.com.

Installing Composer 8.1.3

Below is a video tutorial on installing Composer 8.1.3 on Windows in an Eclipse 4.2 environment.
Note: If Eclipse 3.7 is installed, there is a small extra step described in the Installation chapter of the *Composer 8.1.3 Deployment Guide*.

Link to video

Using the Database Blocks

Below is a video tutorial on using the Database Blocks.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```

Link to video

Debugging VoiceXML Applications

Below is a video tutorial on debugging VoiceXML applications.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```

Link to video

Deploying a Composer Application to a Web Server

Below is a video tutorial on exporting and deploying a Composer application to a web server.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```

Link to video

Creating a Simple Grammar

Below is a video tutorial on building a simple grammar with the Grammar Menu block.

```
Important Note: While the interface for Composer in this video is from release 8.0.1,
the steps are the basically the same for subsequent releases.
```

Link to video