GENESYS

AN ALCATEL-LUCENT COMPANY

Composer 8.0

Routing Applications

User's Guide

About Genesys

Alcatel-Lucent's Genesys solutions feature leading software that manages customer interactions over phone, Web, and mobile devices. The Genesys software suite handles customer conversations across multiple channels and resources—self-service, assisted-service, and proactive outreach—fulfilling customer requests and optimizing customer care goals while efficiently using resources. Genesys software directs more than 100 million customer interactions every day for 4000 companies and government agencies in 80 countries. These companies and agencies leverage their entire organization, from the contact center to the back office, while dynamically engaging their customers. Go to www.genesyslab.com for more information.

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. For more information, contact your sales representative.

Notice

Although reasonable effort is made to ensure that the information in this document is complete and accurate at the time of release, Genesys Telecommunications Laboratories, Inc., cannot assume responsibility for any existing errors. Changes and/or corrections to the information contained in this document may be incorporated in future versions.

Your Responsibility for Your System's Security

You are responsible for the security of your system. Product administration to prevent unauthorized use is your responsibility. Your system administrator should read all documents provided with this product to fully understand the features available that reduce your risk of incurring charges for unlicensed use of Genesys products.

Trademarks

Technical Support from VARs

If you have purchased support from a value-added reseller (VAR), please contact the VAR for technical support.

Technical Support from Genesys

If you have purchased support directly from Genesys, please contact Genesys Technical Support at the regional numbers provided on page 11. For complete contact information and procedures, refer to the ***Genesys Technical Support Guide***.

Ordering and Licensing Information

Complete information on ordering and licensing Genesys products can be found in the ***Genesys Licensing Guide.***

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 80composer_us-ra_10-2010_v8.0.401.00

Table of Contents

Chapter 3

Chapter 4

Chapter 5

Supplements

Index

List of Procedures

Preface

Welcome to the *Composer 8.0 Routing Applications User's Guide.* Composer is an Integrated Development Environment used to develop applications for both Genesys Voice Platform (GVP) and Genesys Universal Routing. This guide focuses on creating applications for Universal Routing.

This document is valid only for the 8.0.4 release of this product.

Note: The 8.0.1 version of this product was known as Composer Voice, as it was used only to develop applications for GVP.

This preface contains the following sections:

For information about related resources and about the conventions that are used in this document, see the supplementary material starting on page 69.

About Composer

An Eclipse-based (www.eclipse.org) Integrated Development Environment, Composer provides for drag-and-drop environment for developing:

- Voice applications for Genesys Voice Platform (GVP) 8.1—a software suite that unifies voice and web technologies to provide a complete solution for customer self-service or assisted service.
- Routing applications for Genesys Universal Routing 8.0 platform, which includes:
  - Universal Routing Server (URS)—which enables intelligent distribution of voice and multimedia interactions throughout the enterprise.
  - Orchestration Server—an open standards-based platform with an SCXML engine, which enables you to manage conversations with customers and the customer service process.

Note: This guide focuses on the creation of applications for the Genesys Universal Routing platform.

Intended Audience

This document is intended for routing application developers, both technical and non-technical. Its primary goal is to introduce you to the Composer interface and the process of building routing applications. The information presented here is at a high level to facilitate a conceptual understanding. The *Composer 8.0.x Help* provides the necessary detail.

This document has been written with the assumption that you have a basic understanding of:

- Computer-telephony integration (CTI) concepts, processes, terminology, and applications
- Network design and operation
- Your own network configurations
- Genesys Framework architecture
- Genesys Universal Routing

Familiarity with State Chart Extensible Markup Language (SCXML) is a plus.

Note: Composer provides a wide range of tools to satisfy the needs of a diverse end user population. This guide assumes you are a non-technical user who does not write SCXML code who wishes to build routing applications by using Composer's diagram editor.

If you have already used Universal Routing's Interaction Routing Designer (IRD) to build routing strategies, you will find similarities with Composer. While IRD uses a proprietary language to build routing strategies, Composer is an open system that uses SCXML to build routing strategies, which it calls *routing workflows.*

Making Comments on This Document

If you especially like or dislike anything about this document, feel free to e-mail your comments to Techpubs.webadmin@genesyslab.com.

You can comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this document. Please limit your comments to the scope of this document only and to the way in which the information is presented. Contact your Genesys Account Representative or Genesys Technical Support if you have suggestions about the product itself.

When you send us comments, you grant Genesys a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Contacting Genesys Technical Support

If you have purchased support directly from Genesys, contact Genesys Technical Support at the following regional numbers:

Region	Telephone	E-Mail
North America and Latin America	+888-369-5555 (toll-free) +506-674-6767	support@genesyslab.com
Europe, Middle East, and Africa	+44-(0)-1276-45-7002	support@genesyslab.co.uk
Asia Pacific	+61-7-3368-6868	support@genesyslab.com.au
Before contacting technical support, refer to the *Genesys Technical Support Guide* for complete contact information and procedures.		

Region	Telephone	E-Mail
Malaysia	1-800-814-472 (toll-free) +61-7-3368-6868	support@genesyslab.com.au
India	000-800-100-7136 (toll-free) +91-(022)-3918-0537	support@genesyslab.com.au
Japan	+81-3-6361-8950	support@genesyslab.co.jp
Before contacting technical support, refer to the *Genesys Technical Support Guide* for complete contact information and procedures.		

Document Change History

This is the first release of the *Composer 8.0 Routing Applications User's Guide*. In the future, this section will list topics that are new or have changed significantly since the first release of this document.

Chapter

1 SCXML-Based Routing Applications

This chapter introduces SCXML-based routing applications created in Composer. It introduces the concept of routing customer interactions, such as phone calls or e-mails, to targets, such as agents or agent groups, with the skills to handle those interactions. It also introduces interaction processing diagrams, routing workflows, the workflow diagram-building blocks, Composer Projects, and the SCXML code editor.

This chapter contains the following sections:

What is Routing?

From a Genesys standpoint, routing is the process of sending an interaction to a target. For example, routing an incoming telephone call from a customer requesting information on Product A to an agent knowledgeable about Product A. In order for such an interaction to reach the appropriate target, the interaction must undergo various types of processing between the time it arrives at the contact center and the selection and use of the appropriate target. You specify the various types of processing that must occur through a *routing application*, which, in case of Composer, is deployed on an web application server.

Routing Applications

A Composer routing application is comprised of one or more *routing workflows*, which are similar to routing strategies created in Universal Routing's Interaction Routing Designer (IRD).

Using Composer, you can create routing applications using the following methods:

- By writing SCXML code in Composer's code editor or more easily, as described in this guide:
- By creating workflow diagrams, which require you to place, connect, and configure *blocks*.

When loaded on and triggered from a routing point, a routing workflow tells Universal Routing Server how to handle and where to direct interactions under different circumstances (a process which may involve other servers).

Routing Application Structure

At the top level of a Composer routing application is an interaction process diagram (IPD). Figure 6 on page 24 shows an example IPD.

Interaction Process Diagrams

In summary, an interaction process diagram or IPD:

- Functions as the starting SCXML page for a routing application.
- Is automatically created when you start a new Project.
- Provides a high-level view of interaction processing flow (see "High-Level Interaction Flow" on page 61 for an example).
- References media servers for incoming interactions.
- Processes those interactions by moving them through interaction queues, workflow strategies for specialized processing, and (optionally) workbins.

Similar to an IRD business process, an IPD defines what happens to customer interactions from the point of arrival at your contact center to the point of completion (usually in the form of a response to the customer).

IPD Blocks

An IPD can contain the following types of blocks:

- Interaction Queue blocks (as described in "Defining an Interaction Queue" on page 71).
- Media Server blocks, (as described in "Adding a Media Server Block" on page 68).
- Workflow blocks (as described in "Adding a Workflow Block" on page 79) referencing a workflow strategy.
- Workbin blocks referencing a temporary storage place for interactions on the agent's desktop.

When processing multimedia interactions, an IPD typically starts with a Media Server block and continues with one or more Interaction Queue, Workflow, and/or Workbin blocks. These IPD-building blocks are available on Composer's `Palette` tab when an IPD is in focus (see Figure 6 on page 24).

Note: When routing voice interactions only, IPDs do not use Media Server and Interaction Queue blocks.

Workflow Diagram-Building Blocks

The workflow-diagram building blocks are the next level of blocks in an IPD. A Workflow block points to a workflow resource (see Figure 8 on page 27), which is either:

- a workflow diagram comprised of the diagram building blocks described in Appendix A, "Composer Blocks," on page 103 or
- an SCXML file, which can be created in Composer's code editor (see Figure 4 on page 20).

This document describes how to create routing applications comprised of IPDs that contain Workflow blocks pointing to workflow diagrams. The workflow diagram-building blocks are available on Composer's `Palette` tab when a workflow diagram is in focus (see Figure 8 on page 27).

Workflow Block Categories

The `Palette` tab groups the workflow diagram-building blocks into the following categories:

- `Flow Control`
- `Routing`
- `Voice Treatment`
- `Server-Side`
- `Context Services`
- `eServices`

Appendix A, “Composer Blocks,” on page 103 summarizes the workflow diagram-building blocks in each of the above categories. The *Composer 8.0.4 Help* details the properties of each block, which are what you configure.

Application Element Relationships

Figure 1 depicts the relationship of the various routing application elements discussed so far.

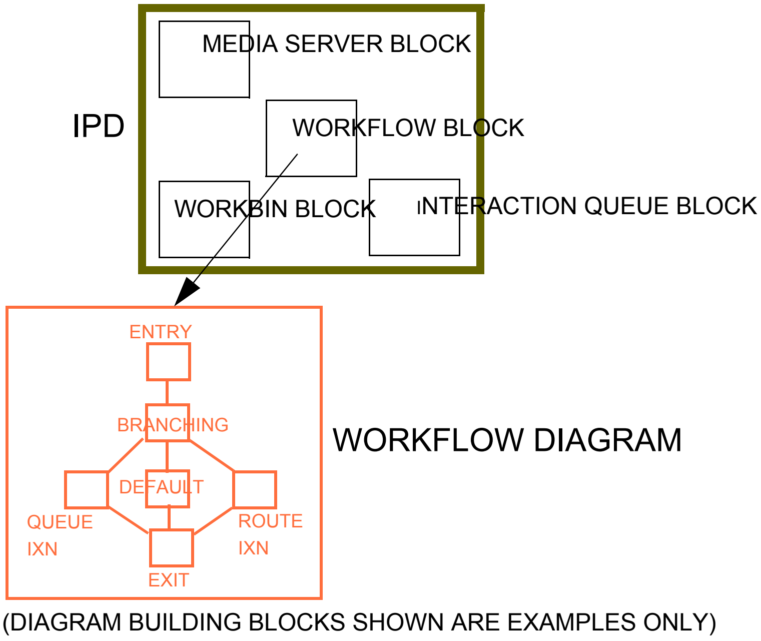


Figure 1: Workflow Block in IPD Pointing to Workflow Diagram

Moving Interactions Between IPDs

You cannot explicitly link IPDs together as there is no higher-level diagram in Composer that shows IPDs as single blocks and then allows you to interconnect them. You can, however, cause interactions to move from one IPD to another IPD through the use of *interaction queues*. This is similar to IRD’s method of linking business processes through interaction queues.

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.

Figure 2 illustrates this graphically.

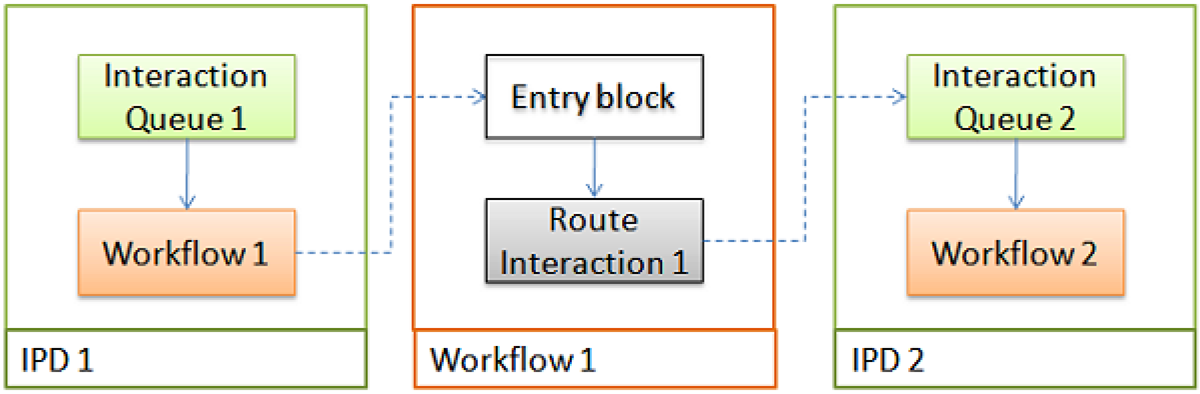


Figure 2: Moving Interactions Between IPDs

For more information, see the “Linking IPDs with Workflows” topic in the *Composer 8.0.4 Help*.

Projects

To organize all the routing application elements just discussed, Composer uses a *Project* to contain everything related to a single routing application. A `Project Explorer` on the upper left of the Composer window contains all the Projects in your workspace (see Figure 3).

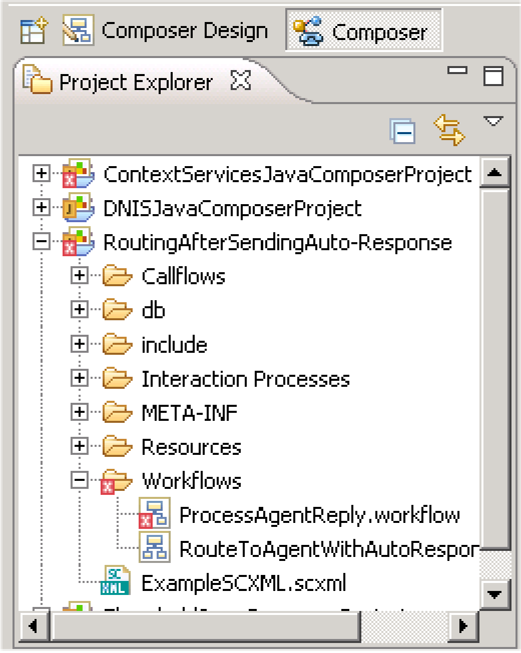


Figure 3: Project Explorer

Note: As described in Procedure: Creating a new Project, on page 62, when you create a new Project, Composer automatically creates a new IPD with the name `default.ixprocess`, which you can change (see Figure 31 on page 64).

Project Workspace

Within Composer, the term *workspace* refers to 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 that appears when you first start Composer gives the option of changing the workspace to a different location. New Projects created in Composer will be created under this workspace as subfolders.

Project Folders

A `Route` or `Integrated Voice and Route` type `Java Composer` Project (see Figure 28 on page 62) will contain some or all of these sub-folders depending on the type of Project:

- `Callflows` – Folder for storing all the callflow diagrams (`.callflow` files) as Composer allows you to create integrated Voice and Route Projects.

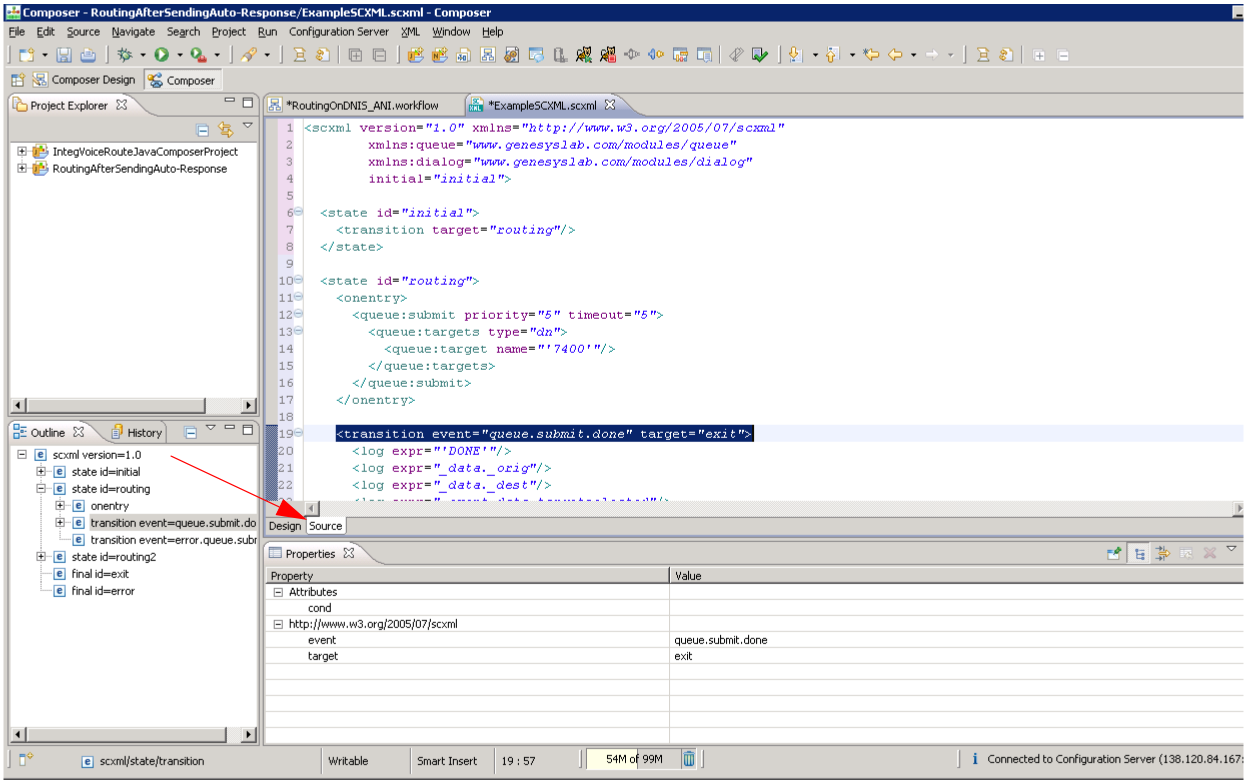
  Note: Callflow diagrams are associated with VXML-based applications created for Genesys Voice Platform (GVP), as described in the *Composer 8.0.x Help*.

- `db` – Database connection properties and`.sql` files are stored here.
- `include` – Composer-provided standard include files used by Backend logic blocks.
- `lib` – Folder for external dependency libraries such as JAR files.
- `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.
- `Interaction Processes` – Folder for storing all the interaction process diagrams (`.ixnprocess` files).
- `Resources` – Folder for the audio and grammar resources. Used only for VXML voice applications as described below.
- `Workflows` – Folder for storing all the workflow diagrams (.workflow files).
- `Scripts` – Folder for user-written ECMAScript.
- `src-gen` – Folder for the code generated SCXML or VXML files. This folder is also used to store any hand-coded VXML or SCXML files that may be part of a Project.
- `src` – Folder for custom code such as backend logic pages written by the user.

SCML Code Editor

For those who prefer to create routing applications by writing their own SCXML code, Composer provides a rich editor with use case templates.

Figure 4 shows example SCXML code in the `Source` tab of the code editor.

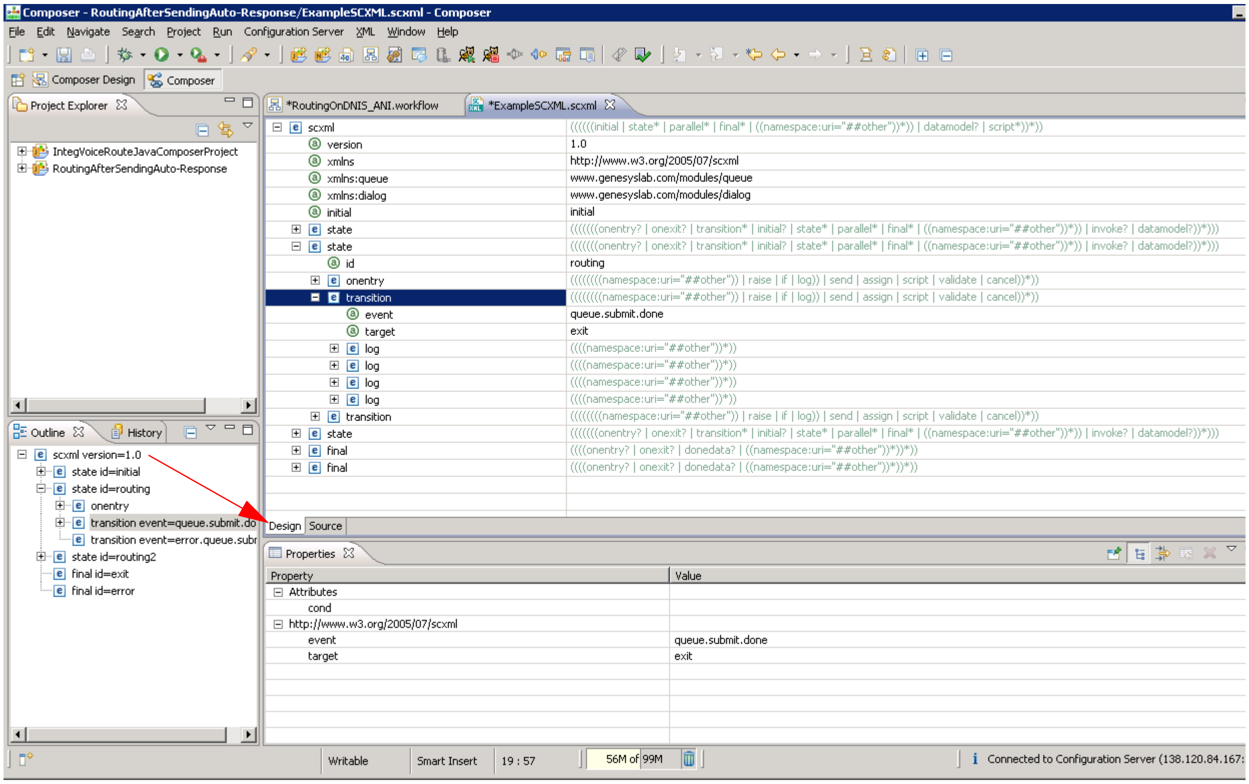

Composer - RoutingAfterSendingAuto-Response/ExampleSCXML.scxml - Composer
File Edit Source Navigate Search Project Run Configuration Server XML Window Help
Composer Design
Composer
Project Explorer
IntegVoiceRouteJavaComposerProject
RoutingAfterSendingAuto-Response
*RoutingOnDNIS_ANI.workflow
*ExampleSCXML.scxml
Outline
History
scxml version=1.0
state id=initial
state id=routing
onentry
transition event=queue.submit.do
transition event=error.queue.subr
state id=routing2
final id=exit
final id=error
Design
Source
Properties
scxml/state/transition
Writable
Smart Insert
19 : 57
54M of 99M
Connected to Configuration Server (138.120.84.167:

```
<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 priority="5" timeout="5">
        <queue:targets type="dn">
          <queue:target name="'7400'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_data._orig"/>
      <log expr="_data._dest"/>
```

Property	Value
Attributes	
cond	
http://www.w3.org/2005/07/scxml	
event	queue.submit.done
target	exit

Figure 4: Composer SCXML Code Editor, Source View

Figure 5 shows the `Design` tab of the code editor.


Composer - RoutingAfterSendingAuto-Response/ExampleSCXML.scxml - Composer
File Edit Navigate Search Project Run Configuration Server XML Window Help
Composer Design
Composer
Project Explorer
IntegVoiceRouteJavaComposerProject
RoutingAfterSendingAuto-Response
*RoutingOnDNIS_ANI.workflow
*ExampleSCXML.scxml
Outline
History
Design
Source
Properties
Writable
Smart Insert
19 : 57
56M of 99M
Connected to Configuration Server (138.120.84.167:

scxml	((((((initial | state* | parallel* | final* | ((namespace:uri="##other"))*)) | datamodel? | script*))*))
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	(((((((onentry? | onexit? | transition* | initial? | state* | parallel* | final* | ((namespace:uri="##other"))*)) | invoke? | datamodel?))*)))
state	(((((((onentry? | onexit? | transition* | initial? | state* | parallel* | final* | ((namespace:uri="##other"))*)) | invoke? | datamodel?))*)))
id	routing
onentry	((((((((namespace:uri="##other")) | raise | if | log)) | send | assign | script | validate | cancel))*))
transition	((((((((namespace:uri="##other")) | raise | if | log)) | send | assign | script | validate | cancel))*))
event	queue.submit.done
target	exit
log	((((namespace:uri="##other"))*))
log	((((namespace:uri="##other"))*))
log	((((namespace:uri="##other"))*))
log	((((namespace:uri="##other"))*))
transition	((((((((namespace:uri="##other")) | raise | if | log)) | send | assign | script | validate | cancel))*))
state	(((((((onentry? | onexit? | transition* | initial? | state* | parallel* | final* | ((namespace:uri="##other"))*)) | invoke? | datamodel?))*)))
final	((((onentry? | onexit? | donedata? | ((namespace:uri="##other"))*))*))
final	((((onentry? | onexit? | donedata? | ((namespace:uri="##other"))*))*))

scxml version=1.0
state id=initial
state id=routing
onentry
transition event=queue.submit.do
transition event=error.queue.subr
state id=routing2
final id=exit
final id=error

Property	Value
Attributes	
cond	
http://www.w3.org/2005/07/scxml	
event	queue.submit.done
target	exit

Figure 5: Composer SCXML Code Editor, Design View

You can view and work directly with source code using standard Eclipse text editing features. Features include:

- Smart double-clicking behavior.
- Context-assisted help when typing tags. Also context-assisted help for attributes of a tag upon pressing Space inside a tag.
- New SCXML documents are created with <scxml> as the top level element with the corresponding schema and namespace specifications.
- Ability to edit tag attribute values from the Properties view.
- Basic editor actions are supported: Cut, Copy, Paste, Save, Save as, Undo, Redo, Search and Replace.
- Syntax highlighting.
- Show and hide Line numbers.
- Add/Remove Bookmark and To-Do markers.
- Task tag feature to auto scan To-Do comments in the code.
- Comparing and reverting to local file history.
- Spell checking by showing yellow squiggly line markers.

Chapter

2 Composer GUI

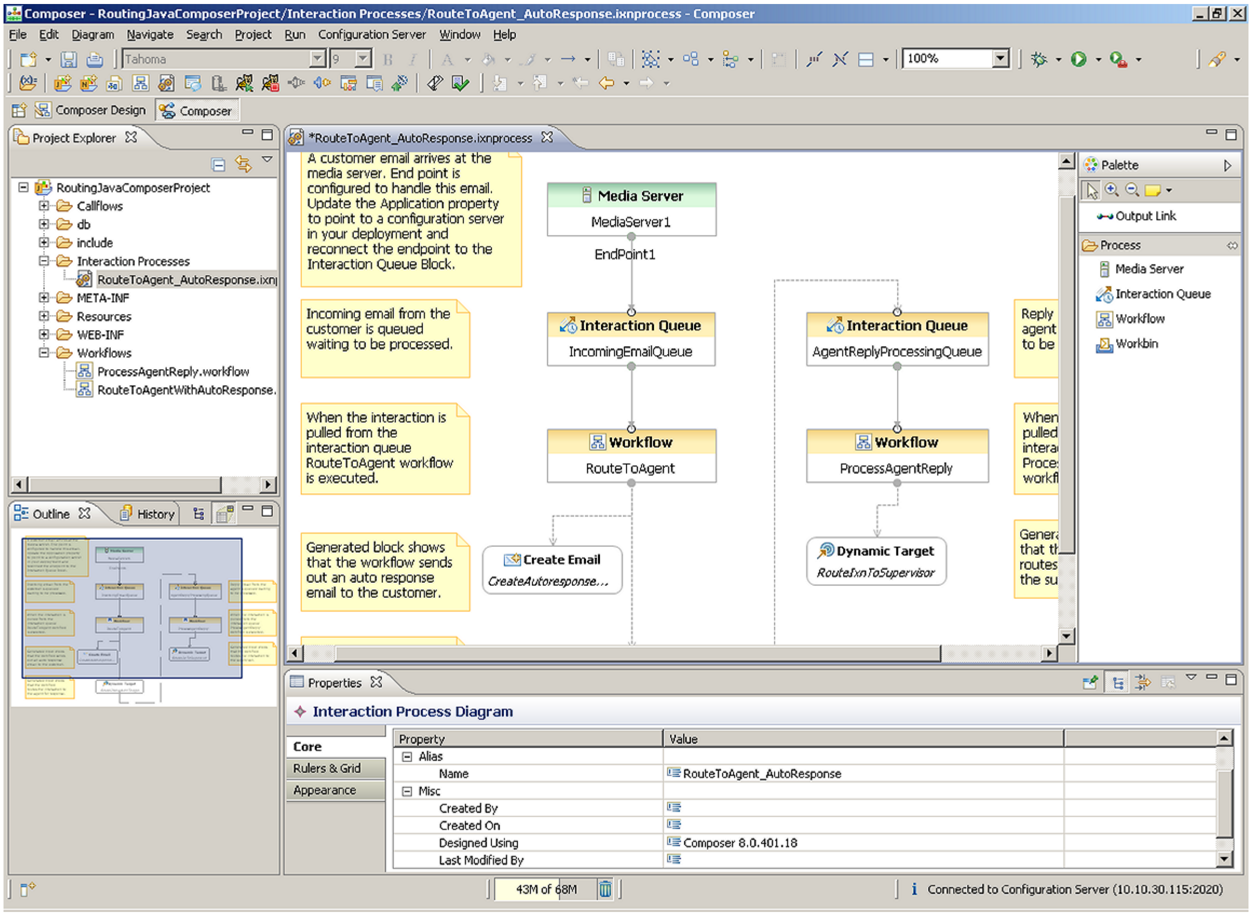
This chapter introduces the main parts of the Composer GUI used for building routing applications. This contains the following sections:

Diagram Editor

This section introduces the Composer's diagram editor, which you use when creating workflow diagrams.

IPD in Canvas Area

Figure 6 shows an example interaction process diagram (as described on page 14) in Composer's central *canvas* area. Note the Media Server, Workflow, and Interaction Queue blocks that comprise the IPD,


Composer - RoutingJavaComposerProject/Interaction Processes/RouteToAgent_AutoResponse.ixnprocess - Composer
File Edit Diagram Navigate Search Project Run Configuration Server Window Help
Tahoma
100%
Composer Design
Composer
Project Explorer
RoutingJavaComposerProject
Callflows
db
include
Interaction Processes
RouteToAgent_AutoResponse.ixnp
META-INF
Resources
WEB-INF
Workflows
ProcessAgentReply.workflow
RouteToAgentWithAutoResponse.
Outline
History
*RouteToAgent_AutoResponse.ixnprocess
Media Server
MediaServer1
EndPoint1
Interaction Queue
IncomingEmailQueue
Interaction Queue
AgentReplyProcessingQueue
Workflow
RouteToAgent
Workflow
ProcessAgentReply
Create Email
CreateAutoresponse...
Dynamic Target
RouteIxnToSupervisor
Reply agent to be
When pulled intera Proce workfl
Genera that th routes the su
Palette
Output Link
Process
Media Server
Interaction Queue
Workflow
Workbin
Properties
Interaction Process Diagram
Core
Rulers & Grid
Appearance
43M of 68M
Connected to Configuration Server (10.10.30.115:2020)

A customer email arrives at the media server. End point is configured to handle this email. Update the Application property to point to a configuration server in your deployment and reconnect the endpoint to the Interaction Queue Block.

Incoming email from the customer is queued waiting to be processed.

When the interaction is pulled from the interaction queue RouteToAgent workflow is executed.

Generated block shows that the workflow sends out an auto response email to the customer.

Property	Value
⊟ Alias	
Name	RouteToAgent_AutoResponse
⊟ Misc	
Created By	
Created On	
Designed Using	Composer 8.0.401.18
Last Modified By	

Figure 6: Example Interaction Process Diagram

Note: *Canvas* is part of the Eclipse standard terminology (see "About Composer" on page 10).

Note the following in Figure 6:

- The name of the IPD appears in the tab above the canvas. In this example, the name is `RouteToAgent_Autoresponse.ixprocess`.

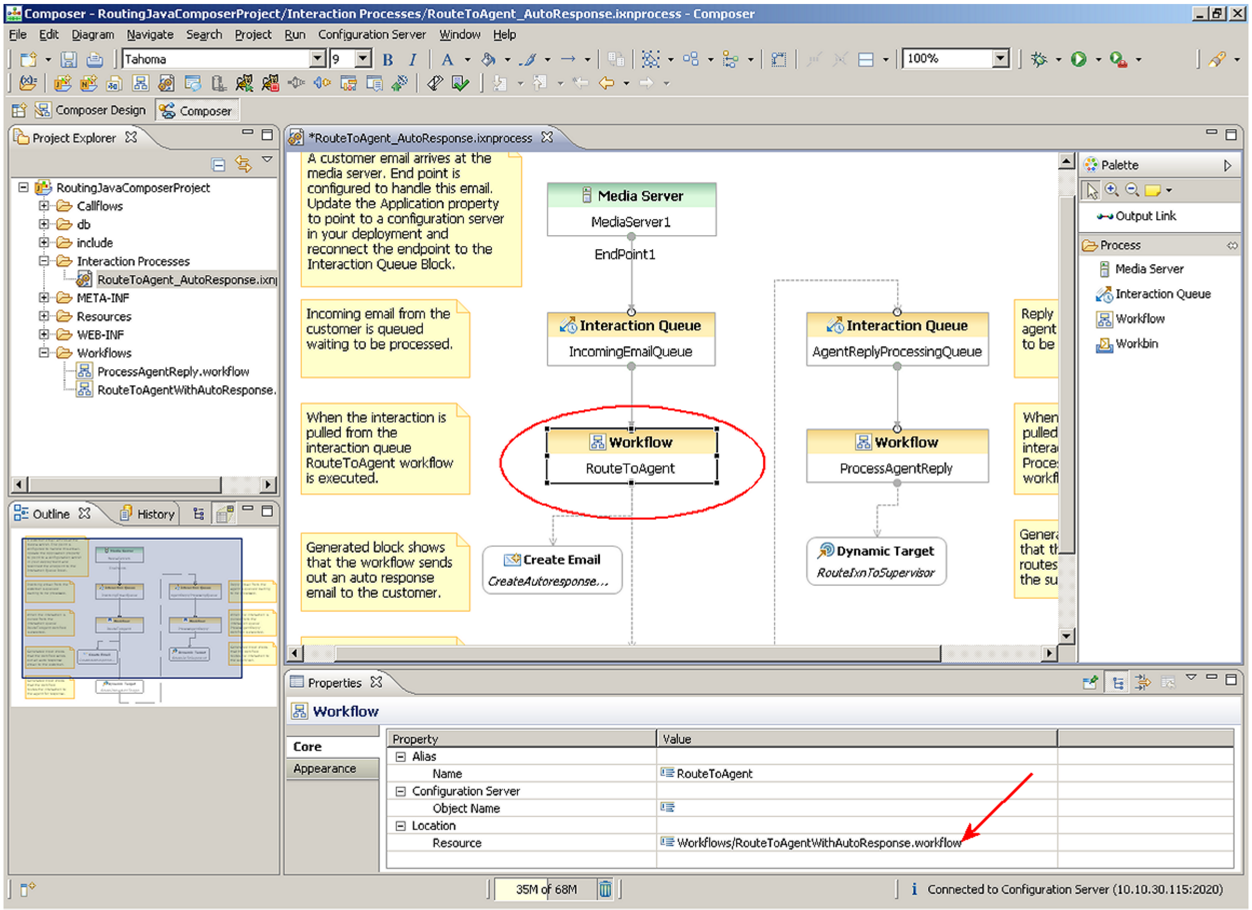
- On the left, a `Project Explorer` contains all the files associated with the routing application.

Note: Genesys recommends grouping a single application under a single Project as the best practice and not grouping multiple applications. When deploying a Composer Project to an application server, the whole Project is deployed as a single `*.war` file.

- Underneath the `Project Explorer` is the `Outline` view, which is useful as a point of reference when working with large workflows.
- On the right is a *pallete* of blocks.
  - When an IPD is in focus, the palette contains the IPD blocks described on page 15: Media Server, Interaction Queue, Workflow and Workbin.
  - When a workflow diagram is in focus, the pallete contains the workflow diagram-building blocks described in Appendix A, "Composer Blocks," on page 103. Figure 8 on page 27 shows an example.
- Underneath the canvas is the `Properties` view. It shows and gives access to the fields that can be configured for the selected block or diagram.
  - In Figure 6 on page 24, a block is not selected so the `Properties` view shows general fields for the diagram; in this case an interaction process diagram instead of a workflow diagram.

Properties View

Assume you click the `RouteToAgent` Workflow block in Figure 6 on page 24 so it is selected. The `Properties` view at the bottom now shows the fields for the selected Workflow block. (see Figure 7).



Composer - RoutingJavaComposerProject/Interaction Processes/RouteToAgent_AutoResponse.ixnprocess - Composer

A customer email arrives at the media server. End point is configured to handle this email. Update the Application property to point to a configuration server in your deployment and reconnect the endpoint to the Interaction Queue Block.

Incoming email from the customer is queued waiting to be processed.

When the interaction is pulled from the interaction queue RouteToAgent workflow is executed.

Generated block shows that the workflow sends out an auto response email to the customer.

Property	Value	
⊟ Alias		
Name	RouteToAgent	
⊟ Configuration Server		
Object Name		
⊟ Location		
Resource	Workflows/RouteToAgentWithAutoResponse.workflow	

35M of 68M

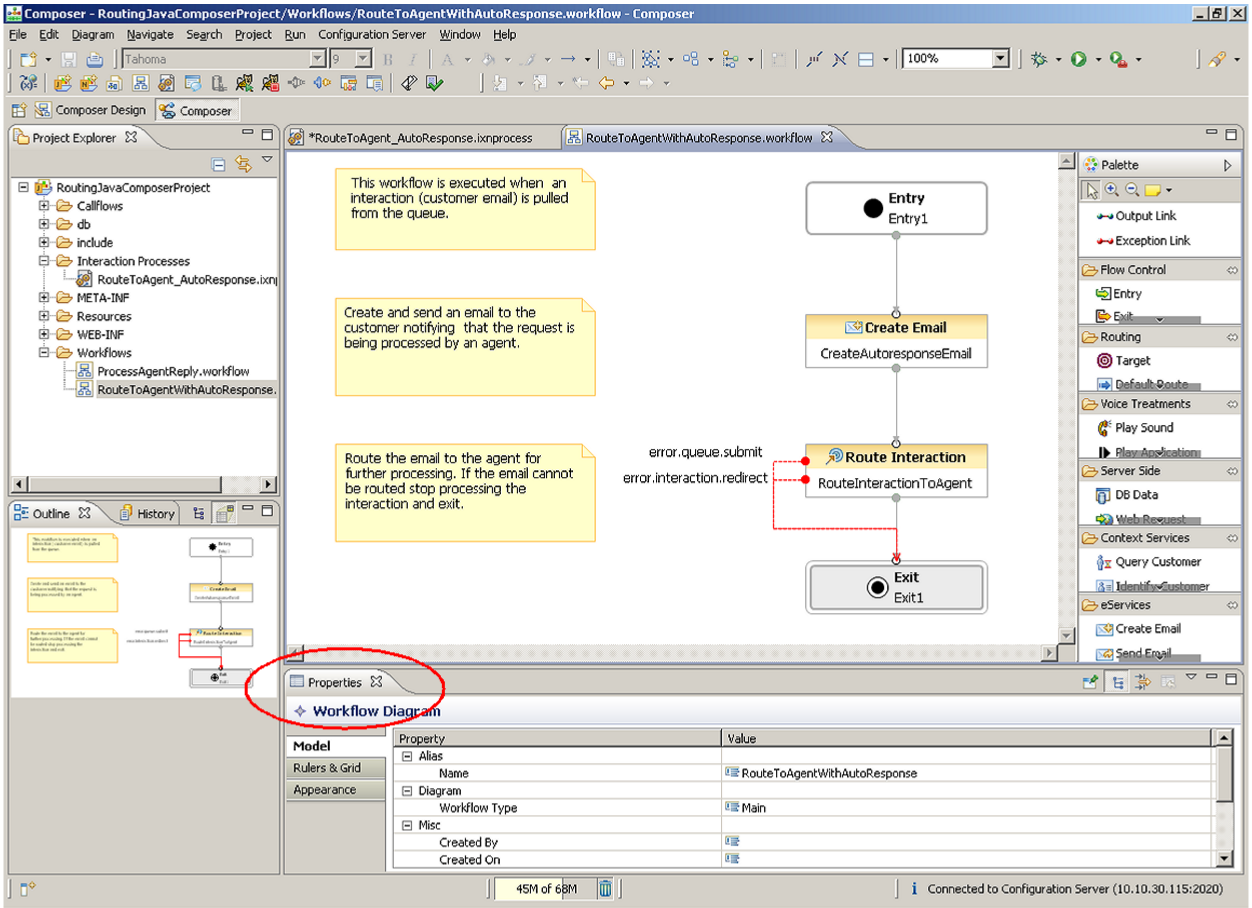
Connected to Configuration Server (10.10.30.115:2020)

Figure 7: Workflow Block Selected in IPD

Resource Property Points to Workflow Diagram

The `Properties` view at the bottom of Figure 7 contains a `Resource` property, which points to a workflow diagram in the `Workflows` folder in the `Project Explorer` on the left. The name of the workflow diagram is `RoutetoAgentWithAutoResponse.workflow`.

To view this workflow diagram on the canvas, double-click its name in the `Project Explorer`. The canvas now appears as shown in Figure 8 on page 27.



This workflow is executed when an interaction (customer email) is pulled from the queue.

Create and send an email to the customer notifying that the request is being processed by an agent.

Route the email to the agent for further processing. If the email cannot be routed stop processing the interaction and exit.

Property	Value
⊟ Alias	
Name	RouteToAgentWithAutoResponse
⊟ Diagram	
Workflow Type	Main
⊟ Misc	
Created By	
Created On	

Connected to Configuration Server (10.10.30.115:2020)

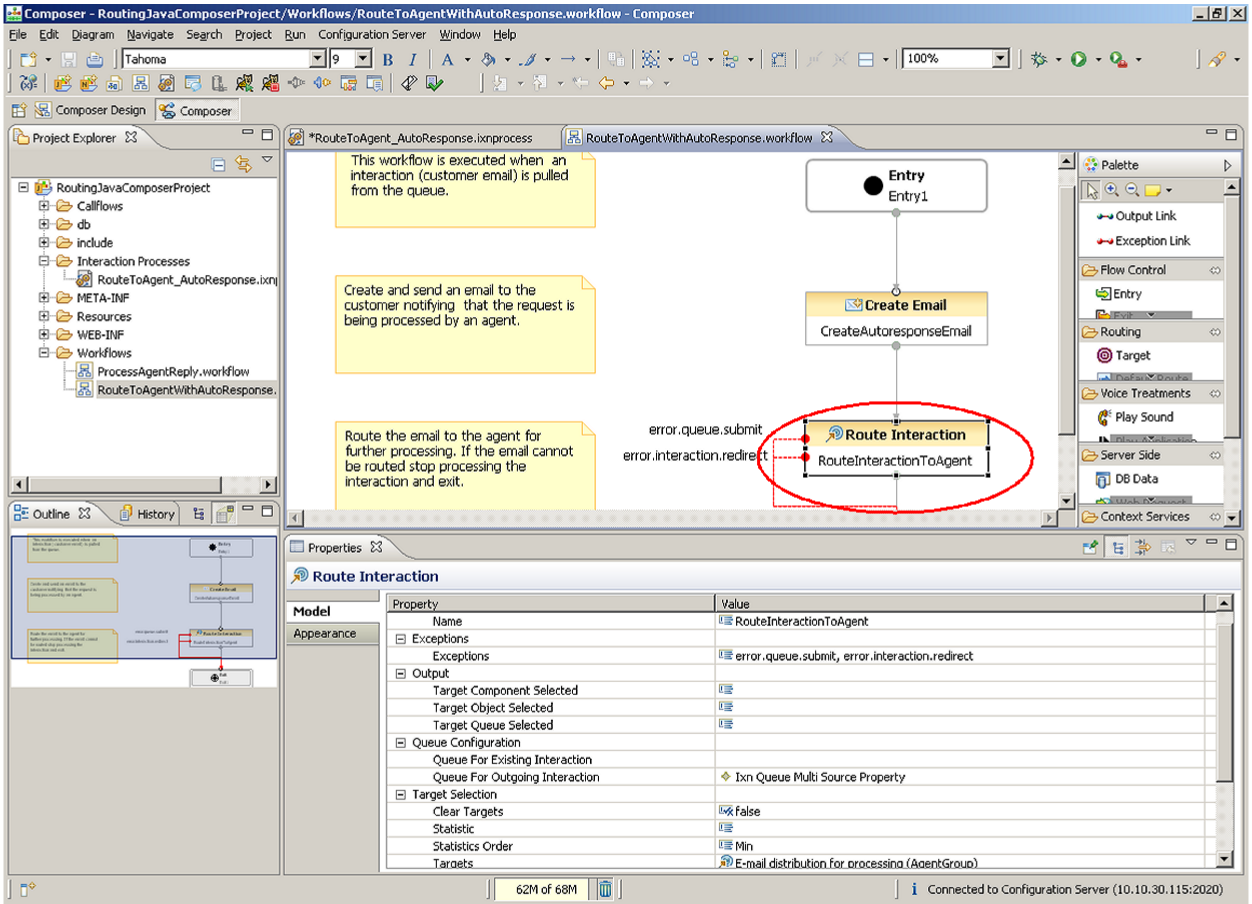
Figure 8: Workflow Diagram Selected in Project Explorer

Note in Figure 8:

- The center canvas area adds a second tab for the workflow diagram: `RoutetoAgentWithAutoresponse.workflow`.
- The palette on the right contains workflow diagram blocks grouped under the categories `Flow Control`, `Routing`, `Voice Treatment`, `Server Side`, `Context Services`, and `eServices`. Appendix A, "Composer Blocks," on page 103 summarizes each block.
- Since no particular block is selected, the `Properties` view shows general properties for the workflow diagram.

Viewing/Defining Block Properties

Assume you double-click the `RouteInteractionToAgent` block in Figure 8. The `Properties` view shows the properties (fields) that can be configured for the selected block (see Figure 9).



This workflow is executed when an interaction (customer email) is pulled from the queue.

Create and send an email to the customer notifying that the request is being processed by an agent.

Route the email to the agent for further processing. If the email cannot be routed stop processing the interaction and exit.

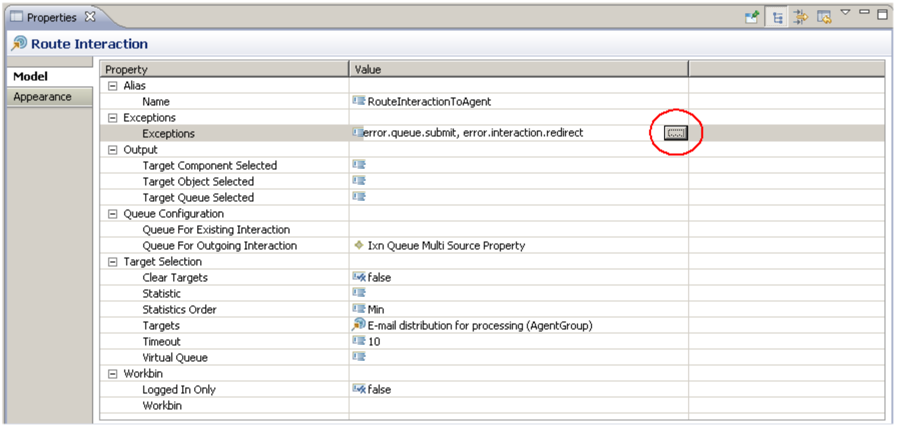
Property	Value
Name	RouteInteractionToAgent
⊟ Exceptions	
Exceptions	error.queue.submit, error.interaction.redirect
⊟ Output	
Target Component Selected	
Target Object Selected	
Target Queue Selected	
⊟ Queue Configuration	
Queue For Existing Interaction	
Queue For Outgoing Interaction	Ixn Queue Multi Source Property
⊟ Target Selection	
Clear Targets	false
Statistic	
Statistics Order	Min
Targets	E-mail distribution for processing (AgentGroup)

Figure 9: Properties View for Selected Workflow Diagram Block

When configuring/editing fields in the `Properties` view:

- Some fields allow you type directly in them.
- Other fields display a ⋯ button when you click under `Value` in the `Properties` view. You then click that button to open a dialog box.

For example, clicking opposite `Exceptions` under `Value` causes this button to appear (see Figure 10).



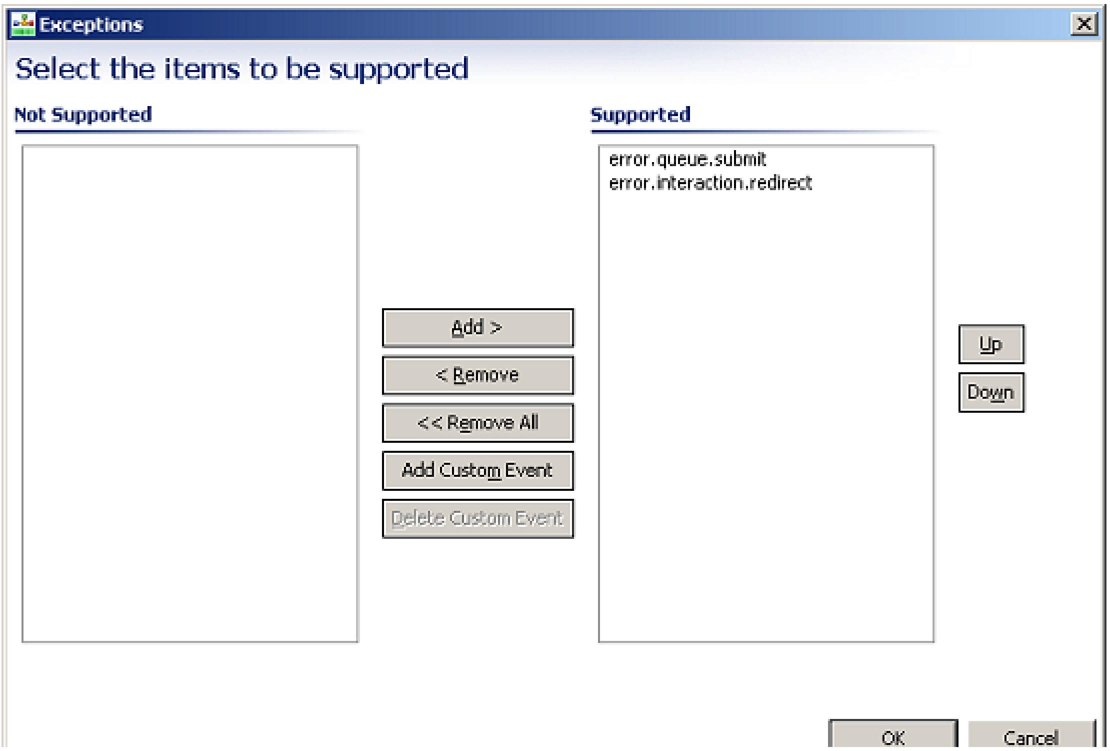
Property	Value
Alias	
Name	RouteInteractionToAgent
Exceptions	
Exceptions	error.queue.submit, error.interaction.redirect
Output	
Target Component Selected	
Target Object Selected	
Target Queue Selected	
Queue Configuration	
Queue For Existing Interaction	
Queue For Outgoing Interaction	Ixn Queue Multi Source Property
Target Selection	
Clear Targets	false
Statistic	
Statistics Order	Min
Targets	E-mail distribution for processing (AgentGroup)
Timeout	10
Virtual Queue	
Workbin	
Logged In Only	false
Workbin	

Figure 10: Button for Accessing Properties Dialog Box

Click this button to bring up a dialog box (see Figure 11) where you configure the handling of events this block may possibly encounter.

Exception/Error Handling

In the case of the `Exceptions` property, clicking the □ button brings up the `Exceptions` dialog box (see Figure 11):



error.queue.submit
error.interaction.redirect

Figure 11: Exceptions Dialog Box

Here you select supported and non-supported exception events for the block. After you click `OK` to close the dialog box, each supported exception event causes an red error port to appear on the block, which can be connected to another block for error handling.

In the example in Figure 12 (using a different workflow), the Identify Customer block defines one exception event so a single red error port appears, which connects to a block for handling the exception event, Query Services.

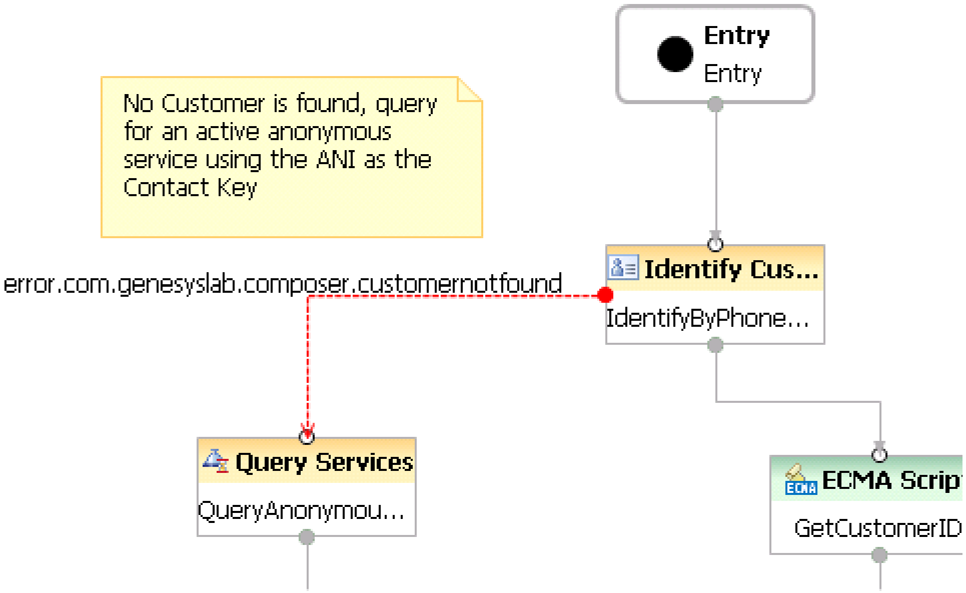

Entry
Entry
No Customer is found, query for an active anonymous service using the ANI as the Contact Key
error.com.genesyslab.composer.customernotfound
Identify Cus...
IdentifyByPhone...
Query Services
QueryAnonymou...
ECMA Scrip
GetCustomerID

Figure 12: Error Port Connected to Block for Error Handling

Exceptions can be configured at two levels:

1. At the individual block-level for local exception handling using the `Exceptions` property.

> **Note:** These block-level exceptions may not be present in all blocks.

2. At the top level in the Entry block using the `Exceptions` property (see Figure 13).

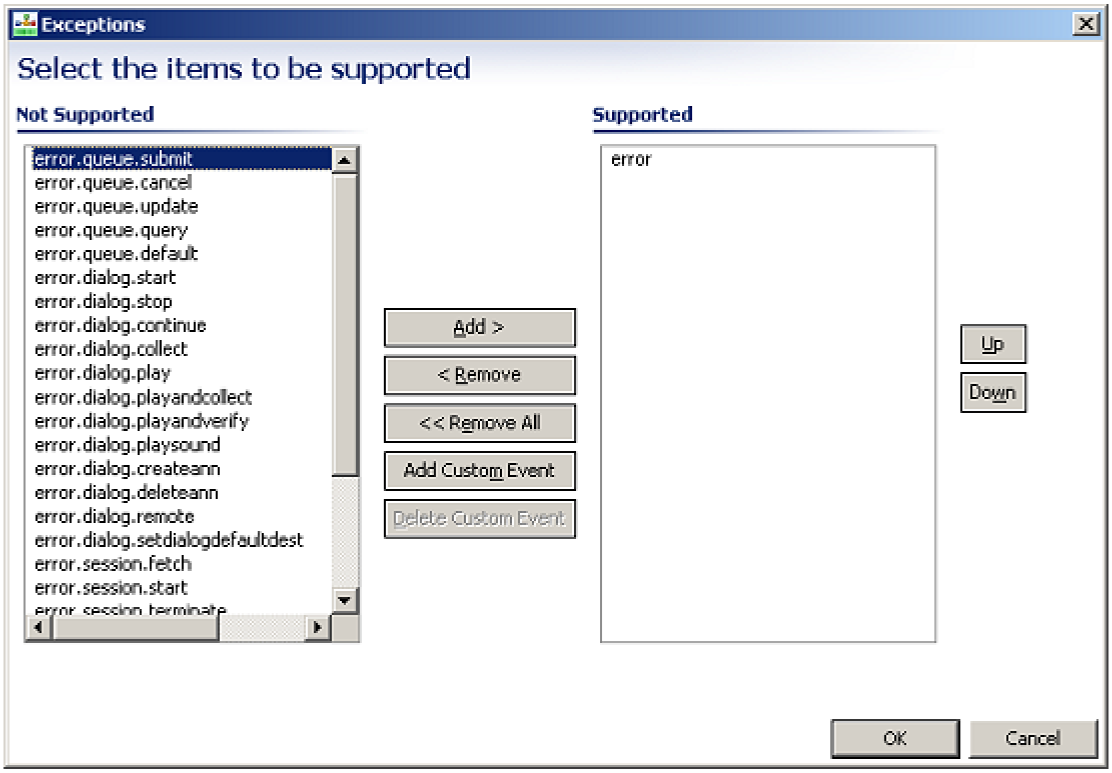


Figure 13: Exceptions Dialog Box for Entry Block

Variables: Project and Application

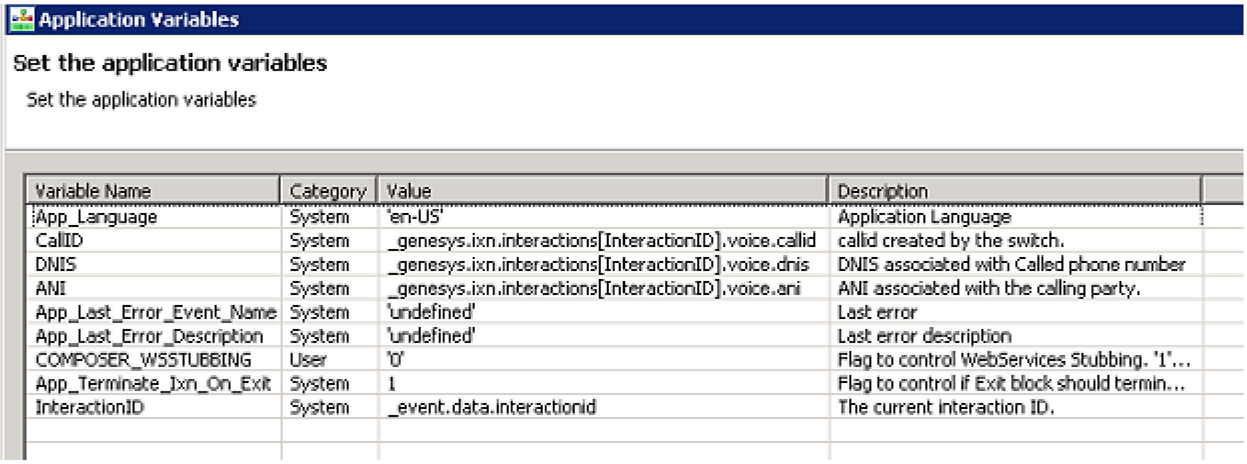
You have the option of defining two types of variables in a routing application:

1. Application-level
2. Project-level

Application Variables

You define Application (workflow) variables in the Entry block.

Clicking the button opposite the `Variables` property in the `Properties` view opens the `Application Variables` dialog box. Figure 14 shows an example completed dialog box.



Set the application variables

Set the application variables

Variable Name	Category	Value	Description
App_Language	System	'en-US'	Application Language
CallID	System	_genesys.ixn.interactions[InteractionID].voice.callid	callid created by the switch.
DNIS	System	_genesys.ixn.interactions[InteractionID].voice.dnis	DNIS associated with Called phone number
ANI	System	_genesys.ixn.interactions[InteractionID].voice.ani	ANI associated with the calling party.
App_Last_Error_Event_Name	System	'undefined'	Last error
App_Last_Error_Description	System	'undefined'	Last error description
COMPOSER_WSSTUBBING	User	'0'	Flag to control WebServices Stubbing. '1'...
App_Terminate_Ixn_On_Exit	System	1	Flag to control if Exit block should termin...
InteractionID	System	_event.data.interactionid	The current interaction ID.

Figure 14: Application Variables Dialog Box

Use application 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.

Composer predefines a number of default routing application variables as shown above in Figure 14 including DNIS and ANI.

Project Variables

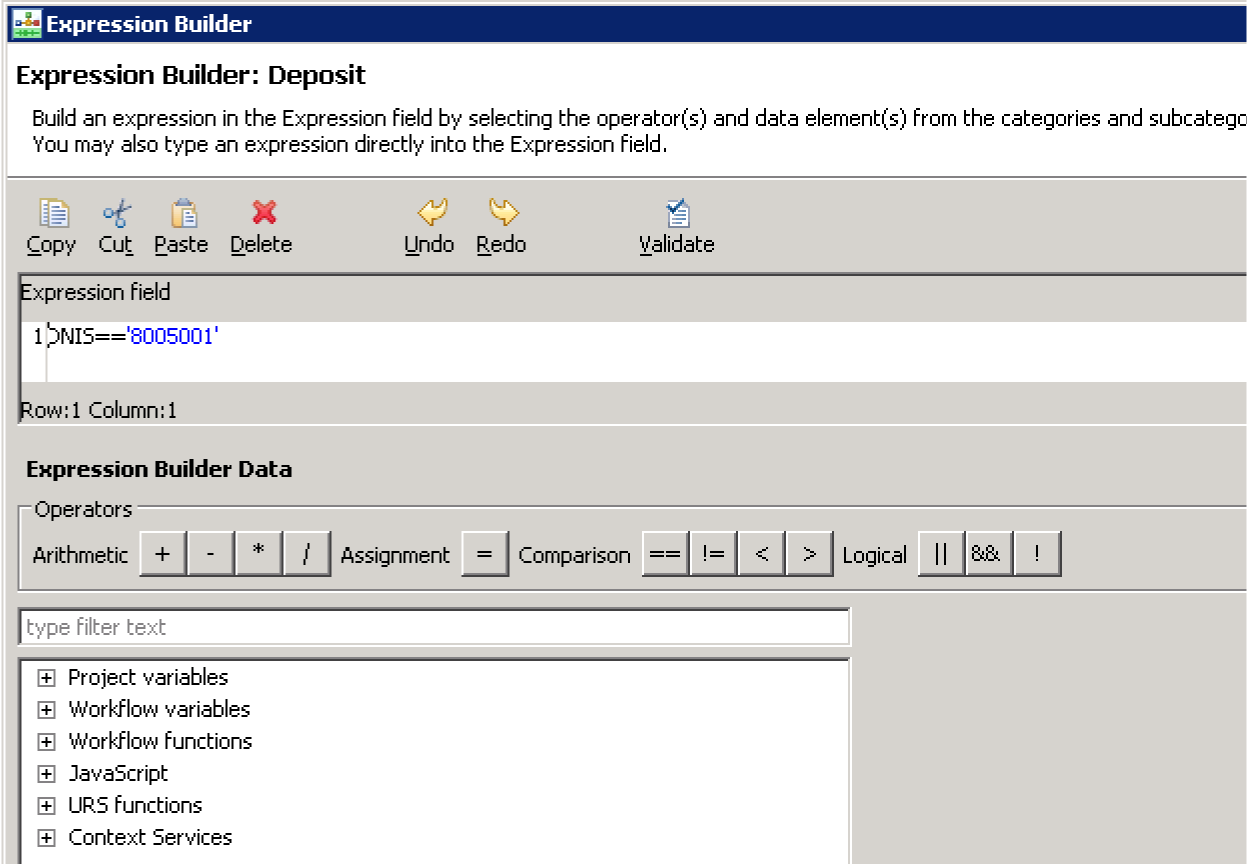
You define the second type of variable in the `Project Variables` dialog box, which opens when you click the access project variables button on the toolbar when an IPD is in focus (see Figure 21 on page 39).

Use Project variables when you need to share information across different workflows.

Expression Builder

Composer's Expression Builder lets you build expressions for branching and conditional routing decisions in a workflow. Figure 15 shows an example simply expression:

```
DNIS=='8005001'
```



Expression Builder: Deposit

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcatego
You may also type an expression directly into the Expression field.

```
1 DNIS=='8005001'
```

Row:1 Column:1

Figure 15: Expression Builder

In this example, the dialed number (DNIS) is stored in a workflow variable called `DNIS` (see Figure 14 on page 32).

In summary, you create expressions by:

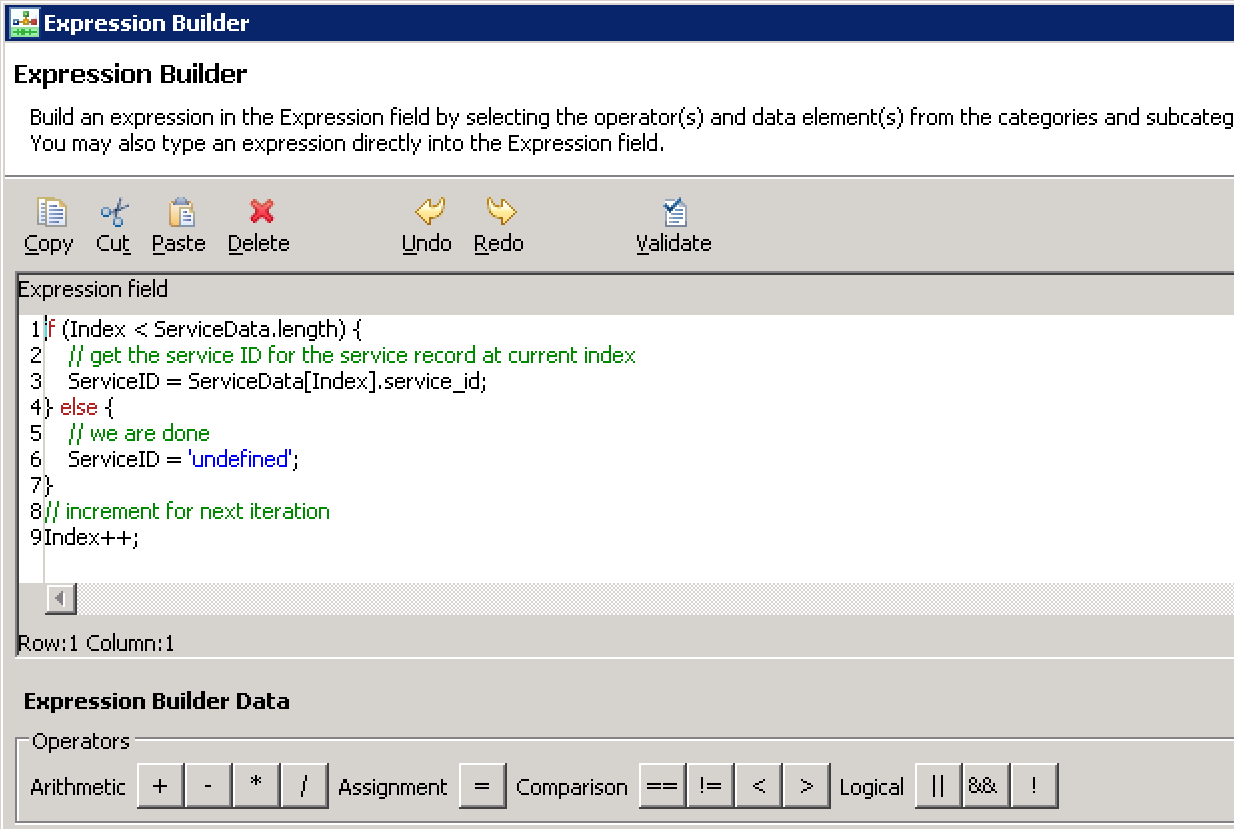
- Expanding the categories shown above and selecting values.
- Using the buttons under `Operators`.
- Typing part of the expression directly when necessary.

Expression Builder opens from the following blocks:

- Assign—`Assign Data` Property
- Branching—`Conditions` Property
- ECMAScript (for workflows)—`Script` Property
- Entry—`Variables` Property
- Log—`Logging Details` Property
- Looping—`Exit Expression` Property

ECMAScript Expressions

Universal Routing Server 8.0 supports SCXML plus ECMAScript as a routing language. While the core SCXML provides State Chart functionality, you can specify URS-specific instructions, such as conditions that can be used for routing decisions, in the form of ECMAScript. The `Script` property in the ECMAScript block brings up Expression Builder for creating those conditions. Figure 16 shows an example ECMAScript expression.



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcateg
You may also type an expression directly into the Expression field.

```
if (Index < ServiceData.length) {
  // get the service ID for the service record at current index
  ServiceID = ServiceData[Index].service_id;
} else {
  // we are done
  ServiceID = 'undefined';
}
// increment for next iteration
Index++;
```

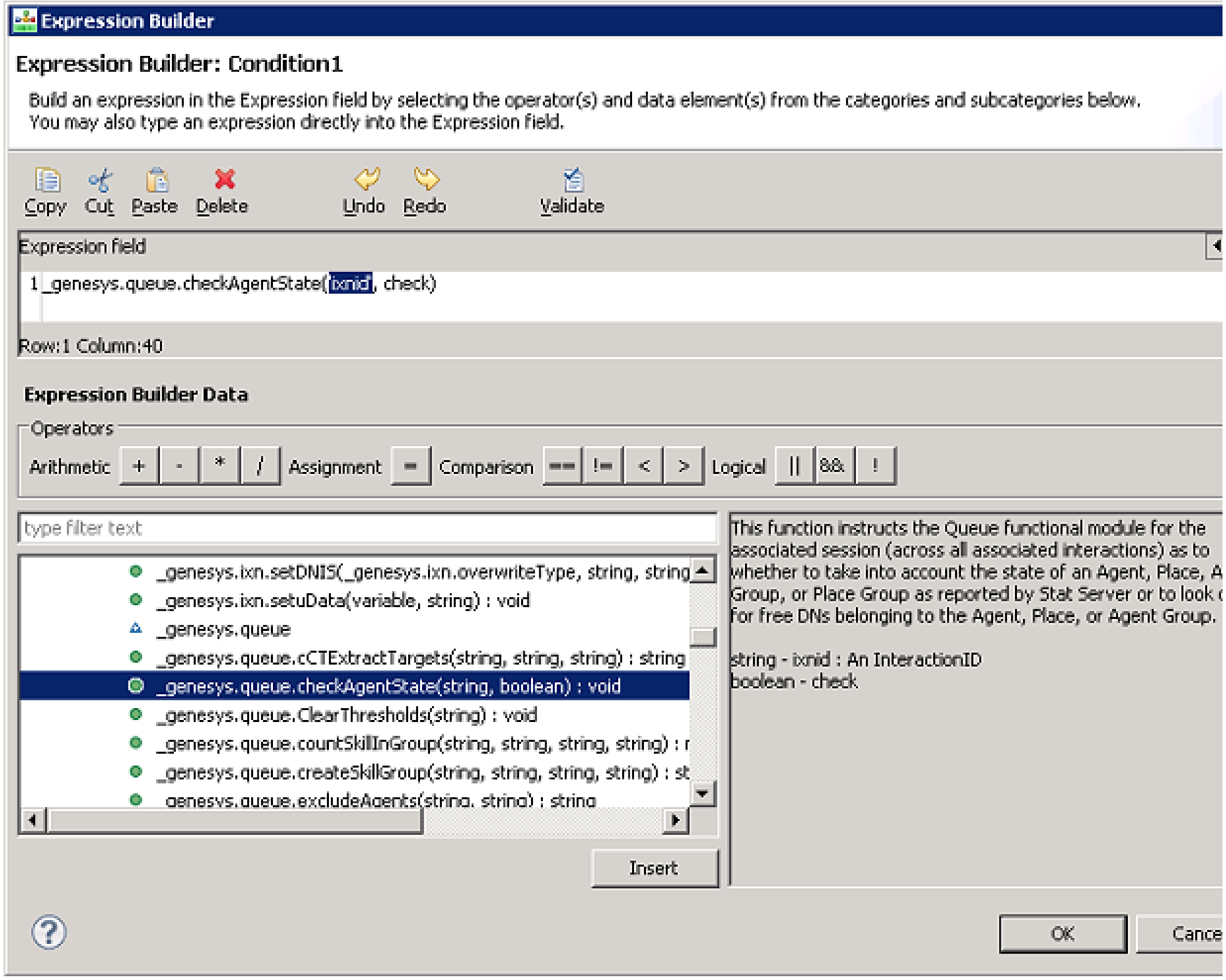
Row:1 Column:1

Figure 16: ECMAScript Expression in Expression Builder

Genesys Functional Modules

You can also build expressions that use the Genesys-supplied Functional Modules as described in Appendix B, “Functional Modules,” on page 113. For example, in Expression Builder:

1. Select the `URS Functions` data category (see Figure 15 on page 33) to display the various categories for the Genesys-supplied Functional Modules.
2. For purposes of this example, select the `_genesys` data subcategory.
3. Double-click a function to insert (see Figure 17).



Expression Builder

Expression Builder: Condition1

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 _genesys.queue.checkAgentState(ixnid, check)
```

Row:1 Column:40

Expression Builder Data

type filter text

```
_genesys.ixn.setDNIS(_genesys.ixn.overwriteType, string, string
_genesys.ixn.setuData(variable, string) : void
_genesys.queue
_genesys.queue.cCTExtractTargets(string, string, string) : string
_genesys.queue.checkAgentState(string, boolean) : void
_genesys.queue.ClearThresholds(string) : void
_genesys.queue.countSkillInGroup(string, string, string, string) : r
_genesys.queue.createSkillGroup(string, string, string, string) : st
genesys.queue.excludeAgents(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, A Group, or Place Group as reported by Stat Server or to look for free DNs belonging to the Agent, Place, or Agent Group.

string - ixnid : An InteractionID
boolean - check

Figure 17: Genesys Functional Modules in Expression Builder

The `_genesys.queue.`modules implement the target selection functionality of URS (finding resources for interactions and delivering interactions to the resource). The selected module, `genesys.queue.checkAgentState('ixnid')` corresponds to the URS CheckAgentState function as described in the Universal *Routing 8.0 Reference Manual*, CheckAgentState function.

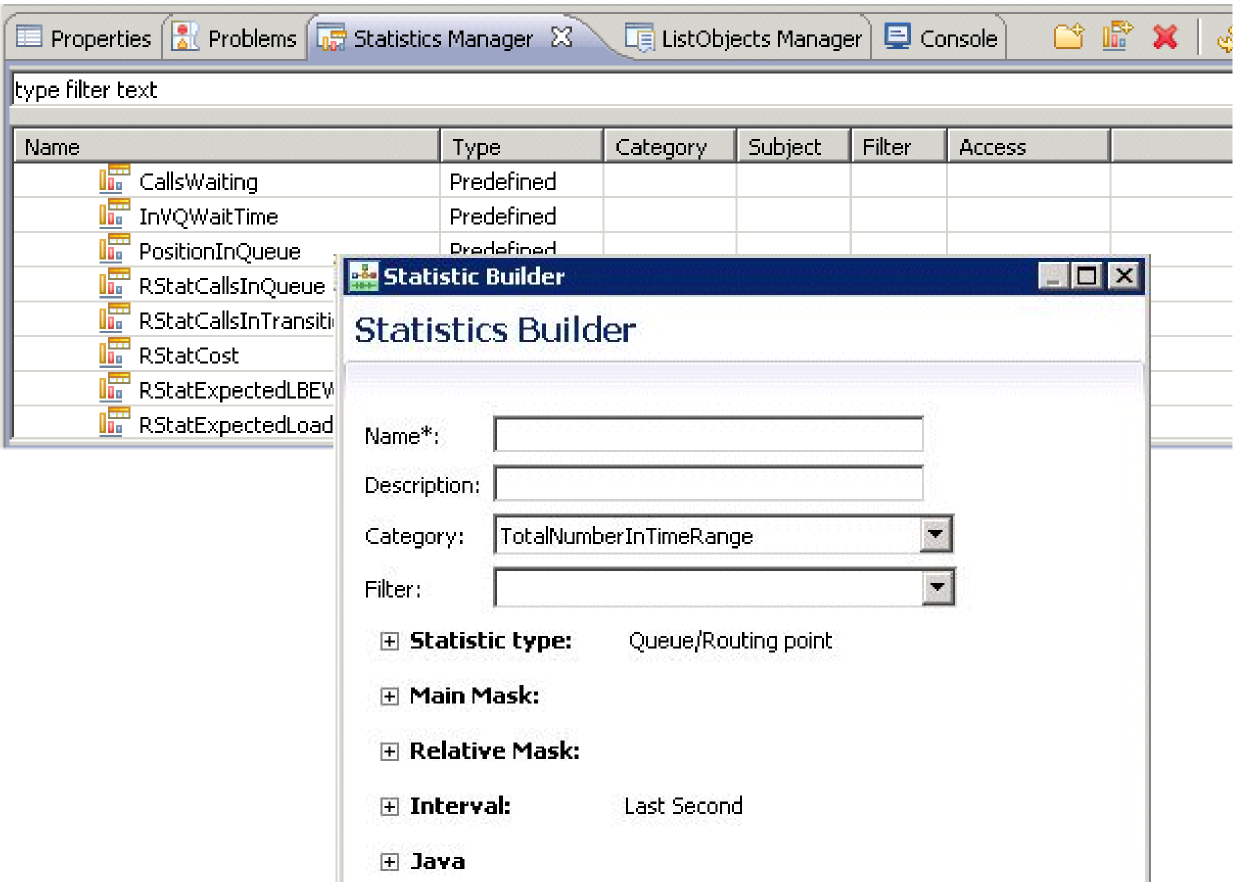
In Figure 17, note that a description of any selected Functional Module appears on the right.

For more information on the Functional Modules, see:

- Appendix B, “Functional Modules,” on page 113.
- The *Genesys 8.0 SCXML Technical Reference*, by selecting `Help` > `Contents` within Composer. Go to section `SCXML Expressions` > `Object Module`.
- Also see: `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`

Statistics Manager/Builder

Composer has a Statistics Manager and Builder, which you can use to instruct Universal Routing Server to use the value of a statistic during target selection, such as `StatTimeInReadyState` (see Figure 18).



Name	Type	Category	Subject	Filter	Access
CallsWaiting	Predefined				
InVQWaitTime	Predefined				
PositionInQueue	Predefined				
RStatCallsInQueue					
RStatCallsInTransiti					
RStatCost					
RStatExpectedLBEW					
RStatExpectedLoad					

Name*:

Description:

Category: TotalNumberInTimeRange

Filter:

⊞ **Statistic type:** Queue/Routing point

⊞ **Main Mask:**

⊞ **Relative Mask:**

⊞ **Interval:** Last Second

⊞ **Java**

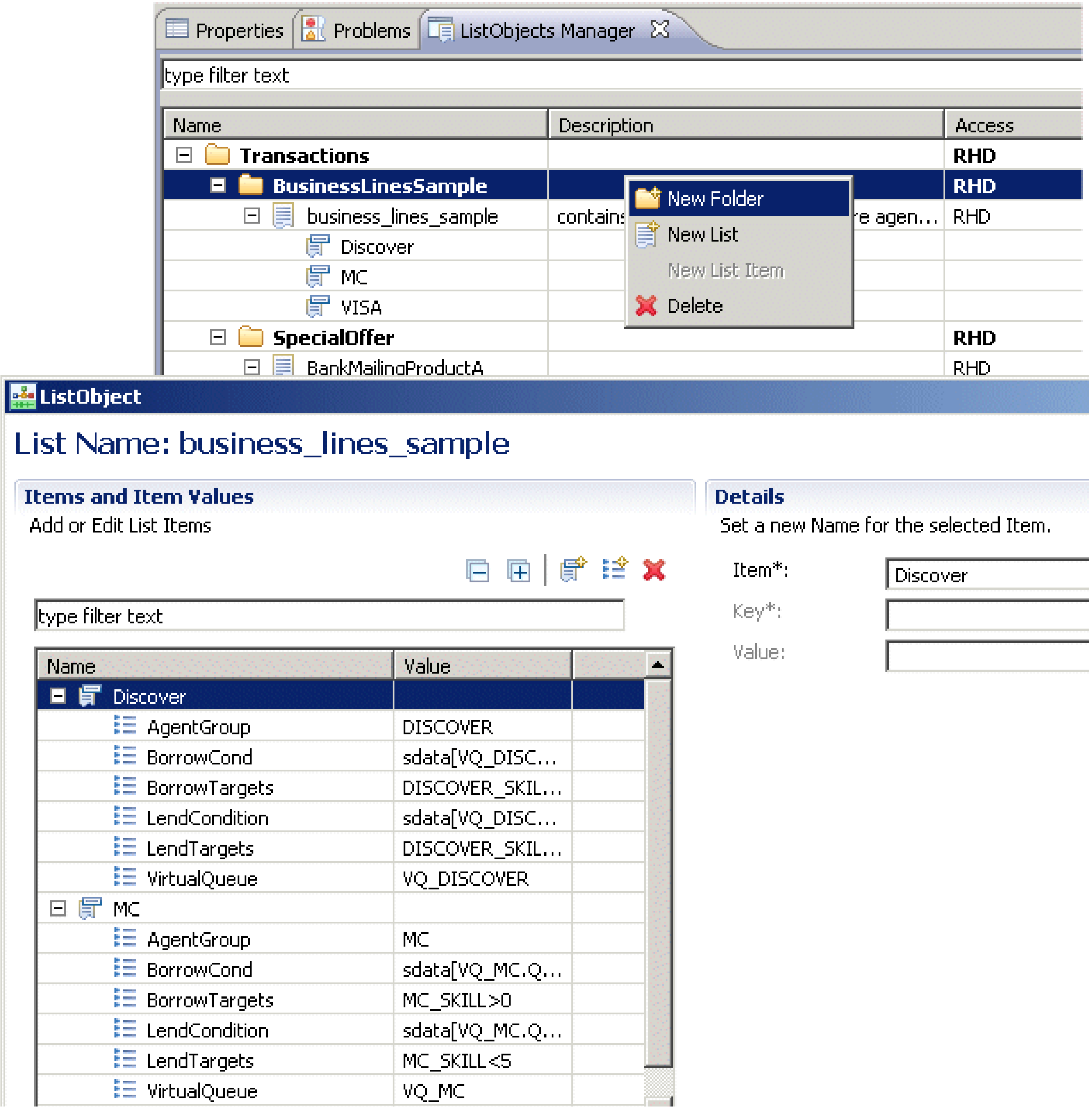
Figure 18: Statistics Manager/Builder

The statistic can be a URS predefined statistic (as described in the *Universal Routing 8.0 Reference Manual*) or a statistic that you create yourself with Statistics Builder. Once you create a statistic, that statistic becomes available for selection via the `Statistics` property in Composer’s Target block.

Opens by clicking the button for Statistics Manager on the main toolbar (see Figure 21 on page 39).

List Objects Manager/Builder

Composer has a List Objects Manager/Builder, which you can use to store/access strings of any nature; for example, DNIS or ANI strings.



Name	Description	Access
Transactions		RHD
BusinessLinesSample		RHD
business_lines_sample	contains e agen...	RHD
Discover		
MC		
VISA		
SpecialOffer		RHD
BankMailingProductA		RHD

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

Details

Set a new Name for the selected Item.

Item*: Discover

Key*:

Value:

Figure 19: List Objects Manager

The strings can be as simple as 800 numbers or as complex as routing conditions. In Expression Builder, two URS Functions (see Figure 17 on page 35) can be used to access List Objects:

- `_genesys.session.listLookupValue` and

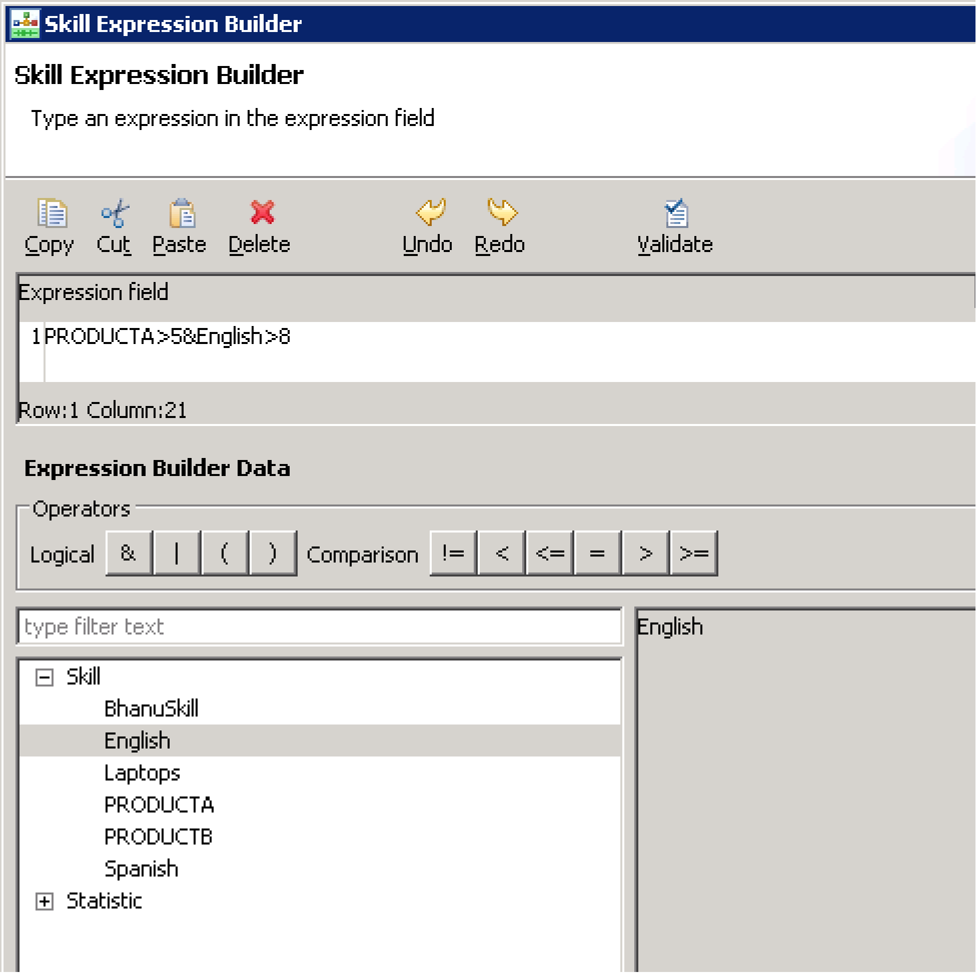
- `_genesys.session.getListItemValue`

List Objects Manager/Builder opens by clicking its button on the main toolbar (see Figure 21 on page 39).

Note: For detailed information on these builders/managers, consult the *Composer 8.0.4 Help*.

Skill Expression Builder

Composer also has a Skill Expression Builder for creating skill expressions used for routing decisions (see Figure 20).



Skill Expression Builder

Type an expression in the expression field

Expression field

```
PRODUCTA>5&English>8
```

Row:1 Column:21

Figure 20: Skill Expression Builder

Skill Expression Builder opens from the `Targets` property in the Target block after selecting the `Skill` as the target type.

Toolbar Buttons for Routing

The main toolbar is displayed at the top of the Composer window directly beneath the menu bar.

Note: Buttons on the main toolbar change based on the active perspective (see “Perspectives” on page 89). 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.

Figure 21 identifies frequently used buttons in `Composer Design` perspective when creating routing applications.

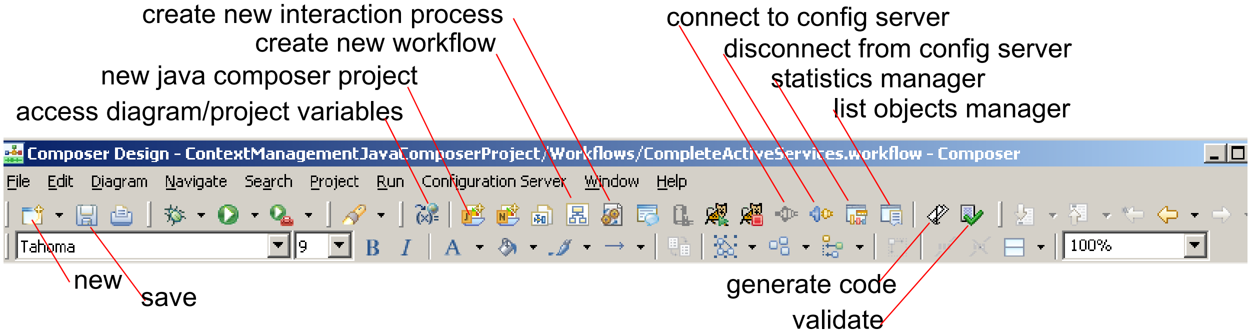

create new interaction process
create new workflow
new java composer project
access diagram/project variables
connect to config server
disconnect from config server
statistics manager
list objects manager
Composer Design - ContextManagementJavaComposerProject/Workflows/CompleteActiveServices.workflow - Composer
File Edit Diagram Navigate Search Project Run Configuration Server Window Help
Tahoma
9
100%
new
save
generate code
validate

Figure 21: Frequently Used Buttons When Creating Routing Applications

Menu Bar

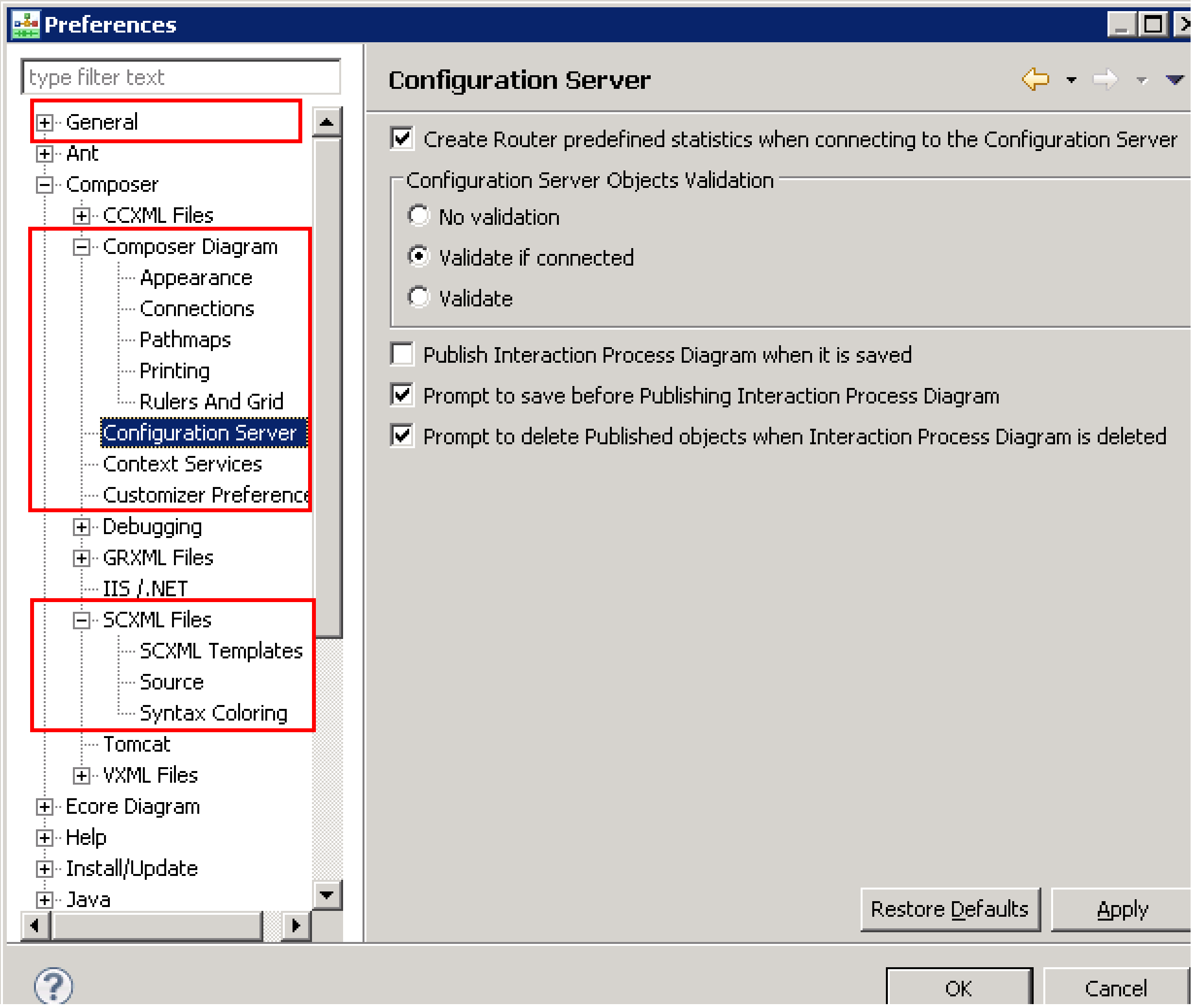
Figure 21 also shows the Composer menu bar. When working with routing applications, some commonly used menu selections are listed below:

- `File` > `New` > `Java Composer Project` or `NET Composer Project` when creating a new routing application. You may also frequently use `File` > `New` > `Workflow Diagram` or `File` > `New` > `SCXML File`. The wizard will prompt you to select the Project. In addition, `Export` and `Import` commands are available.
- The `Diagram` menu contains many commands related to lines, colors, arrows, arranging, aligning, sizing, viewing, filtering, and zooming. There are also additional items in `Composer Diagram` preferences (see Figure 22 on page 40).
- You will want to connect to `Configuration Server` (see page 64) via that menu in order to view objects in the Configuration Database and to update that database when creating new IPDs, diagrams, interaction queues, views, and other objects.

- The Window menu contains many perspective commands as well as giving access to the Preferences dialog box (see Figure 22 on page 40).
- The Help menu access both the Composer and Universal Routing SCXML help systems.

Preferences

Composer Preferences apply to all Projects within the workspace (see page 18). You can set preferences for the following: Configuration Server, SCXML Files, Composer Diagram, Context Services, and Time Zone. To open the Preferences dialog box, select Window > Preferences (see Figure 22).



Configuration Server

☑ Create Router predefined statistics when connecting to the Configuration Server

Configuration Server Objects Validation

○ No validation

● Validate if connected

○ Validate

□ Publish Interaction Process Diagram when it is saved

☑ Prompt to save before Publishing Interaction Process Diagram

☑ Prompt to delete Published objects when Interaction Process Diagram is deleted

Figure 22: Composer Preferences Used for Routing Applications

For details on the routing-related preferences, see the Routing Applications & Workflows book in the *Composer 8.0.4 Help*.

Chapter

3 Summary of Tasks

This chapter summarizes the tasks that are required to create an SCXML-based routing application in Composer. This chapter includes the following sections:

Tasks: Planning & Preparation

After installing Composer as described in the *Composer 8.0 Deployment Guide*, including the post-installation configuration described in that guide, the planning and preparation tasks are provided in Table 1.

Table 1: Planning & Preparation

Task Objective	Related Procedures and Actions
Plan the design.	• See "Planning the Design" on page 47
Install the required software.	This task involves installing the required Genesys software components, such as Universal Routing Server, Orchestration Server, Universal Contact Server, and eServices components, if processing multimedia interactions. • See "Installing the Required Software" on page 50.

Table 1: Planning & Preparation (Continued)

Task Objective	Related Procedures and Actions
Pre-configure database objects.	This task involves creating objects in the Configuration Database either through Genesys Administrator or Configuration Manager. These objects include agents, agent groups, business attributes, standard responses, and so on. • See "Pre-Configuring Database Objects" on page 51. If processing multimedia interactions, this task also includes defining media servers and media server points. • See "Endpoints" on page 68.
Set up Context Services (optional).	Optional. This task is required if managing conversations/personalizing services/offers using the Context Services option of the Universal Contact Server database. • See "Using Context Services" on page 53.
Review the samples.	• See Procedure: Reviewing sample interaction process diagrams, on page 55. • See Procedure: Reviewing sample Projects, on page 55.

Task: Creating a New Project

As described in "Projects" on page 17, a Project organizes all the elements associated with a routing application. Creating a new Project automatically creates a default interaction process diagram (`default.ixnprocess`) and a default workflow diagram (`default.workflow`), which you an edit.

The tasks are provided in Table 2.

Table 2: Creating a New Project

Task Objective	Related Procedures and Actions
Create a new Project.	• Decide whether the Project will be created "from scratch" or based on a predefined Project template (see Figure 25 on page 56). • See Procedure: Creating a new Project, on page 62.

Tasks: Creating the Workflow Diagram(s)

Each Workflow block in an IPD references a workflow diagram (see Figure 7 on page 26). You have the option of creating these diagrams before creating the IPD or having the IPD reference empty placeholder diagrams to be completed later on. If you decide to create the workflow diagrams before the IPD, the required tasks are provided in Table 3.

Table 3: Creating the Workflow Diagram(s)

Task Objective	Related Procedures and Actions
Connect to Configuration Server.	As described on page 64, you have the option of developing routing applications in online or offline mode. To work in online mode and have Configuration Database objects viewable/selectable from Composer blocks: • See Procedure: Connecting to Configuration Server, on page 65.
Connect to the Context Services Server.	This task is used if managing conversations/personalizing services/offers using the Context Services option of the Universal Contact Server database. To work in online mode and have database objects viewable/selectable from Composer blocks: • See Procedure: Connecting to the Context Services Server, on page 66.
Create the workflow diagram(s) that will be referenced in the IPD.	• See Procedure: Creating a workflow diagram, on page 76.

Tasks: Creating the IPD

The tasks that are required to create the routing application's interaction process diagram (see page 14) are provided in Table 4.

Notes: While you can create an IPD on-demand from the toolbar (see Figure 21 on page 39), the Project creation task previously described automatically created an IPD.

The task summary below applies to creating an IPD for processing multimedia interactions. An IPD for voice only interactions contains a single Workflow block.

Table 4: Creating the Interaction Process Diagram

Task Objective	Related Procedures and Actions
Enter IPD properties.	In order to enter IPD properties, the IPD must be in focus. To get the IPD in focus, you can select the `*.ixnprocess` tab or select the IPD in the `Interaction Process` folder in the Project Explorer. • See Procedure: Entering IPD properties, on page 68
Add the Media Server block(s).	This task is required if processing multimedia interactions. • See Procedure: Adding a Media Server Block, on page 70.
Define the interaction queue(s).	An IPD moves interactions from media servers to interaction queues to workflows. The design of your IPD determines how many times this process is repeated. • See Procedure: Defining an Interaction Queue, on page 72.
Define the view to extract interactions from the interactions queue.	You can define the time intervals that Interaction Server uses for checking queues, conditions for extracting interactions, the order for extracting interactions, scheduling, and hints to optimize performance for Oracle databases. • See Procedure: Defining a view for an interaction queue, on page 73.
Add the Workflow block(s).	If the workflow diagram has been previously created, the Workflow block `Resource` property can point to it. • See Procedure: Adding a Workflow block, on page 79.
Connect the IPD blocks.	• See Procedure: Using the Output Link to connect blocks, on page 88.
Publish the IPD.	Publishing an IPD validates Project configuration information and pushes the information out to Configuration Server. • See Procedure: Publishing an interaction process diagram, on page 85.

Tasks: Code Generation and Deployment

The tasks that are required for code generation and deployment are provided in Table 5

Table 5: IPD Code Generation and Deployment

Task Objective	Related Procedures and Actions
Generate the code.	• See Procedure: Generating code for an interaction process diagram, on page 86.
Deploy the application.	Once your application has been unit tested you will need to deploy it to a web application 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. For information on this step: • Consult the book `Validation, Debugging, and Deployment` in the *Composer 8.0 Help*. See the `Deployment` sub-book, Deploying a Routing Application topic. • Also see the section on Application Server requirements in the *Composer 8.0 Deployment Guide*.

Chapter

4 Planning & Preparation

This chapter summarizes recommended planning and preparation processes to undertake prior to working in Composer to create routing applications. It contains the following sections:

Planning the Design

Prior to working in Composer, plan the design of your routing application by considering the various stages of interaction processing that occur (or should occur) at your site. The structure of the IPD will reflect this design.

- If you are routing only voice interactions, the design will focus on the workflow diagram referenced by the IPD. To plan this design, start by studying the sample Projects for routing voice interactions (see Figure 25 on page 56).
- If you are routing multimedia interactions, the design will initially focus on what blocks to include in the IPD as described below followed by what blocks to include in each workflow diagram.

Stages in Multimedia Processing

When planning the design of an IPD for multimedia processing, start by considering the basic stages in the interaction life-cycle at your site. You can then design an IPD that encompasses all stages, or just one stage (see “Moving Interactions Between IPDs” on page 16). The next section 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.

Note: The Genesys Multimedia/eService product contains an `Interaction Workflow Samples` software component. The samples include various business processes (similar to IPDs) and associated queues, views, submitters, workbins, and routing strategies. The design behind the samples could also be applied to routing applications created in Composer. The samples are documented in the appendix of the *Universal Routing 8.0 Business Process User's Guide*. You may wish to study them for design ideas prior to creating routing applications in Composer.

Pre-Routing Stage

The main activities in the pre-routing stage of e-mail handling can potentially include:

- Causing incoming interactions to take different paths in the workflow based on criteria such as time of day, user data contained in the interaction, and so on.
- 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. The samples described in the above note demonstrate how to do this.
- Sending an acknowledgement and/or automatic standard response to the customer who originated the e-mail.
- Determining the agent (if any) who previously handled the interactions that are related to this service.

Route-to-Target Stage

The target may or may not be an agent. 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 check 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.

One or Multiple IPDs?

When designing an IPD for multimedia processing, you have two choices:

1. You can create one complex IPD with multiple workflows that encompass all stages of interaction processing
2. You can create multiple IPDs and link them via interaction queues. For example, you could create one IPD for each stage and link them. See “Moving Interactions Between IPDs” on page 16 for more information.

Summary of Planning/Preparation Process

This section summarizes the entire planning/preparation process prior to actually creating, configuring, and placing blocks in Composer.

1. Determine the interaction processing stages/life cycle at your contact center.
2. List the specific interaction processing functionality required at your contact center within each stage.
3. Determine which Composer blocks will be used in routing workflows to perform the various processing required at each stage.
4. Determine if any ECMAScript expressions will be required to be created in Expression Builder for conditional routing decisions (see the ECMAScript block in “Flow Control Blocks” on page 104).

5. Is there any special interaction processing functionary that is not covered by a Composer block, which will require handwritten SCXML code?
6. Determine which Genesys Functional Modules will be used (if any) in Composer blocks.
7. Determine the media server that will be used. In order to get interactions of a particular media type into an interaction queue in an IPD, the Media Service Application object must have one or more Endpoints defined as described ahead in "Adding a Media Server Block" on page 68.
8. Decide whether you will use one complex IPD or multiple linked IPDs (see "Moving Interactions Between IPDs" on page 16).
9. If you plan on having multiple IPDs linked via workflows, name the queues that will connect the workflows contained within each IPD.
10. 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.

Installing the Required Software

In order to create routing applications, Composer requires the following Genesys products/software components:

- To obtain the full functionality of Composer 8.0.4, the following Genesys products/software components are required: Universal Routing Server ((URS) 8.0.1 and (Orchestration Server (ORS) 8.0.1.

  Note: Universal Routing is a part of the Genesys Customer Interaction Management (CIM) Platform, the collection of core servers that enable the rest of your Genesys environment to process the thousands of interactions representing the needs of your customers. This guide assumes your Genesys CIM Platform is already installed and functioning.

- If you wish to process multimedia interactions, you will need eServices (formerly Multimedia) Interaction Server 8.0. You will also need the servers applicable to the media types being processed, such as Genesys E-mail Server 8.0 (formerly E-mail Server Java).

  For information on the eServices objects that you might wish to use in routing applications, see "eService Objects" on page 51.
- If you wish to use the Context Services capability of Universal Contact Server in routing workflows, you will need Universal Contact Server 8.0.2.

For information on Universal Contact Server database items that you might wish to use in routing applications, see *Universal Contact Server 8.0 Context Services User's Guide*.

Note: For information on other software requirements, consult the *Composer 8.0 Deployment Guide*.

Pre-Configuring Database Objects

Prior to using Composer, it is convenient to have pre-configured certain Configuration Database objects so they will be selectable from Composer menus. The objects described below are in addition to the Skills, Persons, Agent Groups, Places, Place Groups, and other Resource type objects described in the chapter on manually configuring routing in the *Universal Routing 8.0 Deployment Guide*.

eService Objects

At some point in an application that routes multimedia interactions, you may want to use eSevice's Knowledge Management functionality as described in the *eServices/Multimedia 8.0 User's Guide*. If so, it is convenient to pre-configure the following:

- Category codes. Knowledge Manager uses a system of category codes, organized in a tree structure, as a means of organizing pre-written text, called standard responses.
- Standard responses. If using the Create E-mail block (see Table 18 on page 111), you may to use pre-written standard responses, such as for acknowledgement or auto response e-mails. Note: Composer currently does not support the use of field codes in standard responses.

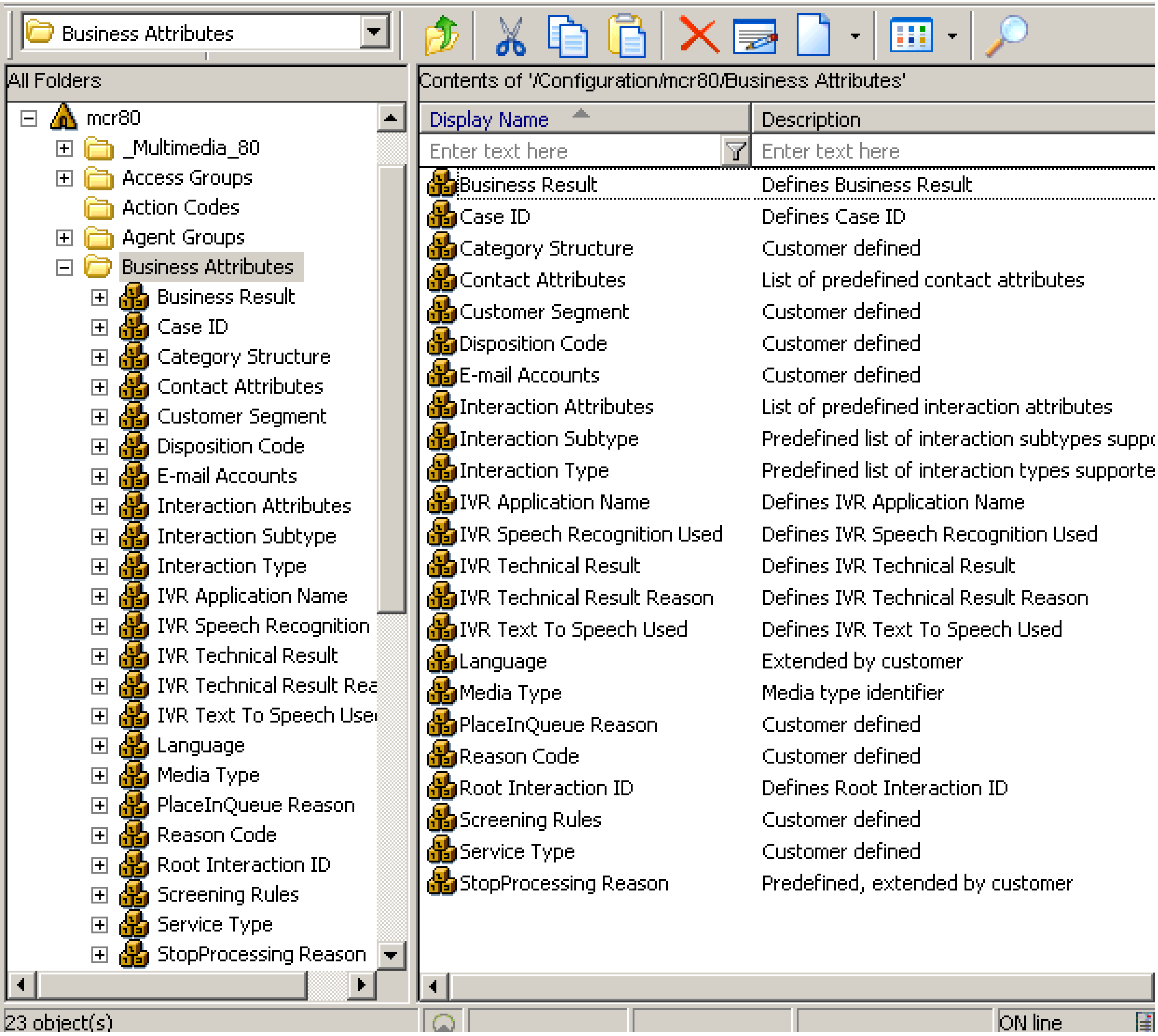
You use eServices's Knowledge Manager to create category trees, and to create and edit the standard responses they can contain.

Note: Any Category structures (and associated standard responses) defined in Knowledge Manager (written to the Universal Contact Server database) are automatically carried over to the Configuration Database. The information is therefore viewable in Configuration Manager or Genesys Administrator.

Business Attributes

At some point in an routing application, you may wish to specify Business Attributes, which are interaction attributes used in different ways within Genesys. The Business Attributes folder (accessible via Configuration

Manager or Genesys Administrator) contains a number of sub-folders containing both customer-defined and Genesys-defined `Business Attributes`. Figure 23 shows the Configuration Manager `Business Attributes` folder.



Contents of '/Configuration/mcr80/Business Attributes'

Display Name	Description
Enter text here	Enter text here
Business Result	Defines Business Result
Case ID	Defines Case ID
Category Structure	Customer defined
Contact Attributes	List of predefined contact attributes
Customer Segment	Customer defined
Disposition Code	Customer defined
E-mail Accounts	Customer defined
Interaction Attributes	List of predefined interaction attributes
Interaction Subtype	Predefined list of interaction subtypes suppo
Interaction Type	Predefined list of interaction types supporte
IVR Application Name	Defines IVR Application Name
IVR Speech Recognition Used	Defines IVR Speech Recognition Used
IVR Technical Result	Defines IVR Technical Result
IVR Technical Result Reason	Defines IVR Technical Result Reason
IVR Text To Speech Used	Defines IVR Text To Speech Used
Language	Extended by customer
Media Type	Media type identifier
PlaceInQueue Reason	Customer defined
Reason Code	Customer defined
Root Interaction ID	Defines Root Interaction ID
Screening Rules	Customer defined
Service Type	Customer defined
StopProcessing Reason	Predefined, extended by customer

Figure 23: Configuration Manager Business Attributes Folder

For example, at some point in a routing application, you may wish to define:

- `E-mail Accounts` if using the Send E-mail or Create E-mail blocks
- `Stop Processing Reason` if using the Stop Interaction block
- `Interaction Subtype` if using the Create SMS block
- `Service Type` if using the Associate Service or Query Services block
- `Disposition Code` if using the Complete Service or Complete State block

If you will be creating expressions in Expression Builder (see Figure 16 on page 34), you may also need to define some new `Business Attributes`.

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, service, and interaction-centric data (current and historical). You can use the Context Services capability for:

- **Service personalization**. You can create routing workflows that alter the customer experience based on information known about the customer.
- **Offer personalization**. Workflows can use the results of previous offers made to the customer to decide whether a new offer should be presented.
- **Service resumption**. Workflows can leverage service state/task information to continue a customer service that was not completed in an earlier interaction.

If the Context Services capability is enabled at your site, you can use the Context Services blocks (see Table 12 on page 108) to create routing applications that extract customer data elements from the UCS Database and apply this knowledge during the routing of interactions.

The `Context Services` category in Expression Builder gives access to customer profile attributes, both core and extension (see Figure 24).

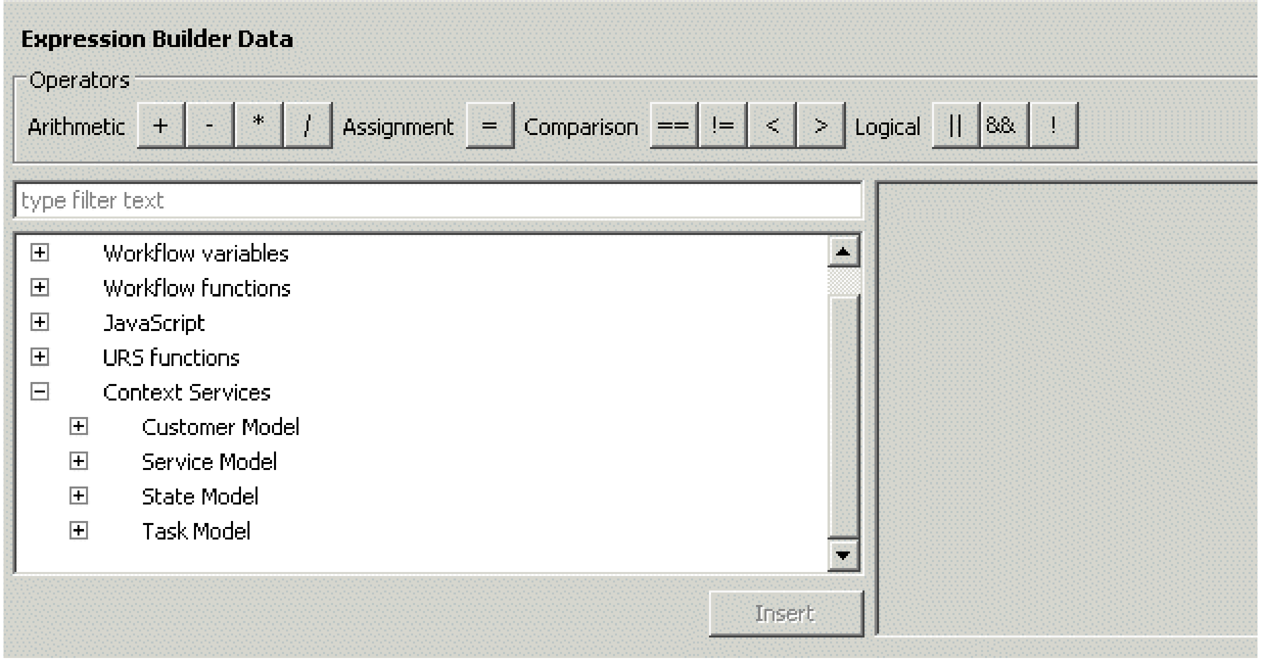


Figure 24: Context Services Data in Expression Builder

Mapping Context Services Attributes

If using the Context Services blocks, you may wish to map Context Services attributes to Configuration Server `Business Attributes` before configuring those blocks in Composer.

- For more information, consult the *Universal Contact Server 8.0 Context Services User's Guide*, Chapter 2, Configuration section. See the description of the `<BusinessAttribute>` option, in the `cview` section.

Managing/Orchestrating Conversations

If you plan to use routing applications to manage/orchestrate conversations with customers, you will need to define certain objects in the Universal Contact Server Database related to services.

Service Model

Genesys Conversation Manager takes the Genesys core capability of routing and extends it. Rather than taking the call (T-Server) or the interaction (eServices/Multimedia) as the basic entity, Conversation Manager takes the service as the basic entity. It provides the ability to "orchestrate" the service process across channels and over time, using dynamic data and business rules to make decisions about operations.

As described in the *Universal Contact Server 8.0 Context Services User's Guide*, a service may be defined as follows:

- It represents a business process, which in turn 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.
- It can span multiple interactions.
- It may include interactions in various media.
- It has a temporal beginning and end.

Universal Contact Server/Context Services makes use of a model in which customers are associated with any number of services. Services are composed of any number of states, and states can in turn be composed of any number of tasks. This three-level structure provides a flexible vocabulary by which organizations store the history of the services that they provide to customers.

Defining Services/States/Tasks

This Service model can also be used by any component that can access UCS/CMS's HTTP interface. You can find more information on this interface in the following documents:

- *Universal Contact Server 8.0 Context Services Developer's Guide*, available on the Genesys SDK Documentation Wiki, covers the writing and the optimization of your applications on top of the Context Services.
- *Universal Contact Server 8.0 Context Services API Reference*, available on the Genesys SDK Documentation Wiki, covers all the representations and methods available throughout the Context Services.

Reviewing the Samples

Within Composer, you have access to various sample Projects, with sample IPDs and workflows.

Procedure: Reviewing sample interaction process diagrams

Start of procedure

To review sample IPDs:

1. Select File > New > Interaction Process Diagram.
2. Select an IPD template and click Next.
3. Select an existing Project.
4. Click Finish. A simple IPD appears in the Interaction Processes folder of the selected Project.

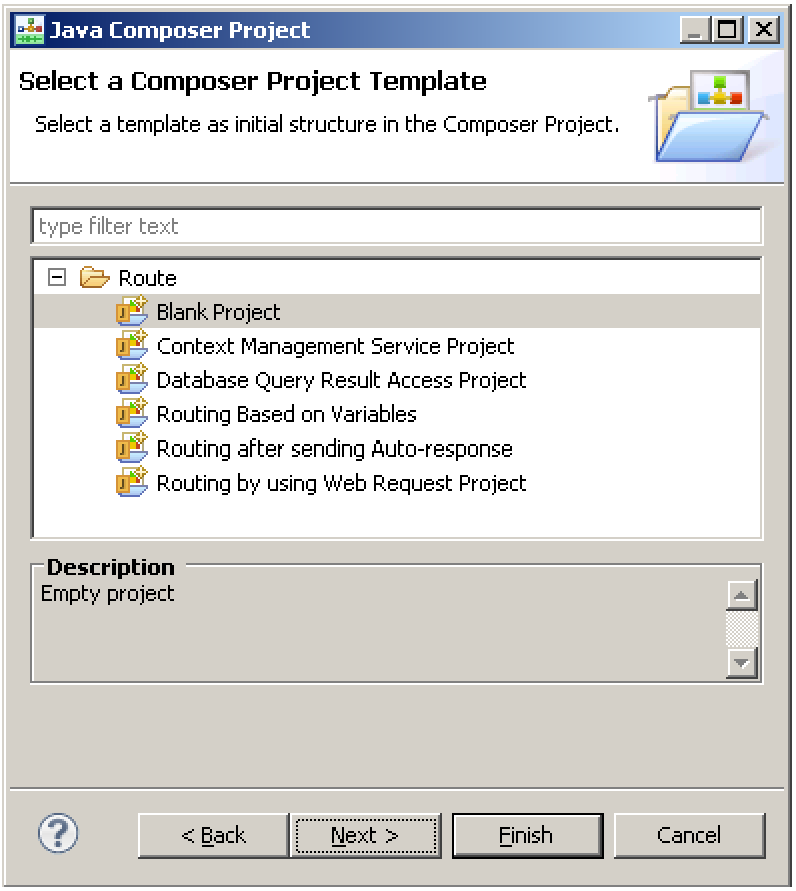
End of procedure

Procedure: Reviewing sample Projects

To review sample Projects:

Start of procedure

1. Select File > New > Java Composer Project.
2. In the resulting dialog box, name the Project, select Route, and click Next. The dialog box shown in Figure 25 opens.



Select a Composer Project Template

Select a template as initial structure in the Composer Project.

- Route
  - Blank Project
  - Context Management Service Project
  - Database Query Result Access Project
  - Routing Based on Variables
  - Routing after sending Auto-response
  - Routing by using Web Request Project

Description: Empty project

Figure 25: Composer-Supplied Project Templates

Each Project shown above contains sample IPDs.

- The `Routing after sending Auto-response` template shows an IPD for routing multimedia interactions.
- The remaining Projects are for voice interactions, which use only Workflow blocks in an IPD.

3. Select a Project template and click `Finish`. The Project appears in the Project Explorer in the upper left of the Composer window.

End of procedure

The next chapter explains how to view the various IPD and Workflow elements.

Enabling/Disabling Functionality

You may hide voice application or routing application development capabilities through a Composer preference setting.

Note: *Voice application* as used above refers to VXML applications that get executed by the Genesys Voice Platform (GVP). Composer Voice applications use voice callflows (as opposed to routing workflows) to generate VXML (as opposed to SCXML).

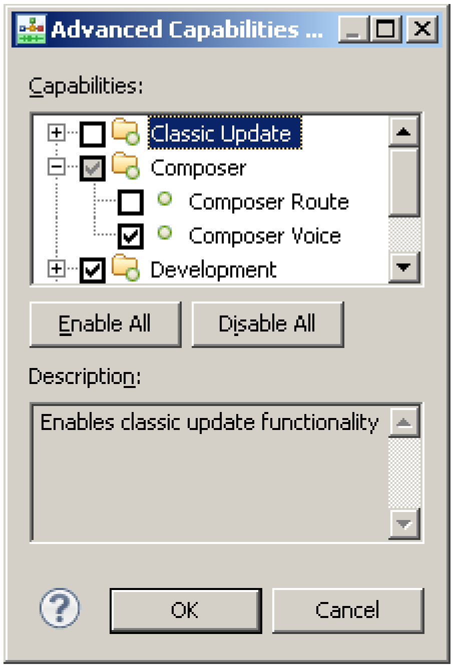
The ability to enable or disable certain functionality is useful for developers who are only developing applications for a specific Genesys platform.

Procedure: Enabling/Disabling Functionality

Start of procedure

To enable or disable routing or voice application development capability:

1. On the main Composer menu, select `Preferences`.
2. Expand `General` (see Figure 22 on page 40) and select `Capabilities`.
3. Click the `Advanced` button.
4. In the `Advanced Capabilities` dialog box, expand `Composer` (see Figure 26).



Capabilities:

- ☐ Classic Update
- ☑ Composer
  - ☐ Composer Route
  - ☑ Composer Voice
- ☑ Development

Description:

Enables classic update functionality

Figure 26: Advanced Capabilities Dialog Box

5. Check/uncheck `Composer Route` or `Composer Voice` as desired.
    - By default, both `Composer Route` and `Composer Voice` are checked. 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 8.0 until you enable Composer Route capability.
6. Click `OK` in both dialog boxes.

End of procedure

Chapter

5 Creating a Routing Application

This chapter leads you through the process of creating a routing application. It contains step-by-step procedures for creating a Project, an interaction process diagram (see page 14), and a routing workflow diagram (see page 15). The procedures re-trace the steps used to create one of the Genesys-supplied Project templates.

This chapter contains the following sections:

To avoid getting lost in detail, the procedures in this chapter are highly summarized. For more detailed information on each procedure, consult the *Composer 8.0.4 Help*.

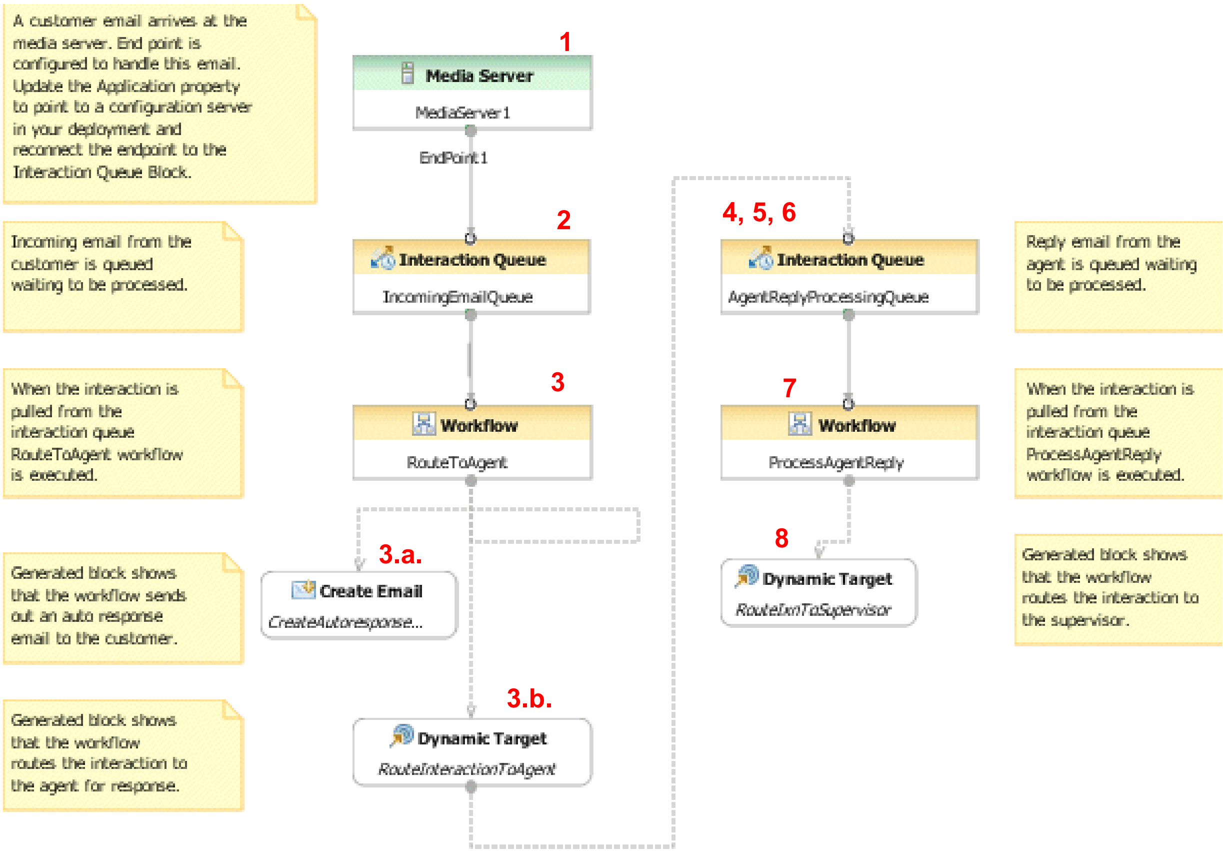
Routing After Sending Auto-response Project Template

Composer provides a set of predefined Project templates containing sample applications (see Figure 25 on page 56). You can start off with the `Blank` Project template and create a routing application from scratch. Or you can use one of the pre-configured templates as a starting point. This chapter starts out with the `Blank` Project template and re-traces the steps used to create the `Routing after sending Auto-response` Project shown in Figure 25 on page 56.

This template demonstrates how to create a routing application that:

- Takes an incoming e-mail from a customer and sends an automatic e-mail acknowledgement.
- Routes the customer's e-mail to an agent for reply.
- Has the agent's reply reviewed by a supervisor.
- Sends out the e-mail reply to the customer.

Figure 27 shows the `RouteToAgent_Autoresponse` IPD associated with the `Routing after sending Auto-response` Project, which we will be re-creating.



A customer email arrives at the media server. End point is configured to handle this email. Update the Application property to point to a configuration server in your deployment and reconnect the endpoint to the Interaction Queue Block.

Incoming email from the customer is queued waiting to be processed.

When the interaction is pulled from the interaction queue RouteToAgent workflow is executed.

Generated block shows that the workflow sends out an auto response email to the customer.

Generated block shows that the workflow routes the interaction to the agent for response.

Reply email from the agent is queued waiting to be processed.

When the interaction is pulled from the interaction queue ProcessAgentReply workflow is executed.

Generated block shows that the workflow routes the interaction to the supervisor.

Figure 27: RouteToAgent_Autoresponse IPD

High-Level Interaction Flow

While the Properties view for any Composer block provides processing detail, an IPD for multimedia processing provides a high-level view of interaction flow. As an example, the numbers below are keyed to the IPD blocks in Figure 27 on page 60.

1. A Media Server IPD block (Media Server1) gets incoming interactions (e-mails, in this case) into the IPD for processing. EndPoint1 is associated with a queue for incoming interactions in the Configuration Database.
2. An Interaction Queue block (IncomingEmailQueue) defines the queue for interactions coming from the media server endpoint.
3. A Workflow IPD block (RouteToAgent) directs Orchestration Server to pull interactions from the queue and process them based on instructions contained in a workflow diagram.
    a. The workflow diagram pointed to by the RouteToAgent Workflow block uses a Create E-mail diagram-building block to send an auto response e-mail to the customer notifying that the system has received their e-mail and someone will be contacting them shortly. The CreateAutoResponse block is a "workflow-generated block" as discussed on page 80.
    b. The same workflow diagram uses a Route Interaction diagram-building block to route the original e-mail from the customer to a target (in this case, an agent group) for a response.

Note: The next two steps (4 and 5) occur outside of the IPD, but could be documented in the IPD with Notes (although this has not been done in the Project template shown in Figure 27 on page 60).

4. After the agent responds (using Reply feature in Genesys Agent Desktop), the response is treated as a new interaction and is placed into the suggested queue.
5. The agent then closes the current interaction (using Done feature in Genesys Agent Desktop).
6. Another Interaction Queue IPD block (AgentReplyProcessingQueue) instructs Orchestration Server to pull the new interaction from the queue and send it to a workflow for processing.
7. A second Workflow IPD block (ProcessAgentReply) points to a workflow diagram that routes the agent response e-mail to a supervisor for review.
8. Once the supervisor is satisfied with the response, the e-mail is placed in the system queue and sent out to the customer.

Creating a New Project

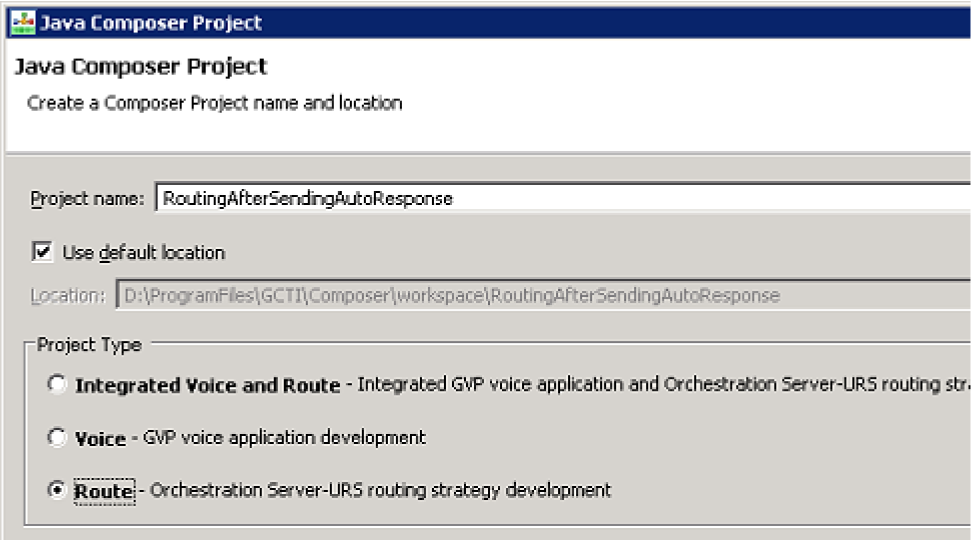
As described on page 17, a Composer *Project* contains everything related to a single routing application. Creating a new Project automatically creates an interaction process diagram with the name of `default.ixnprocess` (which can be changed).

Procedure:
Creating a new Project

Start of procedure

To create a new Project for a routing application:

1. Select `File` > `New` > `Java Composer Project`. This brings up the `Java Composer Project` dialog box.
2. In this dialog box, type a name for your Project. For this example, type `RoutingAfterSendingAutoResponse`.
3. Select the `Use default location` check box.
4. Select the `Route` Project type, which creates a Project associated with the URS 8.0 SCXML Engine/Interpreter. The `Java Composer Project` dialog box now appears as shown in Figure 28.



Java Composer Project

Java Composer Project

Create a Composer Project name and location

Project name: RoutingAfterSendingAutoResponse

☑ Use default location

Location: D:\ProgramFiles\GCTI\Composer\workspace\RoutingAfterSendingAutoResponse

Project Type

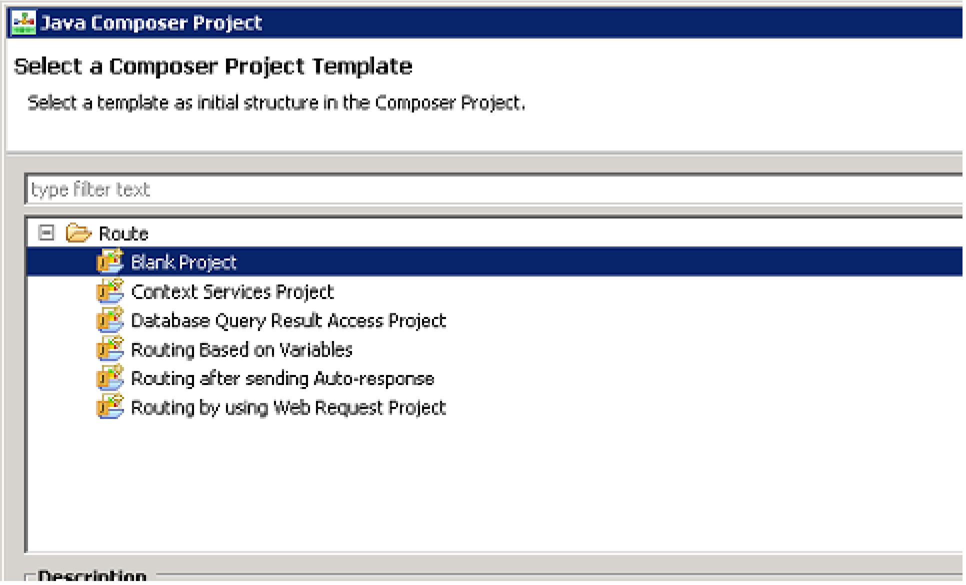
○ **Integrated Voice and Route** - Integrated GVP voice application and Orchestration Server-URS routing str

○ **Voice** - GVP voice application development

● **Route** - Orchestration Server-URS routing strategy development

Figure 28: Java Composer Project Dialog Box

5. Click `Next` (button not shown above). In the dialog box for selecting a Project template, select `Blank Project` (see Figure 29).



Java Composer Project

Select a Composer Project Template

Select a template as initial structure in the Composer Project.

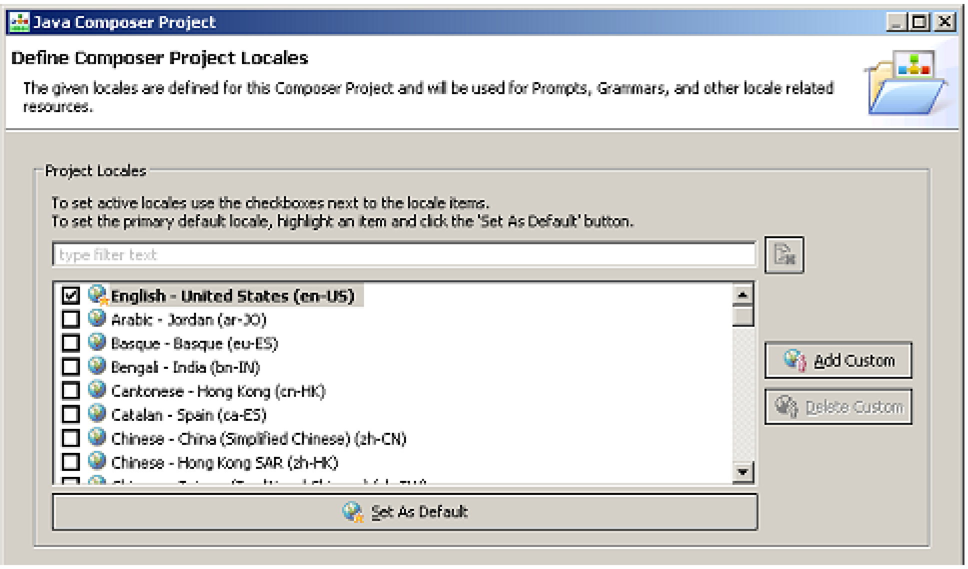
type filter text

- Route
  - Blank Project
  - Context Services Project
  - Database Query Result Access Project
  - Routing Based on Variables
  - Routing after sending Auto-response
  - Routing by using Web Request Project

Description

Figure 29: Select a Composer Project Template Dialog Box

6. Click `Next` (button not shown above).
7. Select the *locale*. As described in the *Composer 8.0.4 Help*, a locale defines a language and region identifier that you want to work with.



Java Composer Project

Define Composer Project Locales

The given locales are defined for this Composer Project and will be used for Prompts, Grammars, and other locale related resources.

Project Locales

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.

type filter text

☑ 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)

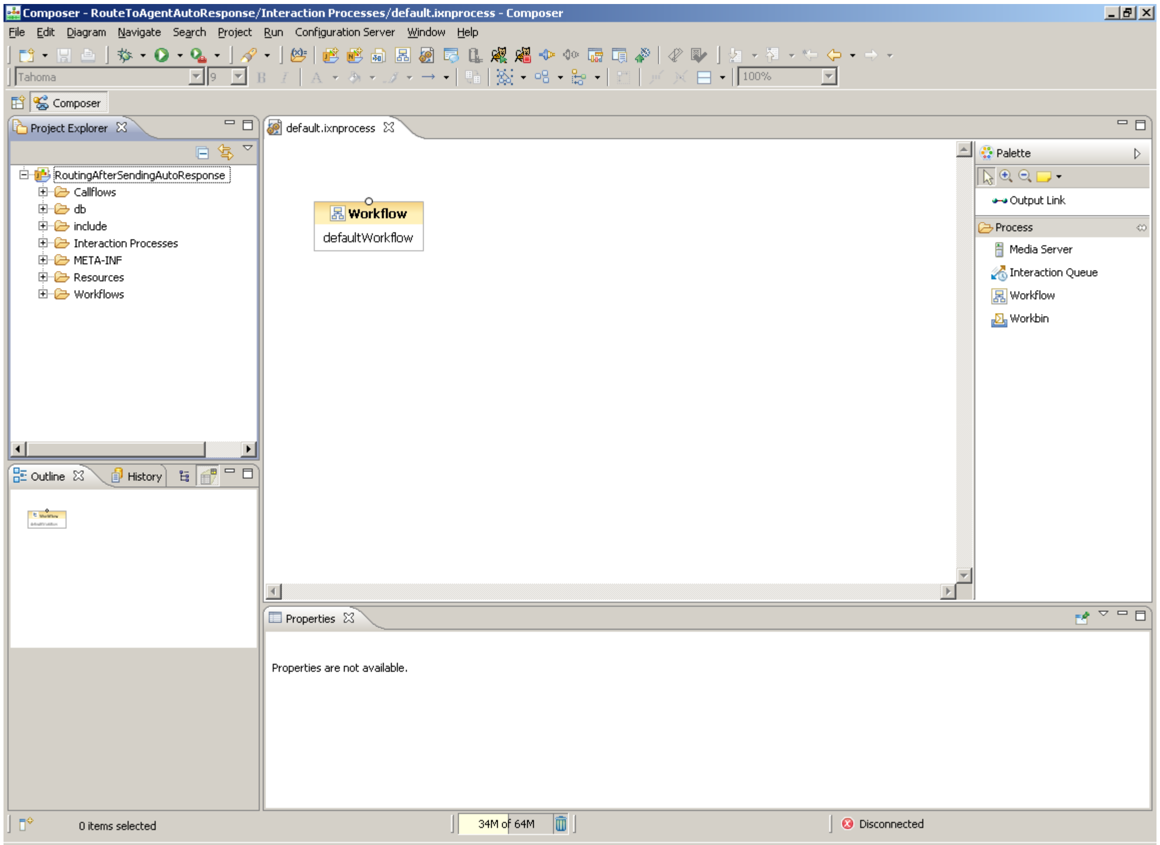
Figure 30: Define Composer Project Locales Dialog Box

8. Click `Finish`. (button not shown).

End of procedure

Composer now starts your new Project based on the `Blank` template, which includes creating a default interaction process diagram (`default.ixnprocess`).

The Project Explorer on the left contains all the files and folders associated with the application so far (see Figure 31).



Properties are not available.

0 items selected

Disconnected

Figure 31: IPD in Composer Perspective, default.ixnprocess Tab

Connecting to Configuration Server

You may develop routing applications:

- With a connection to Configuration Server in “online” mode or
- In an “offline” mode, without connecting to Configuration Server

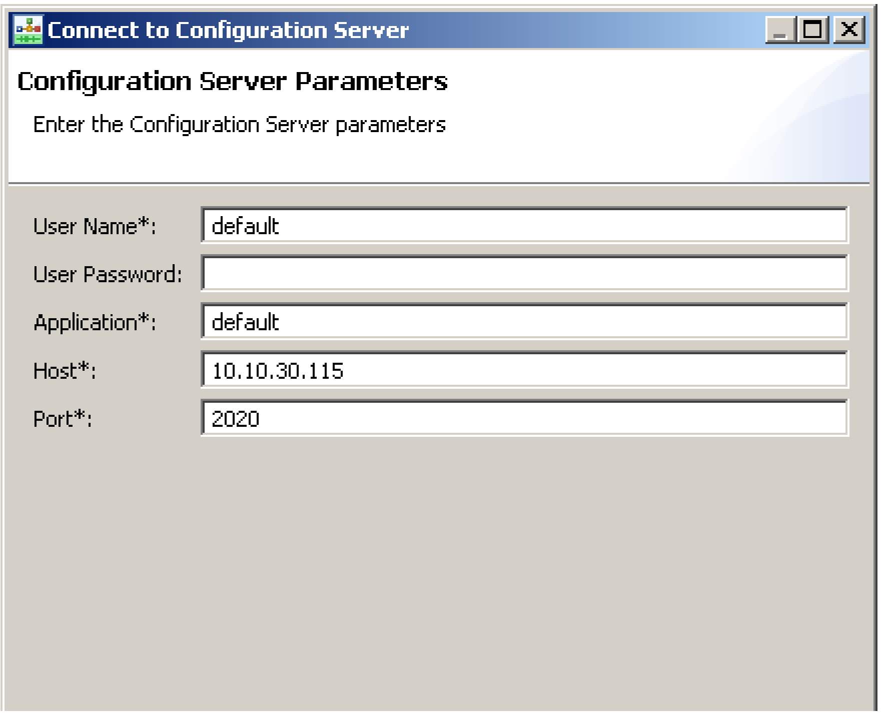
Whether to connect depends on what you wish to do.

- For example, you would need to connect to Configuration Server in order to view and be able to select Configuration Database objects.
- Otherwise, when working in an offline mode, you can manually type the names of Configuration Database objects. Once you connect to Configuration Server, Composer can then validate that these objects actually exist in your Configuration Database, and warn if there are mismatches.

Procedure:
Connecting to Configuration Server

Start of procedure

1. From the Composer main menu, select Configuration Server > Connect. The dialog box in Figure 32 opens.
2. Enter the user name, password, application name, host, and port information for the Configuration Server used in your environment.



Connect to Configuration Server

Configuration Server Parameters

Enter the Configuration Server parameters

User Name*: default

User Password:

Application*: default

Host*: 10.10.30.115

Port*: 2020

Figure 32: Connect to Configuration Server Dialog Box

3. Click Next (button not shown).
4. Select the tenant. For a single-tenant environment, select Environment.
5. Click Finish. Composer can now access Configuration Server data during validation (if configured to do so) and other operations.

End of procedure

Defining Contact Services Preferences

The `Routing after sending Auto-response` Project shown in Figure 25 on page 56 does not use Context Services (see page 53). Information on setting Context Services is included here because many readers will be using those services in routing applications. In this case, you will want to set Contact Services preferences because:

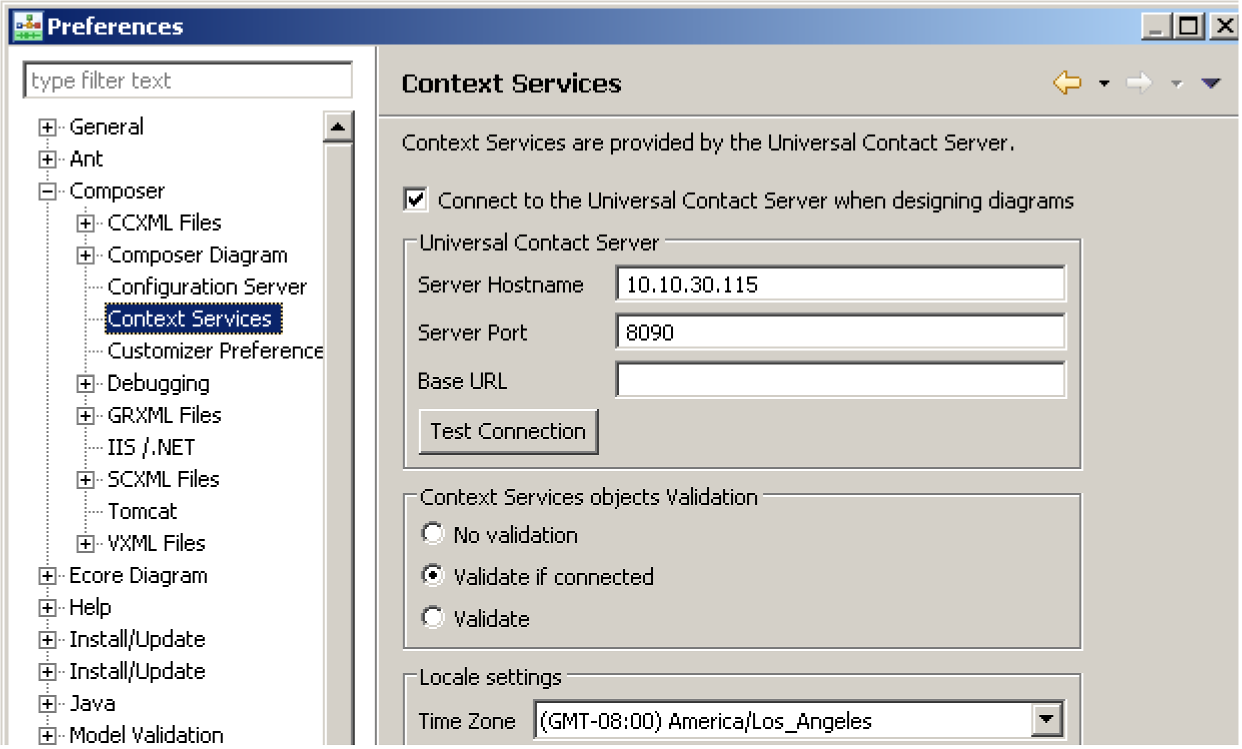
- Customer attributes defined in the Universal Contact Server Database will then be viewable in Expression Builder (see Figure 24 on page 53).
- Many Context Services blocks (see Table 12 on page 108) will display customer profile core and extension attributes for selection. While you can work in offline mode, if connected to the Context Services server, you can select these attributes instead of manually entering them in dialog boxes.

Procedure:
Connecting to the Context Services Server

Start of procedure

If the Context Services capability is enabled, set preferences as follows:

1. Go to `Window` > `Preferences` > `Composer` > `Context Services` (see Figure 33).



Context Services

Context Services are provided by the Universal Contact Server.

☑ Connect to the Universal Contact Server when designing diagrams

Universal Contact Server

Server Hostname	10.10.30.115
Server Port	8090
Base URL	

Test Connection

Context Services objects Validation

○ No validation

● Validate if connected

○ Validate

Locale settings

Time Zone (GMT-08:00) America/Los_Angeles

Figure 33: Context Services Preferences

2. Check the following box to specify online or offline mode: `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.
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 the `Options` tab, select the `cview` section, and the `port` option.

5. Enter the base URL for the Context Services server (UCS).
6. 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.

Note: If the Context Services Service (CMS) is in Maintenance mode, as described in the *Universal Contact Server 8.0 Context Services User's Guide*, testing the connection will show a failure status message: Connection failure (maintenance mode). This indicates that Composer is able to talk to CMS but CMS must be switched to Production mode before Composer blocks can successfully work with it.

7. Under `Context Services object Validation`, select one of the following:
   - `No validation`
   - `Validate if connected`
   - `Validate`
8. Click `OK`.

End of procedure

Entering IPD Properties

When an IPD is in focus, the `Properties` contains fields associated with the IPD (see Figure 6 on page 24). Use the procedure below to define general information about the IPD.

Procedure: Entering IPD properties

Start of procedure

1. If not selected, click the `default.ixnprocess` tab to bring the IPD properties into view. Or expand the `Interaction Processes` folder in the `Project Explorer` and double-click `default.ixnprocess`.
2. Enter the fields as follows:
   a. `Created By`—Enter your name as the author of the Project.
   b. `Created On`—Auto-populated by Composer to indicate the timestamp when the diagram was created.
   c. `Designed Using`—Auto-populated by Composer to indicate version of Composer used to create this diagram.
   d. `Last Modified By`—Provided by you to indicate who updated the diagram last.
   e. `Last Modified On`—Filled in by Composer when the diagram is modified.
   f. `Version`—Provided by you for versioning purposes during development.
3. Select `File` > `Save` from the menu to save the IPD as it exists so far.

End of procedure

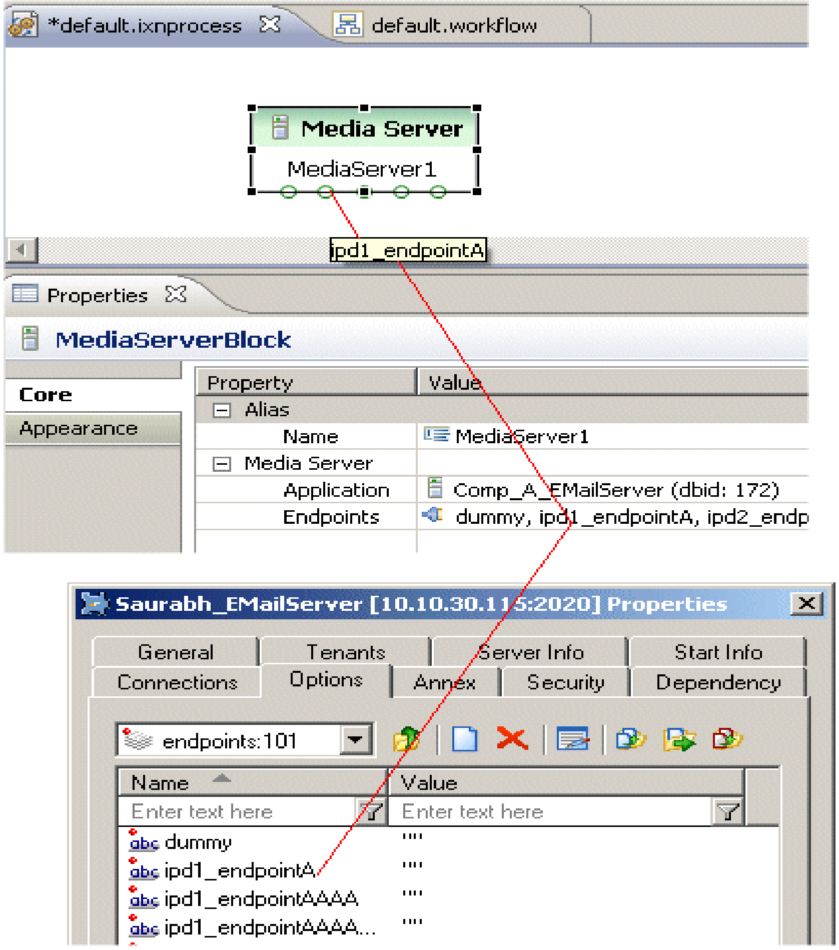
Note: Composer can be used with Eclipse Plug-ins, such as those that provide integration to source code management systems, such as ClearCase and Subversion. For more information, see the Integrating with Source Control Systems topic in the *Composer 8.0.4 Help*.

Adding a Media Server Block

The first block in the IPD shown in Figure 27 on page 60 is a Media Server block. Use the Media Server block to direct interactions of a particular media type (other than voice) into an IPD.

Endpoints

A media server is associated with one or more *endpoints*, with each endpoint connecting the media server to an interaction queue. In order for a Media Server block to show endpoints, those endpoints must first exist in the Configuration Database. Then, after you select the media server application via the `Application` property, the endpoint ports appear on the Media Server block as well as being listed in the `Properties` view. 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...	""

Figure 34: Media Server Endpoints

Procedure: Adding a Media Server Block

Note: Adding a Media Server block in an IPD is a convenience, but is not technically necessary. In Genesys Configuration Manager or Genesys Administrator, you can go into the Application object for the media server that will be handling your interactions, and create and configure the endpoints there via the Options tab. This will allow you to configure how the interactions get placed into interaction queues.

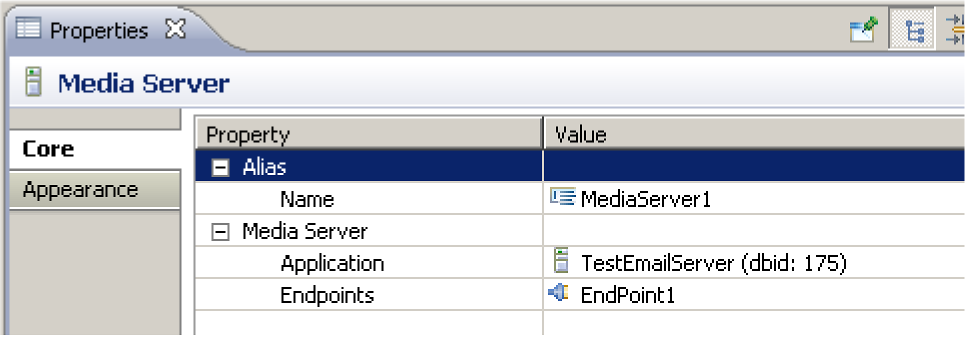
Start of procedure

1. Outside of Composer, in Genesys Administrator or Configuration Manager, use the instructions in the *eServices (Multimedia) 8.0 Deployment Guide* to create the media server Application object and its endpoints.
2. Drag and drop the Media Server block onto the canvas. Or use one of the methods covered in “Adding Blocks to the Canvas” on page 87 to add the Media Server block to the canvas.
3. Configure the following fields in the Properties view as shown in Figure 35:
   a. Name—Name the Media Server block in the IPD.
   b. Application—Select a media server to specify the CfgApplication object in Configuration Server that this block represents.
   c. Endpoints—Click under Value to open a dialog box where you can select one or more endpoints, which can then be connected to interaction queues. 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.
4. Click the File menu and select Save.

End of procedure

IPD Block Properties

The Properties view for the MediaServer1 block is shown in Figure 27 on page 60 is shown below in Figure 35.



Property	Value
Alias	
Name	MediaServer1
Media Server	
Application	TestEmailServer (dbid: 175)
Endpoints	EndPoint1

Figure 35: Properties View for MediaServer1 Block

Defining an Interaction Queue

The second block in the IPD shown in Figure 27 on page 60 is an Interaction Queue block. Use it to define a multimedia (non-voice) interaction queue in an IPD and to create a *view*, which can define the conditions for pulling interactions out of the queue for submittal to a workflow. Publishing (see page 85) pushes this information into the Configuration Database as a `CfgScript` object of type `InteractionQueue`.

About Interaction Queues:

- Queues that you defined with the Interaction Queue block appear for selection in the Queue Interaction block.
- By design, each Interaction Queue block has only one green output port, which can only be connected to a Workflow block in the IPD.
- No updates to Configuration Server are created until you invoke the Publish operation described on page 85.
- 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 the following topic in *Composer 8.0.4 Help*: Linking IPDs with Workflows.

Procedure: Defining an Interaction Queue

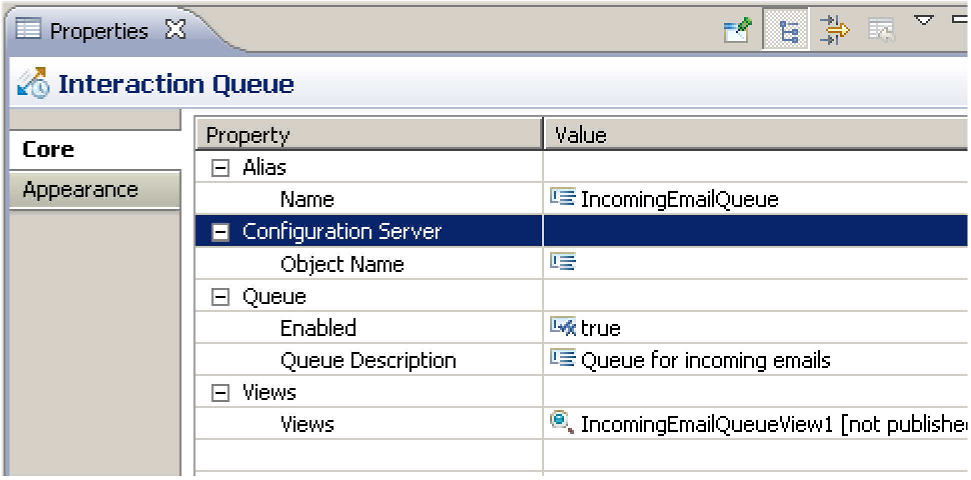
Start of procedure

1. Drag and drop the Interaction Queue block onto the canvas. Or use one of the methods covered on "Adding Blocks to the Canvas" on page 87 to add an Interaction Queue block.
2. Configure the following fields in the `Properties` view as shown in Figure 36:
   a. `Name`—Use this property to name the Interaction Queue block in the IPD.
   b. `Queue Enabled`—Select `true` or `false` to enable or disable this queue in Configuration Server.
   c. `Queue Description`—Enter a description for the interaction queue. This property will map to the `Description` key in the Annex section `Queue` of the `CfgScript` object for the interaction queue.
   d. `Queue Name`—Mandatory. You must enter a name for the interaction queue, which will appear on the block under the value for the `Name` property. No updates to the Configuration Database are created until you invoke the Publish operation.
   e. `Views`—Use this property to define one or more views for an interaction queue. Each view represents an exit channel from the queue into a workflow.
3. Click the `File` menu and select `Save`.

End of procedure

IPD Block Properties

The `Properties` view for the `IncomingEmailQueue` Interaction Queue block shown in Figure 27 on page 60, is shown in Figure 36.



Interaction Queue

Property	Value
⊟ Alias	
Name	IncomingEmailQueue
⊟ Configuration Server	
Object Name	
⊟ Queue	
Enabled	true
Queue Description	Queue for incoming emails
⊟ Views	
Views	IncomingEmailQueueView1 [not publishe

Figure 36: Properties View for IncomingEmailQueue Block

The next procedure describes configuring the `Views` property shown in Figure 36.

Procedure: Defining a view for an interaction queue

Purpose: To define an exit channel from an interaction queue into a workflow. The view defines the criteria that must be met before an interaction can exit out from the interaction queue. A view can also enable certain types of interactions to be processed earlier than others. It is mandatory to define at least one view for an interaction queue in an IPD although you don't have to define specific conditions.

Start of procedure

1. In an Interaction Queue block, click under `Value` to display a ⬚ button.
2. Click the ⬚ button to open the `View Properties` dialog box.
3. Click `Add` to display `Main`, `Parameterized Conditions`, and `Segmentation` tabs (see Figure 37 on page 75).
4. Complete the `Main` tab. Continuing with this re-creation, use the entries in Figure 37. Summary information on each field is presented below.
   a. `Enabled`—Check the box to make the view ready to extract interactions.
   b. `Name`—Enter a name for the view to be used when saving as a Configuration Database `Script` object of type `InteractionQueueView`.
   c. `Description`—Enter text describing the view.

d. 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 property.

e. Condition—You have the option of creating an expression to be used as the basis for extracting interactions from the queue.

f. 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: "priority desc, ReceivedAt"

   Extracts according to priority and the received at timestamp if priority is the same. For more information on the interaction attributes that can be used for ordering, see the section on System Properties in the chapter on Interaction Properties in the *eServices (Multimedia) 8.0 User's Guide*.

g. 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 8.0 User's Guide.*

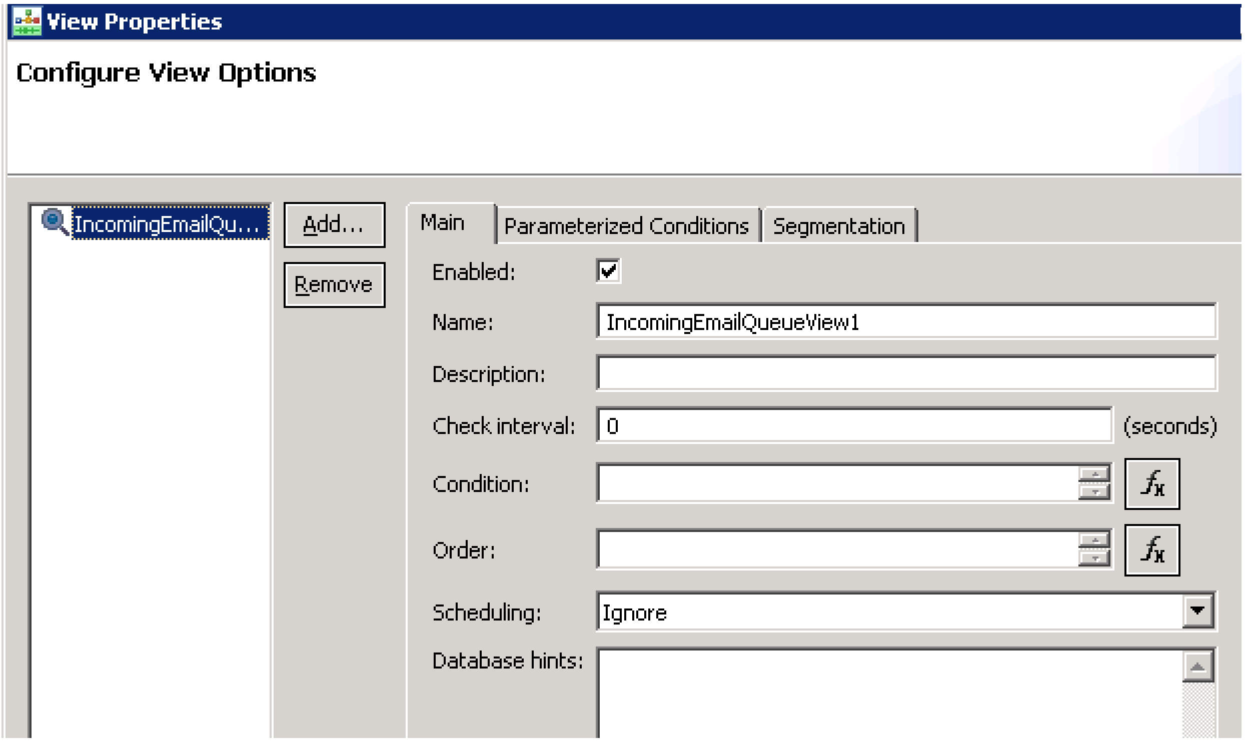
h. Database Hints—This field is only applicable to an Oracle database. You can apply a Hint, which will cause Oracle to use a specific index to optimize performance.

Note: For detailed information on all tabs, consult the *Composer 8.0.4 Help*, Interaction Queue Views topic.

5. After you complete the applicable fields in these tabs, click OK to close the View Properties dialog box. After publishing, each view will be created as a separate CfgScript object of type InteractionQueueView.
6. Select File > Save from the menu to save the IPD as it exists so far.

End of procedure

Figure 37 shows the View Properties dialog box for the IncomingEmailQueue defined in the first Interaction Queue block in the IPD.



View Properties

Configure View Options

IncomingEmailQu...

Enabled: ☑

Name: IncomingEmailQueueView1

Description:

Check interval: 0 (seconds)

Condition:

Order:

Scheduling: Ignore

Database hints:

Figure 37: View Properties Dialog Box for IncomingEmailQueue

The `Main` tab indicates no special condition for pulling interactions from the queue. The `Parameterized Conditions` and `Segmentation` tabs are not used in the Project.

Note: Important! While you can define multiple views for an interaction queue, the Interaction Queue block will have only one output port and therefore will feed interactions to only one workflow strategy.

Connecting the Blocks

So far, we have created the Media Server block and the first Interaction Queue block. At this point, you can connect the Media Server block Endpoint to the top input port of the Interaction Queue block. If necessary, use the information in “Connecting Blocks” on page 88.

IPD Flow So Far

When an e-mail (new multimedia interaction) arrives at the Media Server `Endpoint1`, it gets queued to `IncomingEmailQueue` (see Figure 36 on page 73).

Creating a Workflow Diagram

Prior to adding a Workflow block, you may wish to configure the workflow diagram or SCXML file to which interactions should be sent for processing. You can then select it in the Workflow block properties.

Note: The procedure below is a very high-level summary of the steps to create a workflow diagram. A detailed example is available in the topic "Your First Application: DNIS Routing" in the Building SCXML-Based Workflows book of the *Composer 8.0.4 Help.*

Procedure:
Creating a workflow diagram

Start of procedure

1. Click the button on the main toolbar to create a new workflow (see Figure 21 on page 39) 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.
   - Right-click the `Workflows` folder in the Project Explorer and select `New` > `Other` > `Workflow Diagram`. 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. In this example, it is `RoutingAfterSendingAutoResponse`.
4. Name the diagram (must have an extension of `.workflow`). In this example, the name is `RouteToAgentWithAutoResponse.workflow` (see Figure 41 on page 80).
5. Click `Finish`. The `Workflows` folder in the Project Explorer shows the name of your diagram under your Project.
6. Since each workflow diagram starts with an Entry block and ends with an Exit block, you can place those blocks now. The Entry block is where you define variables (see "Variables: Project and Application" on page 31).
7. Build the workflow diagram.
8. Validate the code by selecting `Diagram` > `Validate`. You can also click the Validate icon (see Figure 21 on page 39) 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.

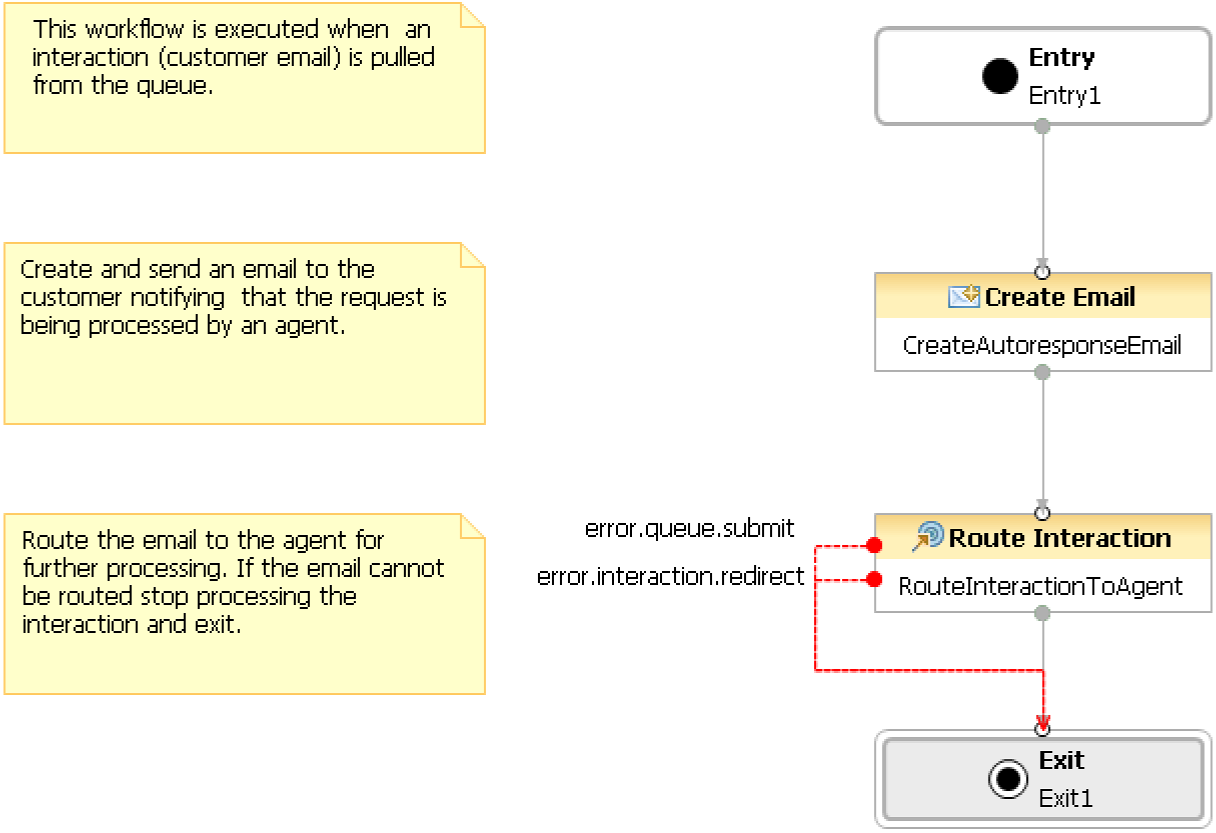
9. Generate the code (see page 86).

End of procedure

Continuing with our routing application re-creation, build the routing strategy shown in Figure 38. Use the `Properties` view entries shown in Figure 39 on page 78 and Figure 40 on page 79.

RouteToAgentWithAutoResponse.workflow

The workflow diagram referenced by the `RouteToAgent Workflow` block in Figure 27 on page 60 is shown in Figure 38.



This workflow is executed when an interaction (customer email) is pulled from the queue.

Create and send an email to the customer notifying that the request is being processed by an agent.

Route the email to the agent for further processing. If the email cannot be routed stop processing the interaction and exit.

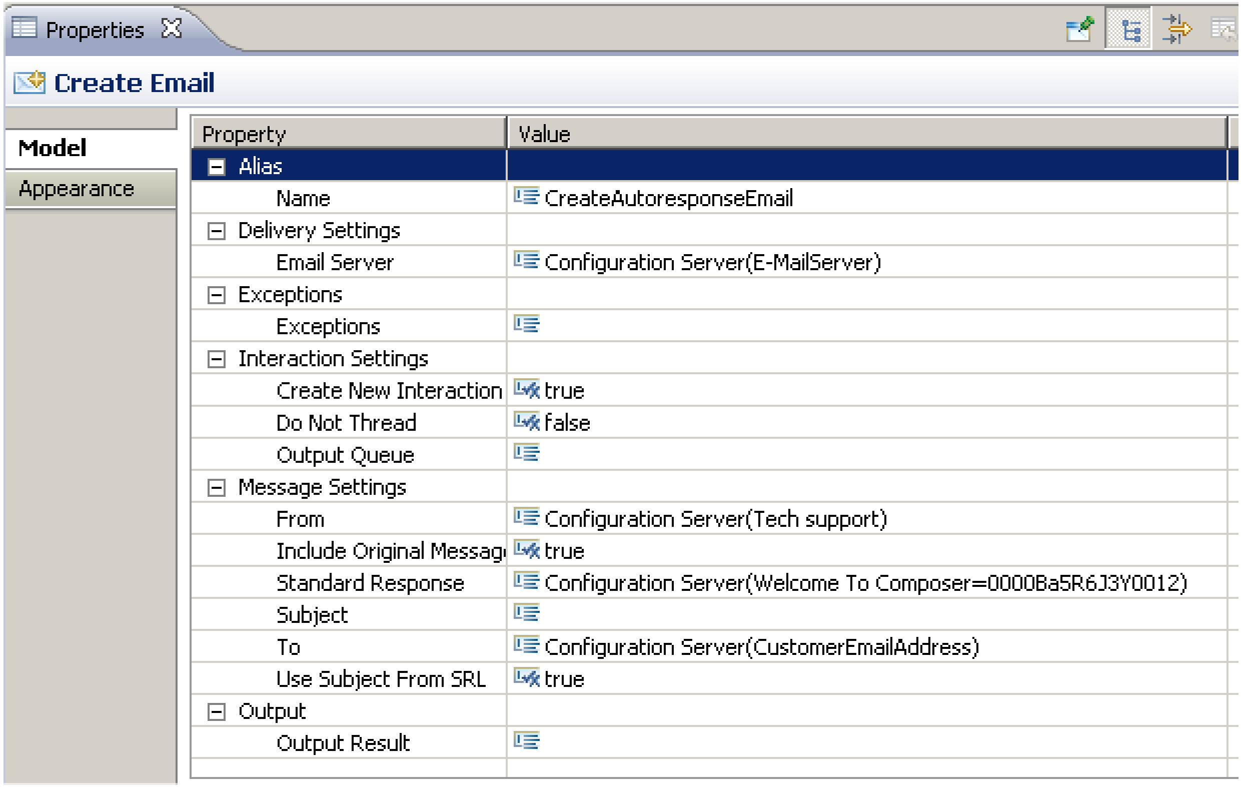
Figure 38: RouteToAgentWithAutoResponse Workflow

The `RouteToAgentWithAutoResponse` workflow:

- Uses the Create E-mail block to send out an auto-response e-mail to the customer notifying that the system has received their e-mail and someone will be contacting them shortly.
- Uses the Route Interaction block to route the original e-mail from the customer to an agent group for a response. This block also specifies a suggested queue for the new interaction (agent's response e-mail).

Workflow Diagram Block Properties

The `Properties` view for the `CreateAutoresponseEmail` block in Figure 38 is shown in Figure 39.

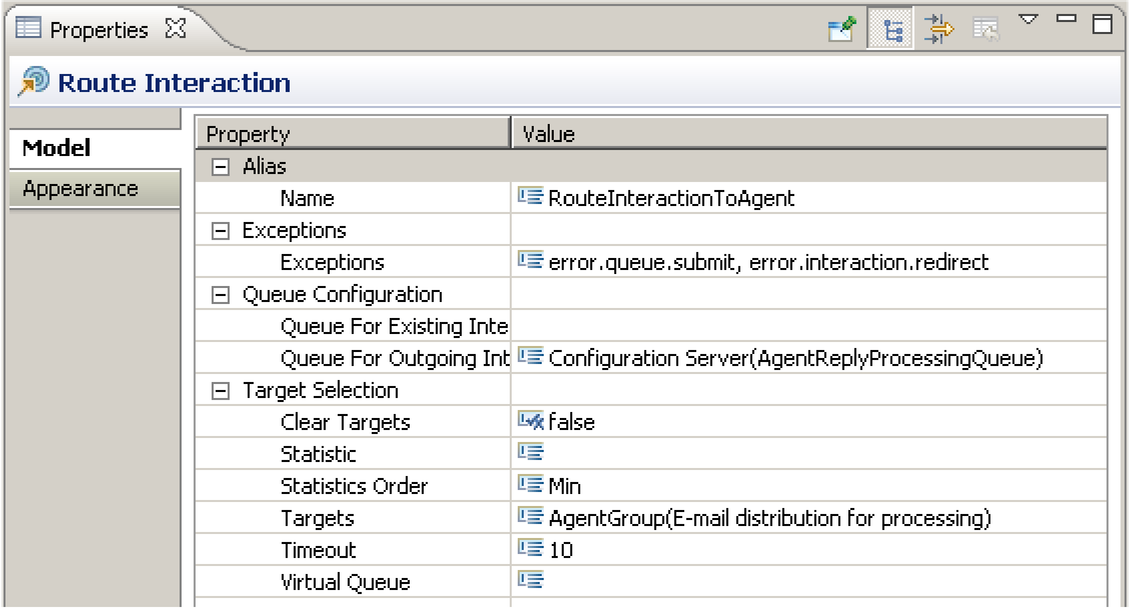


Create Email

Property	Value
⊟ Alias	
Name	CreateAutoresponseEmail
⊟ Delivery Settings	
Email Server	Configuration Server(E-MailServer)
⊟ Exceptions	
Exceptions	
⊟ Interaction Settings	
Create New Interaction	true
Do Not Thread	false
Output Queue	
⊟ Message Settings	
From	Configuration Server(Tech support)
Include Original Messag	true
Standard Response	Configuration Server(Welcome To Composer=0000Ba5R6J3Y0012)
Subject	
To	Configuration Server(CustomerEmailAddress)
Use Subject From SRL	true
⊟ Output	
Output Result	

Figure 39: Create AutoresponseEmail Block Properties

The `Properties` view for the `RouteInteractionToAgent` block Figure 38 is shown in Figure 40.



Property	Value
⊟ Alias	
Name	RouteInteractionToAgent
⊟ Exceptions	
Exceptions	error.queue.submit, error.interaction.redirect
⊟ Queue Configuration	
Queue For Existing Inte	
Queue For Outgoing Int	Configuration Server(AgentReplyProcessingQueue)
⊟ Target Selection	
Clear Targets	false
Statistic	
Statistics Order	Min
Targets	AgentGroup(E-mail distribution for processing)
Timeout	10
Virtual Queue	

Figure 40: RouteInteractionToAgent Block Properties

Adding a Workflow Block

The third block in the IPD we are re-creating (see Figure 27 on page 60) is a Workflow block. Use this block in an IPD to point to a workflow resource (workflow diagram or SCXML file) to which interactions should be sent for processing.

Note: The `RouteToAgentWithAutoResponse` workflow diagram is already configured (see Figure 38 on page 77) in this example, so it can be selected from the Workflow block.

Procedure: Adding a Workflow block

Start of procedure

1. Drag and drop a Workflow block onto the canvas. Or use one of the methods covered on “Adding Blocks to the Canvas” on page 87 to add the Workflow block to the canvas.
2. Configure the following properties using the entries in Figure 41.
   a. `Name`—Use this property to name the Workflow block in the IPD. Continuing with the routing application re-creation, name the block `RouteToAgent` as shown in Figure 27 on page 60.

b. Resource—Use this property to point to a workflow diagram or SCXML file.

i. Click under Value to display the … button.

ii. Click the … button to open the Select Resource dialog box.

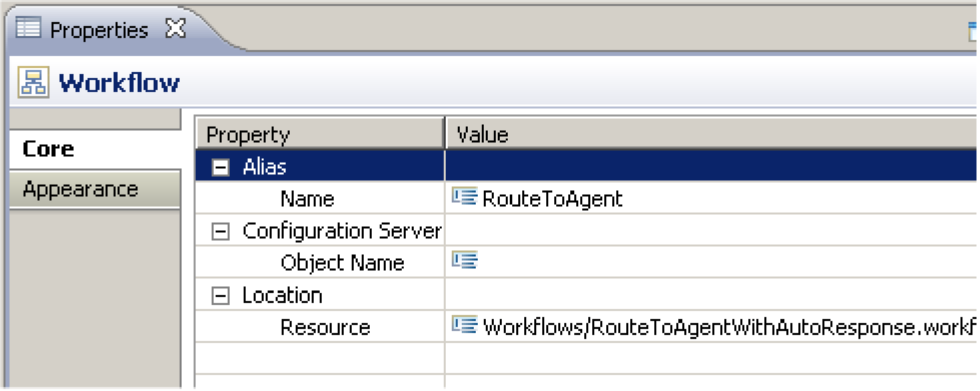
iii. Select the workflow resource, which can be a workflow (strategy) diagram that exists in any of the Projects in the Composer workspace or an SCXML file created in Composer's SCXML Editor (see page 20). In this case, select the RouteToAgentWithAutoresponse workflow (see Figure 38 on page 77).

iv. Click OK.

End of procedure

IPD Block Properties

The Properties view for the RouteToAgent Workflow block in Figure 27 on page 60 is shown in Figure 41.



Properties

Workflow

Core

Appearance

Property	Value
⊟ Alias	
Name	RouteToAgent
⊟ Configuration Server	
Object Name	
⊟ Location	
Resource	Workflows/RouteToAgentWithAutoResponse.workf

Figure 41: Properties View for RouteToAgent Workflow Block

Workflow-Generated Blocks

Workflow-generated blocks help you visually interpret an IPD. They take the form of outgoing connections and plain white blocks, that automatically appear from a Workflow block. They represent objects specified inside the Workflow block, such as queues, routing targets, and stop processing instructions. Accordingly, Composer generates the following types of workflow-generated blocks:

- Queue Reference
- Dynamic Target
- Stop

Since these blocks are automatically generated by Composer, they are not available on the IPD palette. Composer uses a different color, appearance, and structure for them allowing you to easily differentiate them from other blocks.

In the IPD we are re-creating (see Figure 27 on page 60), there are three workflow-generated blocks:

- `CreateAutoResponse` Queue Reference
- `RouteInteractionToAgent` Dynamic Target
- `RouteIxnToSupervisor` Dynamic Target

Connecting the Blocks

Previously we created and connected the Media Server block and the first Interaction Queue block in Figure 27 on page 60. You can now connect the first Interaction Queue block to the first Workflow block in the flow. If necessary, use the information in "Connecting Blocks" on page 88.

IPD Flow So Far

1. When a new multimedia interaction (an e-mail in this case) arrives at the Media Server `Endpoint1`, it gets queued to `IncomingEmailQueue` (see Figure 36 on page 73).
2. Orchestration Server then pulls the interaction from the queue and starts processing the workflow referenced by the `RouteToAgent` workflow block (see Figure 41 on page 80).
   a. The `RouteToAgentWithAutoresponse` workflow uses the Create E-mail block (see Figure 38 on page 77) to send out an auto-response e-mail (see `Standard Response` property in Figure 39 on page 78) to the customer, notifying that the system has received their e-mail and someone be contacting them shortly.
   b. After sending the auto-response e-mail, the original e-mail from the customer is then routed to the `E-mail distribution for processing` agent group for a response (see the `Targets` property in Figure 40 on page 79). The Route Interaction block is used for this purpose.
   c. The Route Interaction block also specifies a suggested queue for the new interaction (see the `Queue for Outgoing interaction` property in Figure 40 on page 79).

Note: The next two steps (3 and 4) occur outside of the IPD, but could be documented in the IPD with Notes (although this has not been done in the Project template shown in Figure 27 on page 60).

3. After the agent responds (using Reply feature in Genesys Agent Desktop), the response is now treated as a new interaction and is then placed into the suggested queue.

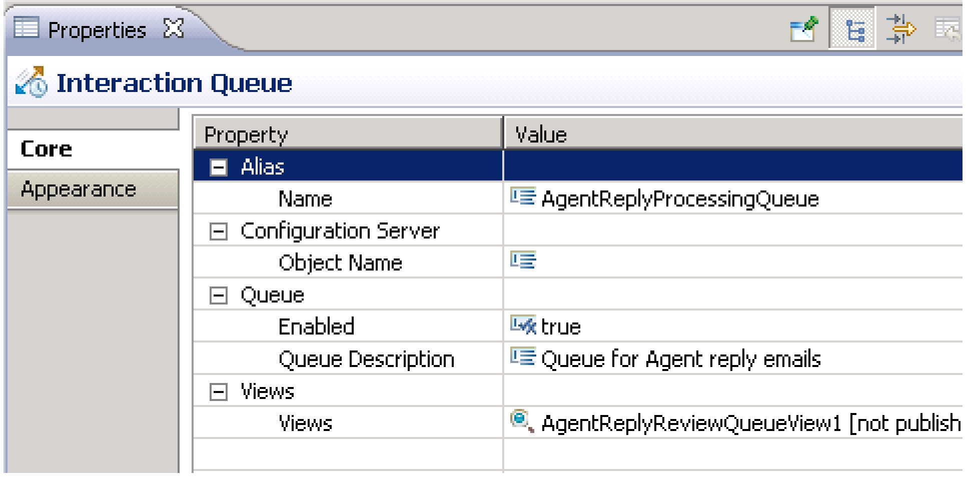
4. The agent then closes the current interaction (using Done feature in Genesys Agent Desktop).

Adding the Remaining IPD Blocks

To finish creating the IPD shown in Figure 27 on page 60, we must add another Interaction Queue block and another Workflow block.

Defining Second Interaction Queue

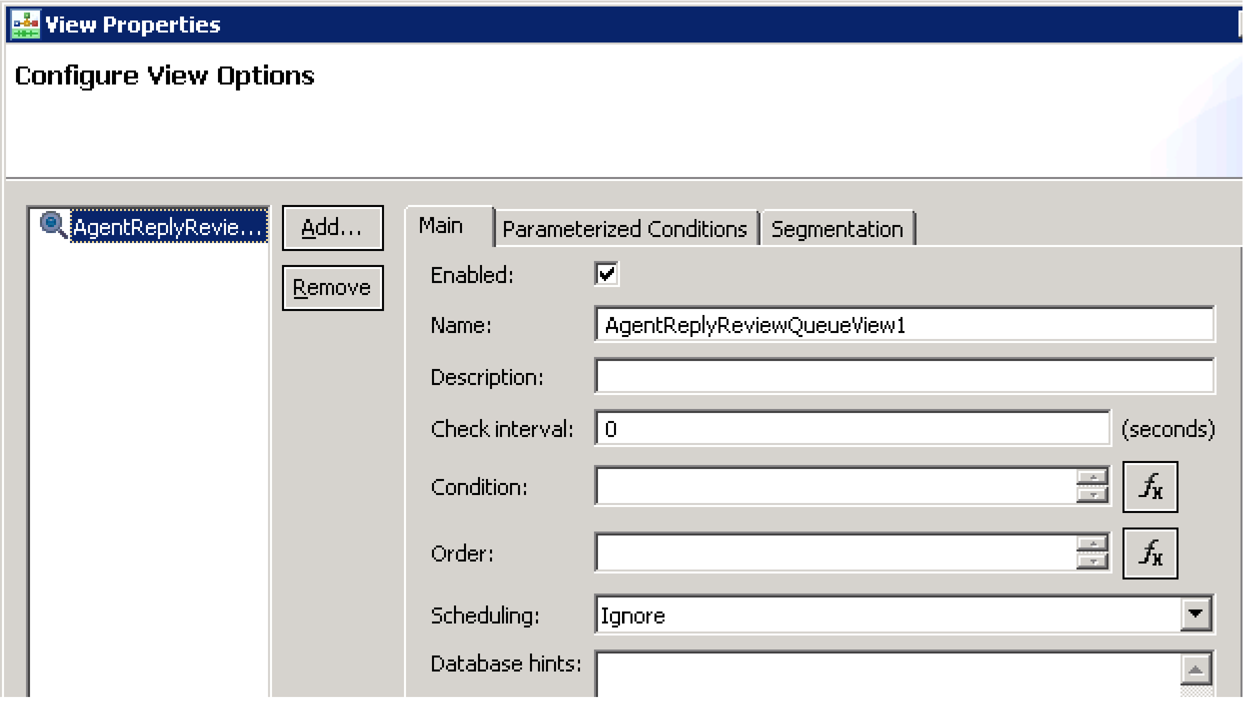
The sixth block in the IPD shown in Figure 27 on page 60 is an Interaction Queue block as previously described on page 71. The function of this Interaction Queue block is to define the interaction queue for the agent reply e-mail. As shown in Figure 40 on page 79, the queue for the outgoing interaction is AgentReplyProcessingQueue. This name is reflected in both the name of the Interaction Queue block and in the Properties view for the block (see Figure 42).



Property	Value
⊟ Alias	
Name	AgentReplyProcessingQueue
⊟ Configuration Server	
Object Name	
⊟ Queue	
Enabled	true
Queue Description	Queue for Agent reply emails
⊟ Views	
Views	AgentReplyReviewQueueView1 [not publish

Figure 42: Properties View for AgentReplyProcessingQueue Block

As described on page 71, a view is responsible for pulling interactions out of queues for submittal to a workflow. Figure 43 shows the view configuration for the AgentReplyProcessingQueue.



Enabled: ☑

Name: AgentReplyReviewQueueView1

Description:

Check interval: 0 (seconds)

Condition:

Order:

Scheduling: Ignore

Database hints:

Figure 43: View Properties Dialog Box for AgentReplyProcessing Queue

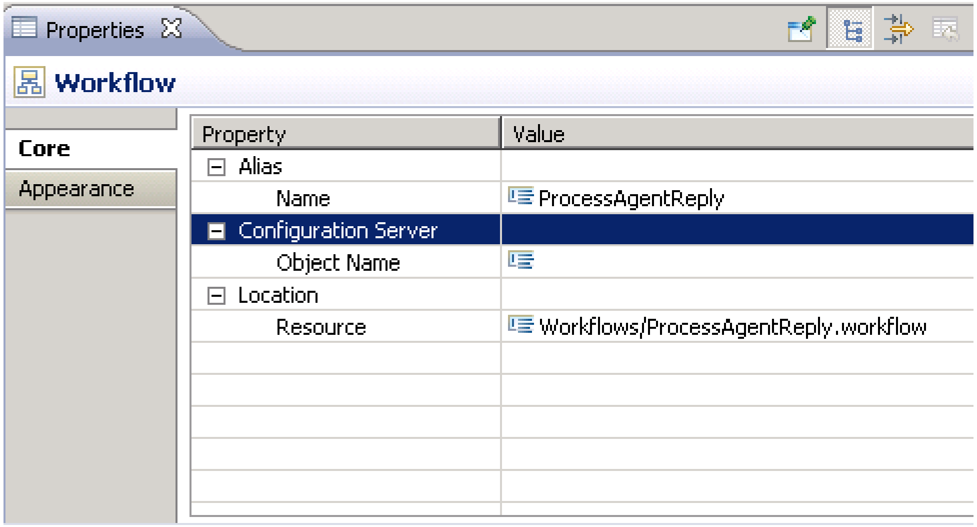
The `Main` tab indicates no special condition for pulling interactions from the queue. The `Parameterized Conditions` and `Segmentation` tabs are not used in the Project. Fields in this dialog box were previously summarized on page 73.

Adding a Second Workflow Block

The seventh block in the IPD shown in Figure 27 on page 60 is another Workflow block as previously described on page 79. The function of `ProcessAgentReply Workflow` block is to specify the next workflow resource to which interactions should be sent for processing.

Note: The process for adding a Workflow block was previously described in Procedure: Adding a Workflow block, on page 79.

As can be seen in the `Properties` view for this second Workflow block, the name of the workflow resource is `ProcessAgentReply.workflow` (see Figure 44).

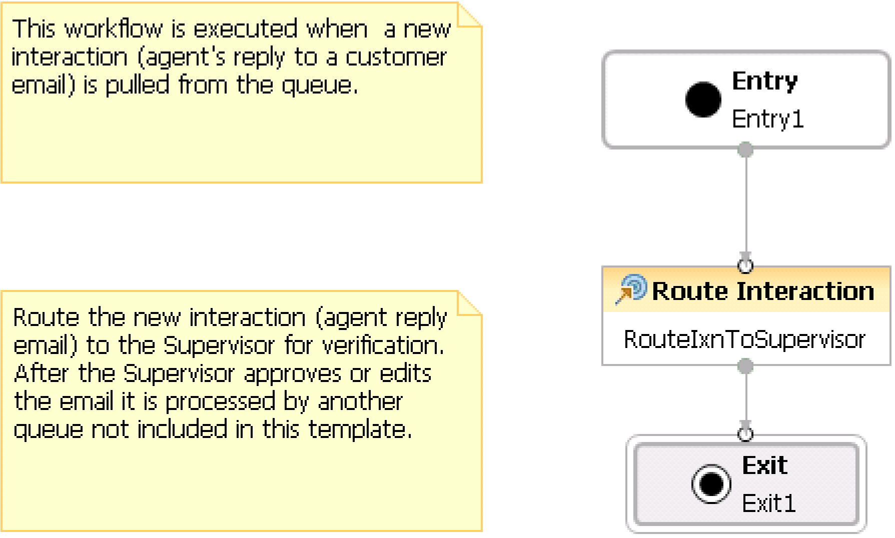


Property	Value
⊟ Alias	
Name	ProcessAgentReply
⊟ Configuration Server	
Object Name	
⊟ Location	
Resource	Workflows/ProcessAgentReply.workflow

Figure 44: Properties View for ProcessAgentReply Workflow Block

ProcessAgentReply.workflow

The workflow diagram referenced by the `ProcessAgentReply` Workflow block in Figure 27 on page 60 is shown in Figure 45.



This workflow is executed when a new interaction (agent's reply to a customer email) is pulled from the queue.

Route the new interaction (agent reply email) to the Supervisor for verification. After the Supervisor approves or edits the email it is processed by another queue not included in this template.

Figure 45: ProcessAgentReply Workflow

When Orchestration Server pulls the new interaction from the `AgentReplyProcessing` queue, the `ProcessAgentReply` workflow gets executed.

- ProcessAgentReply is a simple workflow which routes the agent response e-mail to a supervisor for review.
- Once the supervisor is satisfied with the response, the e-mail 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.

Publishing an IPD

Publishing an IPD validates Project configuration information and pushes the information out to Configuration Server. For example, when you configure an Interaction Queue block or a Workflow 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. Or you can set preferences to automatically publish upon saving.

Procedure:
Publishing an interaction process diagram

Start of procedure

1. Before publishing, you must connect to Configuration Server as described on page 64.
2. Right-click an IPD in the `Interaction Processes` folder in the Project Explorer.
3. Select `Publish to Configuration Server`.
4. As an alternative to the above steps, click the tab on the canvas that contains the IPD and click the toolbar button to publish the active IPD to Configuration Server (see Figure 21 on page 39).

End of procedure

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 (see Figure 34 on page 69), you must set “endpoint” point key in the `pop-clientX` section of the e-mail server Application to the correct end-point. For more details, see the Deploying a Routing Application topic in the Validation, Debugging, & Deployment book of the *Composer 8.0.4 Help*.

Generating Code

You must generate source code for both the IPD and associated workflow diagrams. All generated files go into the `src-gen` folder of the Project.

Procedure:
Generating code for an interaction process diagram

Start of procedure

1. Click the tab on the canvas that contains the IPD.
2. Click the toolbar button to generate code (see Figure 21 on page 39). The generated file name in the src-gen folder will follow the format:
   `<ipd_diagram_name>_<workflow_block_name>.scxml`

End of procedure

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.

Deploying the Application

Deploying a routing application on an application server is beyond the scope of this guide. For information on this subject, consult the *Composer 8.0.x Help.*

Chapter

6

Working with Blocks and Diagrams

This chapter describes how to work with Composer diagram editor. It contains the following sections:

Working with Blocks

In Figure 48 on page 91, note the various categories of workflow diagram blocks: `Flow Control`, `Routing`, `Voice Treatments`, `Server Side`, `Context Services`, and `eServices`. See Appendix A, "Composer Blocks" on page 103 for information on blocks in each category and deciding which blocks to use.

To display blocks for a given category, simply click the category name; for example, click `Flow Control`.

Adding Blocks to the Canvas

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.

Connecting Blocks

Blocks are connected to each other using connection links (see `Output Link` and `Exception Link` under `Palette` in Figure 48 on page 91. There are two types of connection links:

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

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

Method #1

Procedure: Using the Output Link to connect blocks

Start of procedure

1. Click the `Output Link` (or `Exception Link)` icon on 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.

End of procedure

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 (see Figure 13 on page 31).

Method #2

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 `Output Link` (or `Exception Link`) in the palette to create a connection between the two blocks.

Method #3

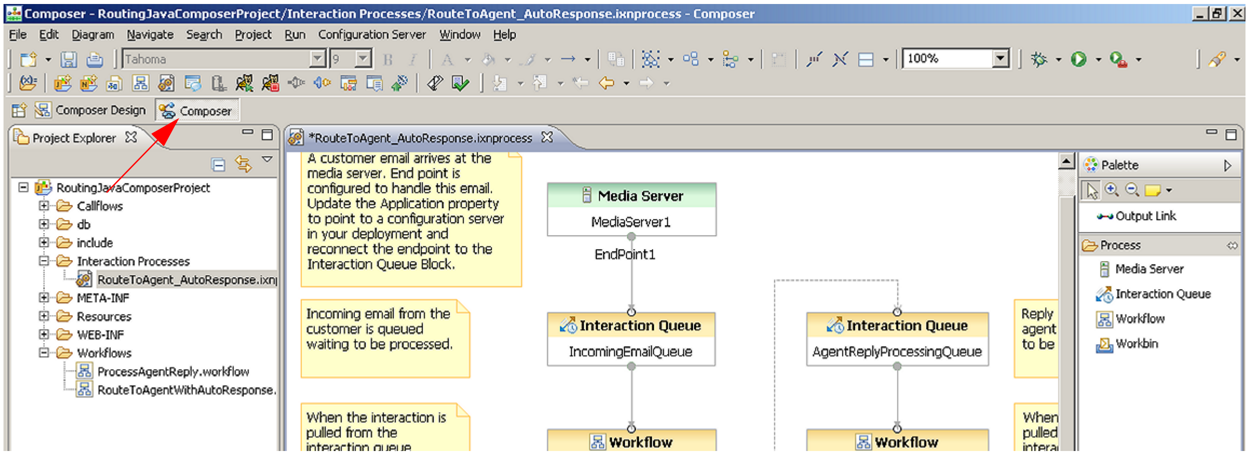
Some users may find this method convenient:

1. Start dragging from a connection port of the originating block.
2. Drop the block into a blank portion on a canvas (any place in the canvas other than a block will also do). A menu representation of the palette will pop up showing a list of all block types.
3. Click the correct block type. A new block of that type will be added and the originating block will be connected to it.

Note: When connecting to a Treatment block, it is your responsibility to verify that your Switch/T-Server combination supports the treatment you select. Composer does not perform a compatibility check.

Perspectives

Within Composer, a perspective is an arrangement of different sections of the GUI in a manner that facilitates easy use of a particular feature, such as design or debugging. All of the figures shown so far in this chapter show `Composer` perspective as indicated by the `Composer` button being selected underneath the toolbar (see Figure 46).


Composer - RoutingJavaComposerProject/Interaction Processes/RouteToAgent_AutoResponse.ixnprocess - Composer
File Edit Diagram Navigate Search Project Run Configuration Server Window Help
Tahoma
9
100%
Composer Design
Composer
Project Explorer
RoutingJavaComposerProject
Callflows
db
include
Interaction Processes
RouteToAgent_AutoResponse.ixn
META-INF
Resources
WEB-INF
Workflows
ProcessAgentReply.workflow
RouteToAgentWithAutoResponse.
*RouteToAgent_AutoResponse.ixnprocess
Media Server
MediaServer1
EndPoint1
Interaction Queue
IncomingEmailQueue
Interaction Queue
AgentReplyProcessingQueue
Workflow
Workflow
Palette
Output Link
Process
Media Server
Interaction Queue
Workflow
Workbin

A customer email arrives at the media server. End point is configured to handle this email. Update the Application property to point to a configuration server in your deployment and reconnect the endpoint to the Interaction Queue Block.

Incoming email from the customer is queued waiting to be processed.

When the interaction is pulled from the interaction queue

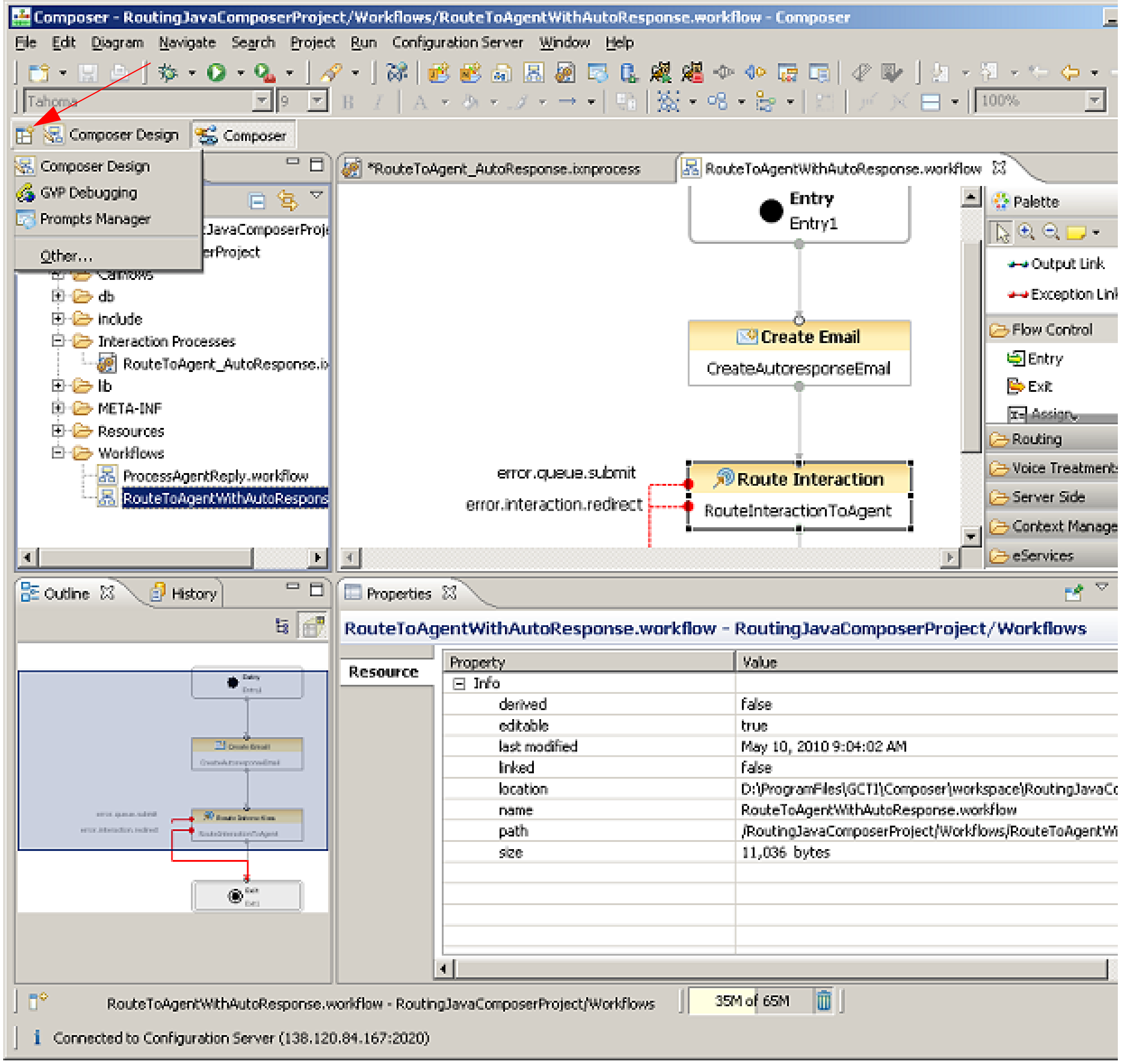
Reply agent to be

When pulled intera

Figure 46: Bar for Perspective Buttons, Composer Perspective Selected

Changing Perspectives

To change perspectives, click the button to open a perspective shown below to drop down a menu of available perspectives (see Figure 47):



RouteToAgentWithAutoResponse.workflow - RoutingJavaComposerProject/Workflows

Property	Value
Info	
derived	false
editable	true
last modified	May 10, 2010 9:04:02 AM
linked	false
location	D:\ProgramFiles\GCTI\Composer\workspace\RoutingJavaCo
name	RouteToAgentWithAutoResponse.workflow
path	/RoutingJavaComposerProject/Workflows/RouteToAgentWi
size	11,036 bytes

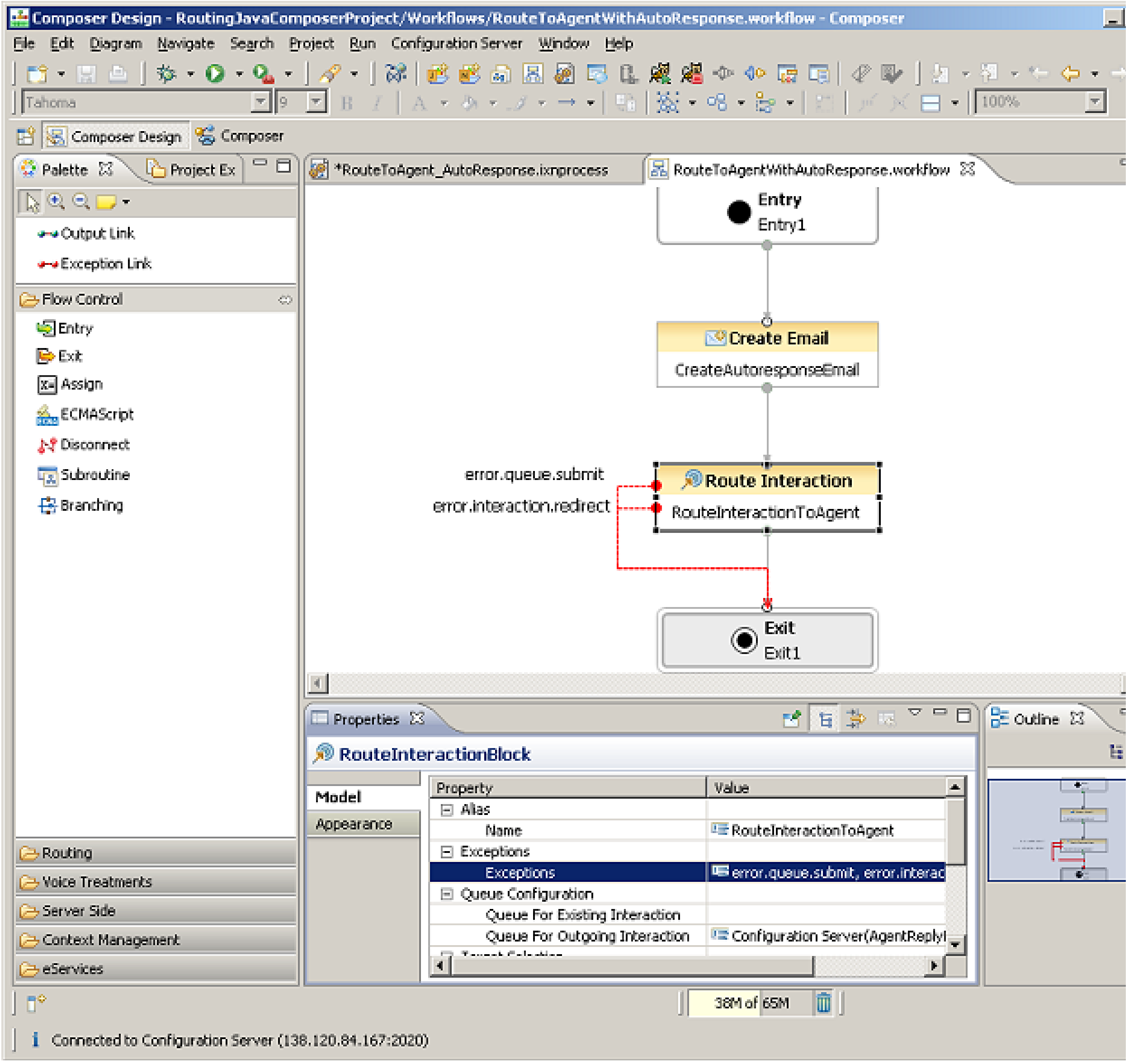
RouteToAgentWithAutoResponse.workflow - RoutingJavaComposerProject/Workflows 35M of 65M

Connected to Configuration Server (138.120.84.167:2020)

Figure 47: Open Perspective Button

When creating routing applications, you will use either `Composer` perspective or `Composer Design` perspective. The other perspectives on the menu are used for Composer voice applications (not covered in this guide) or are listed there by default since Composer is based on Eclipse (see page 10).

Assume you select `Composer Design` perspective as shown in Figure 47. The GUI changes to be more streamlined for placing, connecting and configuring blocks in the canvas as shown in Figure 48:



Property	Value
⊟ Alias	
Name	RouteInteractionToAgent
⊟ Exceptions	
Exceptions	error.queue.submit, error.interac
⊟ Queue Configuration	
Queue For Existing Interaction	
Queue For Outgoing Interaction	Configuration Server(AgentReply)

Connected to Configuration Server (138.120.84.167:2020)

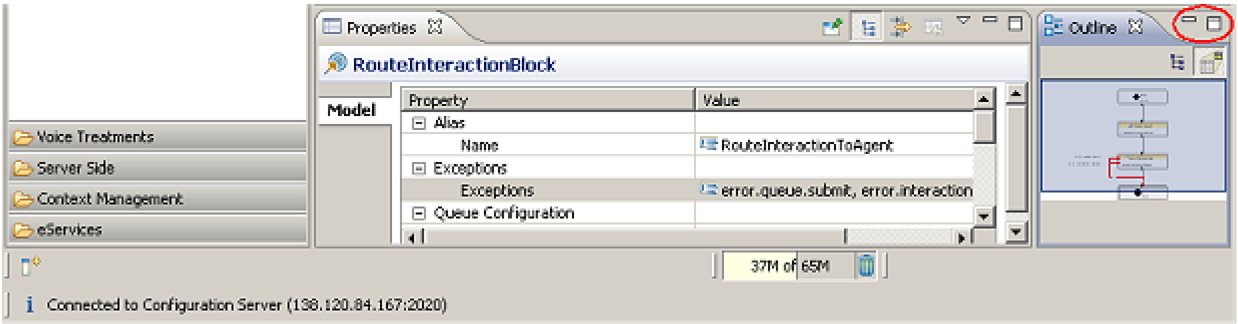
Figure 48: Composer Design Perspective

- You can also change perspectives by selecting `Window` > `Open Perspective` from the menu bar.

Note: For purposes of simplicity, this guide does not detail all ways to accomplish a task. For information on all methods, always consult the *Composer 8.0.4 Help.*

Customizing a Perspective

You can customize a perspective by maximizing/minimizing views as well as adding views. Buttons for this purpose appear on the right side of all views. Figure 49 shows these buttons on the upper right for the `Outline` view.

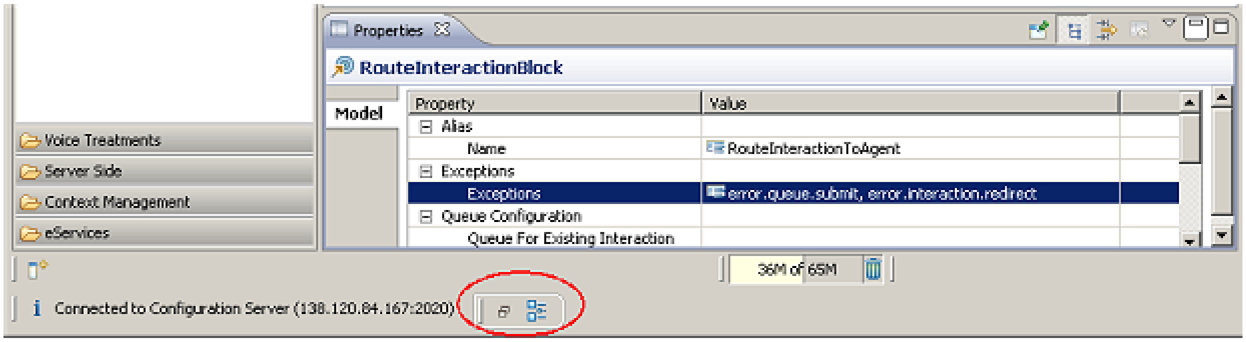


Property	Value
⊟ Alias	
Name	RouteInteractionToAgent
⊟ Exceptions	
Exceptions	error.queue.submit, error.interaction
⊟ Queue Configuration	

Connected to Configuration Server (138.120.84.167:2020)

Figure 49: Maximize and Minimize Buttons for a View

Assume that you click the minimize button. The area now appears as shown in Figure 50:



Property	Value
⊟ Alias	
Name	RouteInteractionToAgent
⊟ Exceptions	
Exceptions	error.queue.submit, error.interaction.redirect
⊟ Queue Configuration	
Queue For Existing Interaction	

Connected to Configuration Server (138.120.84.167:2020)

Figure 50: Toolbar to Restore a View

As shown above, the minimization causes a small toolbar to appear at the bottom of Composer. Click the first button (restore) to replace the view next to the `Properties` tab.

Maximization works similarly. Assume you click the maximize button. The Composer GUI shows only the `Outline` view. As a result, three small toolbars representing views that no longer appear are in close proximity to where they were originally located (see Figure 51):

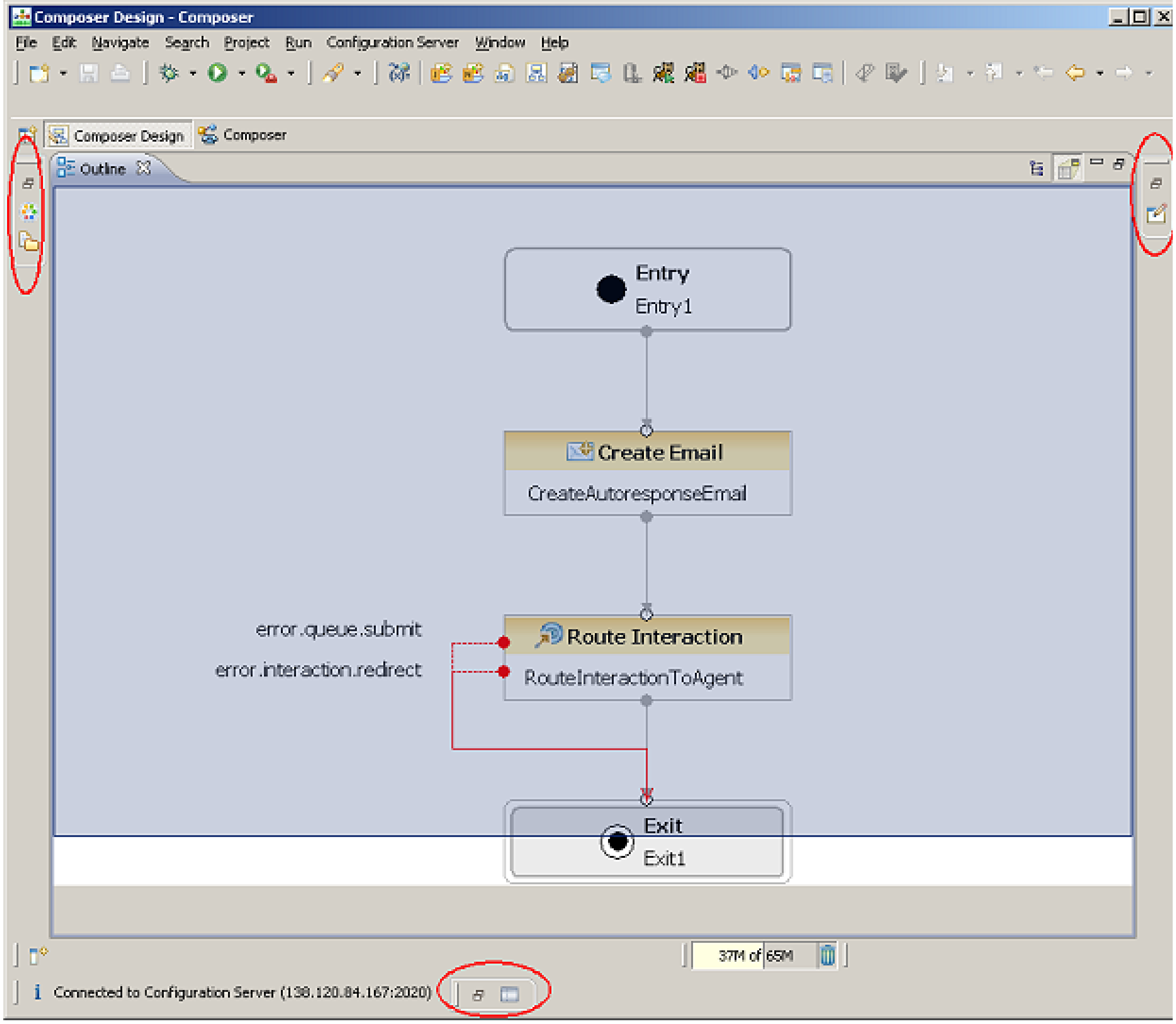

Composer Design - Composer
File Edit Navigate Search Project Run Configuration Server Window Help
Composer Design
Composer
Outline
Entry
Entry1
Create Email
CreateAutoresponseEmail
error.queue.submit
error.interaction.redirect
Route Interaction
RouteInteractionToAgent
Exit
Exit1
37M of 65M
Connected to Configuration Server (138.120.84.167:2020)

Figure 51: Composer Outline View Maximized

- Click the first button (restore) on the top left toolbar to cause Composer to appear as it did before the maximization (restore all views).
- Click the first button on the bottom toolbar to restore only the `Properties` view.
- Click the first button on the top right toolbar to restore only the editing area.

Different views can be displayed by selecting `Window` > `Show View` from the menu and then selecting a view.

Saving a Customized Perspective

The Windows menu has a `Save Perspective As` command. It also has various other perspective commands.

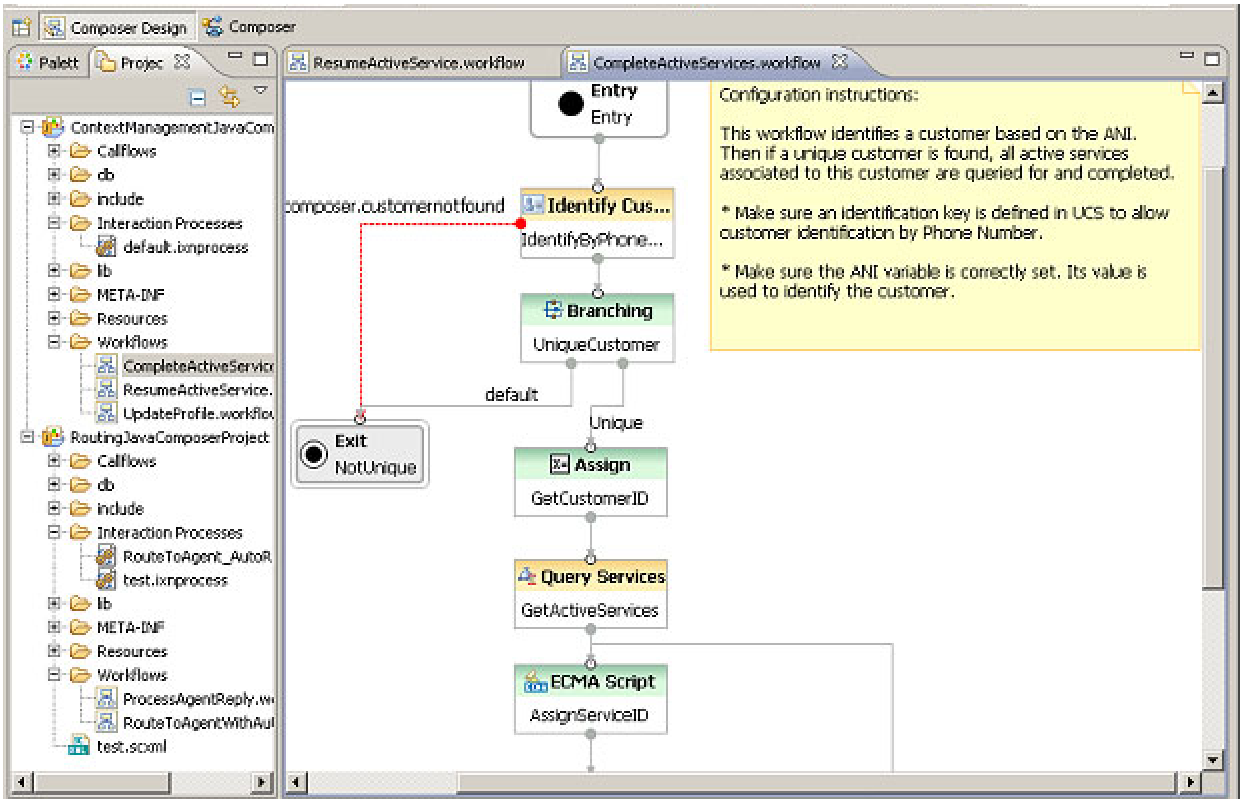
Viewing More Than One Diagram

The procedure below describes how to view more than one diagram side-by-side in Composer:

Procedure: Viewing Multiple Diagrams

Start of procedure

1. Select the diagram by double-clicking it in the `Workflows` folder of the Project Explorer. The diagram appears in the canvas area.
2. Select the next diagram. The canvas now contains two tabs as shown in Figure 52.



Configuration instructions:

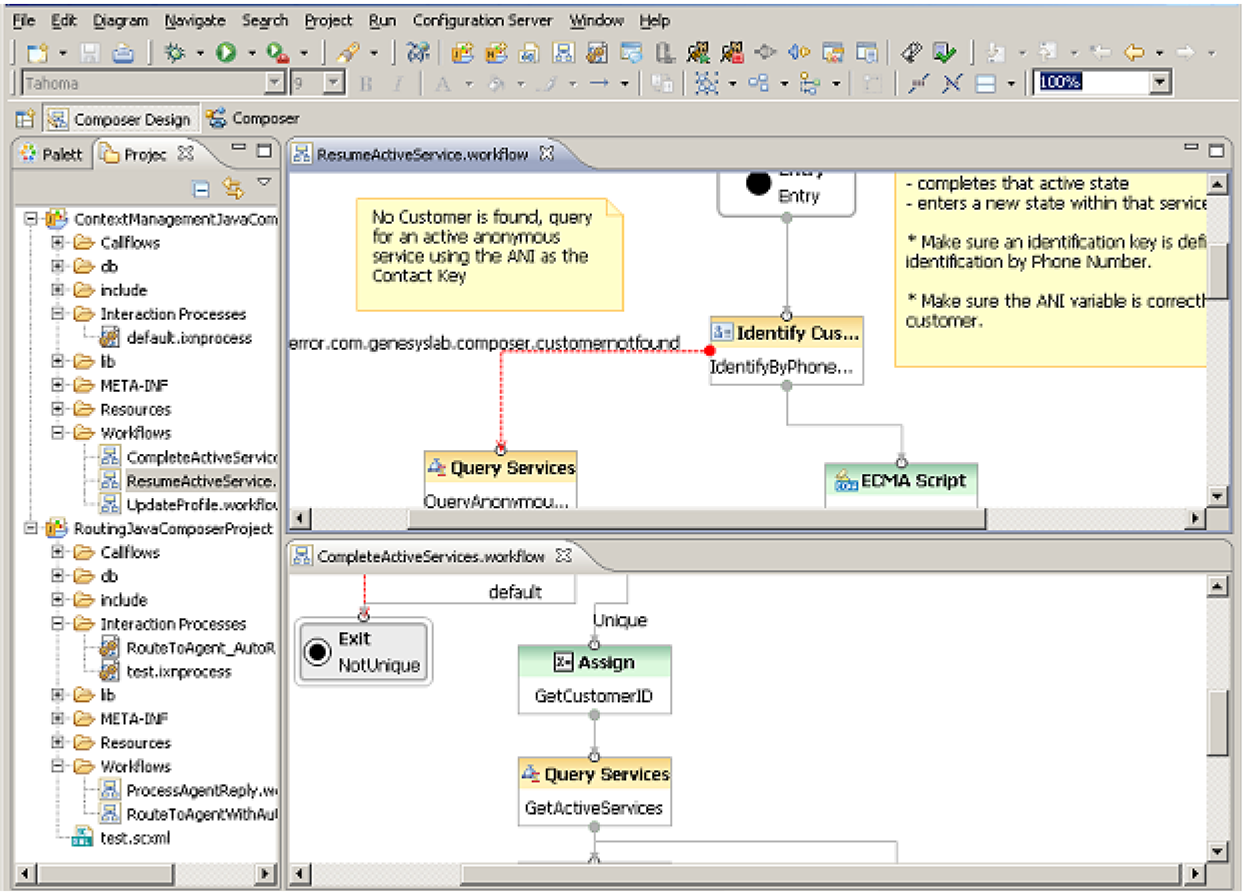
This workflow identifies a customer based on the ANI. Then if a unique customer is found, all active services associated to this customer are queried for and completed.

* Make sure an identification key is defined in UCS to allow customer identification by Phone Number.

* Make sure the ANI variable is correctly set. Its value is used to identify the customer.

Figure 52: Two Workflow Diagrams Selected from Project Explorer

3. To view diagrams, one on top of the other:
   a. Place the cursor on a tab, hold down the mouse button, and drag up. As you drag, you will notice a rectangular outline.
   b. Release the cursor. The dragged diagram appears on top of the other diagram (see Figure 53).



No Customer is found, query for an active anonymous service using the ANI as the Contact Key

- completes that active state
- enters a new state within that service

* Make sure an identification key is defi identification by Phone Number.

* Make sure the ANI variable is correctl customer.

Figure 53: Two Workflow Diagrams Visible in Canvas Area

4. To view diagrams side-by-side:
   a. Place the cursor on a tab and drag it to the side until the rectangular outline appears.
   b. Release the cursor and the dragged diagram appears on top of the other diagram.
5. You can always decrease the magnification or close the `Outline` view if necessary.

End of procedure

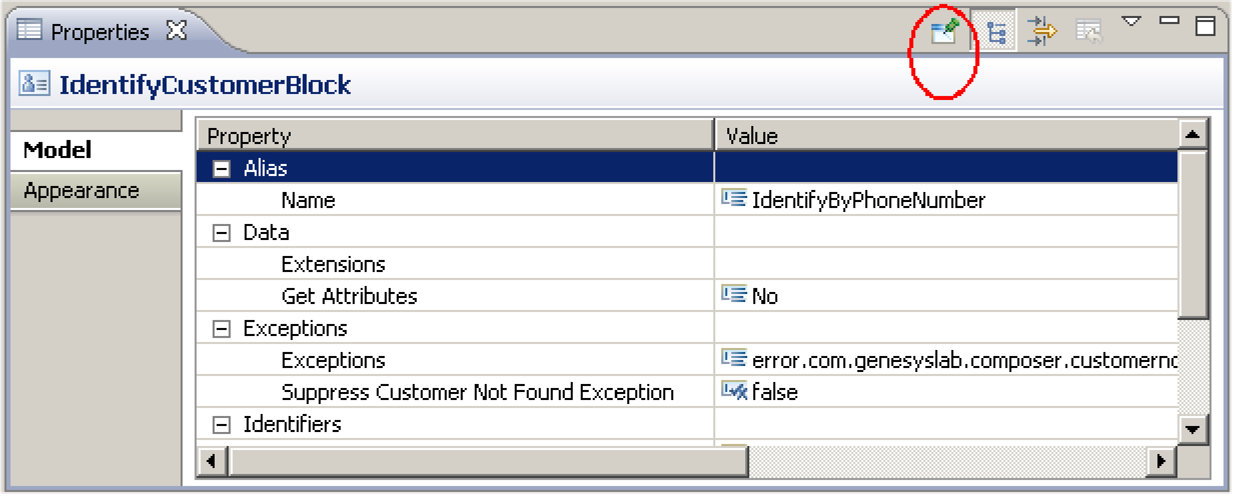
Viewing Properties for Two Blocks

The procedure below describes how to view properties of two blocks at the same time.

Procedure: Displaying Properties for Two Blocks

Start of procedure

1. Select the first block.
2. In the `Properties` tab, click the button for pinning the `Properties` view to the current selection (see Figure 54).



Property	Value
⊟ Alias	
Name	IdentifyByPhoneNumber
⊟ Data	
Extensions	
Get Attributes	No
⊟ Exceptions	
Exceptions	error.com.genesyslab.composer.customernc
Suppress Customer Not Found Exception	false
⊟ Identifiers	

Figure 54: Button for Pinning a Properties View to the Selected Block

Until you un-click the toggle, Composer will always show the properties of the currently selected object even if a different object is selected.

3. Create a new `Properties` view. Drag out the tab to another row. They will now show up side-by-side.
4. Select the second block. The second properties view will show properties of the second block.

End of procedure

Saving Workflow Diagrams as Templates

You can save a diagram as a template and have it appear on the list of available templates when creating a new workflow diagram. Diagrams saved as templates can exported to/imported from the file system or to other Composer installations.

Procedure:
Saving a workflow diagram as a template

Start of procedure

To save a diagram as a template:

1. In the Project Explorer, right-click the diagram in the `Workflows` folder.
2. Select `Save 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.`
5. Click `OK` to close the dialog box.

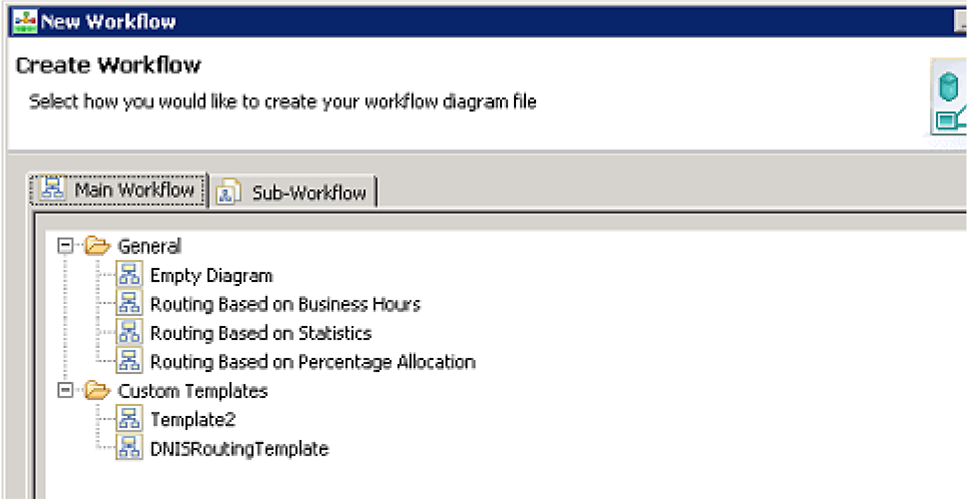
End of procedure

Procedure:
Accessing Saved Templates

Start of procedure

To view a diagram previously saved as template:

1. From the `File` menu, select `New` >`Workflow Diagram`. The New Workflow dialog box opens. The template appears in the `Main Workflow` tab under `Custom Templates` (see Figure 55).



New Workflow

Create Workflow

Select how you would like to create your workflow diagram file

- General
  - Empty Diagram
  - Routing Based on Business Hours
  - Routing Based on Statistics
  - Routing Based on Percentage Allocation
- Custom Templates
  - Template2
  - DNISRoutingTemplate

Figure 55: New Workflow Dialog Box

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` folder under the current Project.

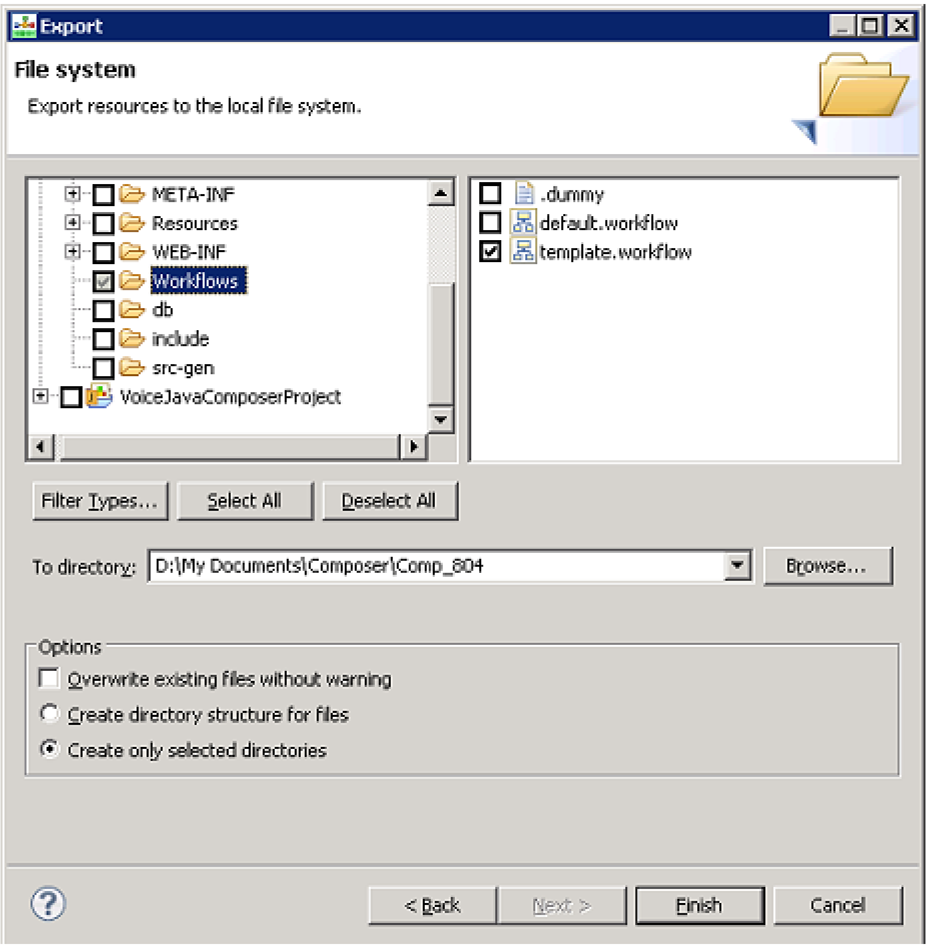
End of procedure

Procedure:
Exporting a Diagram Template to the File System

To export a diagram to the file system or another user’s Composer:

Start of procedure

1. In the Project Explorer, right-click the diagram in the `Workflows` folder.
2. Select `Export`. The `Export` dialog box opens.
3. Under `General`, select `File System` and click `Next`. The `File System` dialog box opens (see Figure 56).


Export

File system

Export resources to the local file system.

- □ META-INF
- □ Resources
- □ WEB-INF
- ☑ Workflows
- □ db
- □ include
- □ src-gen
- □ VoiceJavaComposerProject

- □ .dummy
- □ default.workflow
- ☑ template.workflow

To directory: D:\My Documents\Composer\Comp_804

Options

- □ Overwrite existing files without warning
- ○ Create directory structure for files
- ● Create only selected directories

Figure 56: File System Dialog Box for Export

4. In the `File System` dialog box, select the folder containing the diagram(s).
5. On the right, click check boxes to indicate the diagram(s) to export.
6. Opposite `To directory`, select the Composer installation to export to or browse for the destination directory.
7. At the bottom of the dialog box, select one of the following:
   - `Overwrite existing files without warning`
   - `Create directory structure for files`
   - `Create only selected directories (default).`
8. Click `Finish`.

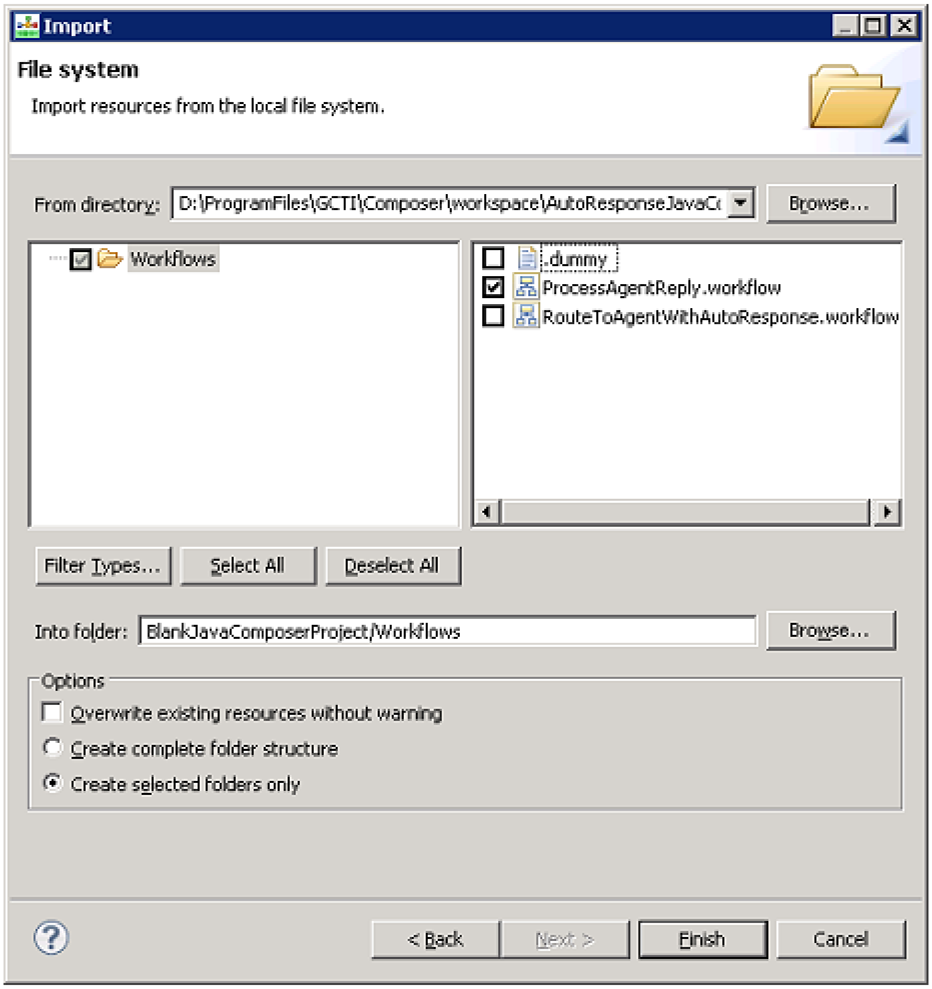
End of procedure

Procedure: Importing a Diagram Saved as a Template

Start of procedure

To import a diagram previously saved as a template:

1. In the Project Explorer, right-click the diagram in the `Workflows` 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. Figure 57 shows an example.



Import

File system

Import resources from the local file system.

From directory: D:\ProgramFiles\GCTI\Composer\workspace\AutoResponseJavaC

☑ Workflows

□ .dummy
☑ ProcessAgentReply.workflow
□ RouteToAgentWithAutoResponse.workflow

Into folder: BlankJavaComposerProject/Workflows

Options
□ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Figure 57: File System Dialog Box for 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`.

End of procedure

Appendix

A Composer Blocks

This appendix introduces you to Composer's IPD and diagram-building blocks. It also lists the Interaction Routing Designer block equivalent for each Composer block, where one exists. See the *Composer 8.0 Help* for the properties associated with each Composer block.

This appendix contains the following sections:

IPD Blocks

As described in "Interaction Process Diagrams" on page 14, an interaction process diagram is comprised of various IPD-building blocks. Table 6 summarizes Composer's IPD blocks.

Table 6: Composer interaction process diagram Blocks

IPD Block	Description
Interaction Queue	Use this block to define a multimedia (non-voice) interaction queue in an interaction process diagram and to create views, which define conditions for pulling interactions out of queues for submittal to workflows. See "Defining an Interaction Queue" on page 71 for more information.
Media Server	Represents an existing media server, such as an e-mail server. Use to direct interactions from a media server into an interaction process diagram. See "Adding a Media Server Block" on page 68 for more information.

Table 6: Composer interaction process diagram Blocks (Continued)

IPD Block	Description
Workflow	Points to an existing Workflow resource (either a workflow diagram or SCXML file) to which an interaction can be sent for specialized processing. See “Adding a Workflow Block” on page 79 for more information.
Workbin	Use to define a temporary storage area for interactions accessible from the agent desktop.

Workflow Diagram-Building Blocks

A workflow diagram is comprised of various blocks, which can be thought of as “diagram-building” blocks. Composer’s Palette tab (see Figure 52 on page 94) groups these blocks into different categories. This section summarizes the blocks in each category.

Flow Control Blocks

Table 7 summarizes the workflow diagram-building blocks used for flow control.

Table 7: Flow Control Blocks

Block Name	Description
Assign	Use to assign a computed value/expression or a literal value to a variable.
Branching	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.
Disconnect	Use to disconnect the caller and end the call. The Disconnect block invokes the Cancel Call treatment, which ends the workflow and deletes the interaction from URS memory.
ECMAScript	Use to build an ECMAScript expression for routing decisions. Universal Routing Server 8.0 supports SCXML plus ECMAScript as a routing language. While the core SCXML provides State Chart functionality, you can specify URS-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.
Entry	All workflow diagrams must start with an Entry block. Defines variables that can be shared across different blocks in the same workflow.

Table 7: Flow Control Blocks (Continued)

Block Name	Description
Exit	Use to terminate the workflow or to return control back to the calling workflow in case of a sub-workflow (subroutine).
Log	Use to record information about the application; for example, caller-recorded input that is collected while the application is running, or error messages.
Looping	Use this block to iterate over a sequence of blocks multiple times in the following scenarios: • Iterate over a sequence of blocks based on a self-incrementing counter (FOR). • Iterate indefinitely until an exit condition is met (WHILE). • Iterate over records/data returned by the DB Data block (CURSOR/FOREACH). Also, populate variables if variables mapping is defined as described in the *Composer 8.0 Help*. • Iterate over data returned by Context Services blocks (FOREACH). Also, populate variables if variables mapping is defined. • Iterate over a JSON array defined in the application.
Subroutine	Use to invoke external SCXML documents or a sub-workflow created using Composer.

Routing Blocks

Table 8 summarizes the workflow diagram-building blocks used for routing.

Table 8: Routing Blocks

Block Name	Description
Default Route	Instructs URS to route a voice interaction to the default destination.
Force Route	Forces Universal Routing Server to route the interaction to the first target type without any other operations.
Queue Interaction	Places a non-voice interaction into an existing queue.
Route Interaction	Routes a non-voice interaction to one or more target objects.
Routing Rule	Select routing rules that currently exist in the Configuration Database, such as those created with Interaction Routing Designer: You can select load balancing, percentage, or statistics routing rules.

Table 8: Routing Blocks (Continued)

Block Name	Description
Stop Interaction	Requests Interaction Server to stop processing an interaction and allows assignment of a reason code.
Target	Routes a voice interaction to a target. Can be used for percentage and/or conditional routing using threshold expressions, such as those used for share agent by service level agreement routing.

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. Table 9 summarizes the workflow diagram blocks for voice treatments.

Table 9: Voice Treatment Blocks

Block Name	Description
Create User Announcement	Use to record a caller announcement.
Cancel Call	Use to stop a currently running dialog.
Delete User Announcement	Use to 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.
Set Default Route	Use to set/change the default destination.
User Input	Use to play a text-to-speech announcement, to play an announcement and collect digits, and for collecting digits.

eServices (Multimedia) Blocks

The eServices blocks perform specialized processing of multimedia interactions. Table 10 summarizes these blocks.

Table 10: eServices Blocks

Block Name	Description
Create E-mail	Use to create an e-mail message to be sent out to a customer or to another agent and to specify the interaction queue where the outbound e-mail should be placed.
Create SMS	Use 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 cell phones and other handheld devices.
Send E-mail	Use to send an e-mail waiting in a queue that was previously created using the Create E-mail block.
Send SMS	Use this block to send an Short Message Service (SMS) message created with the Create SMS block.

Server-Side Blocks

Server-Side blocks provide the ability to interact with internal and external custom server-side pages, Web Services, and URLs. Table 11 summarizes Server-Side blocks.

Table 11: Server-Side Blocks

Block Name	Description
Backend	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.
DB Data	Use for connecting to a database and retrieving/manipulating information from/in a database.
External Service	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.
Web Request	Use to invoke any supported HTTP web request or REST-style web Service.
Web Service	Use to invoke Web Services.

Context Services Blocks

Table 12 summarizes the workflow diagram-building blocks used for Context Services as described on page 53.

Table 12: Context Services Blocks

Block Name	Description
Associate Service	Use the Associate Service block to associate an anonymous service record with a customer whose profile exists in the database used for Context Services.
Complete Service	Use to mark an active service as completed in the Universal Contact Server Database used for Context Services.
Complete State	Use to mark the completion of a specified state in the context of a service in the database used for Context Services.
Complete Task	Use to mark the application as completing a specified task within a service/state.
Create Customer	Use to create a voice callflow/routing workflow that includes the capability to create a customer profile through Context Services.
Enter State	Use the Enter State block to mark the entry of the application into a specified state in the context of a service.
Identify Customer	Use to identify a customer in the database based on search criteria, which can be customer profile core data or customer profile extension data.
Query Customer	Use to look up a customer's core profile and profile extension attributes.
Query Services	Use to query the database for a list of services associated with a particular Customer ID or, in case of unassociated services, the Contact Key.
Query States	Use to query the database used for Context Services for active and completed states data for a specified service.
Query Tasks	Use to query the database used for Context Services for active and completed tasks within a state for a specified service.
Start Service	Use to communicate the creation or start of a service in the UCS Database.
Start Task	Use to mark the application as entering a specified task within a service/state.
Update Customer	Use to update the customer profile in the database used for Context Services.

Composer Blocks Mapped to IRD Objects

The tables below list IRD objects based on their IRD toolbar category name and point to the corresponding Composer diagram building block.

Note: Composer refers to the fundamental element of a workflow diagram as a *block* whereas in IRD documentation, this element is referred to as an *object*.

Table 13: IRD Data & Services Category

IRD Object Name	Composer Block Name
Database Wizard object	DB Data block
Web Service object	Web Service block
External Service object	External Service block

Table 14: IRD Miscellaneous Category

IRD Object Name	Composer Block Name
Assign object	Assign block
Call Subroutine object	Subroutine block
Entry object	Entry block
Exit object	Exit block
Function object	ECMAScript block
If object	Assign, Branching, ECMAScript, Log, Entry, and Looping blocks all open Expression Builder

Table 15: IRD Routing Category

IRD Object Name	Composer Block Name
Selection object	Target block
Percentage object	Target block
Default object	Default Route block

Table 15: IRD Routing Category (Continued)

IRD Object Name	Composer Block Name
Force Routing	Force Routing block
Percentage, Load Balancing, Statistics Routing Rule objects	Routing Rule block
Statistics object	Target block

Table 16: IRD Segmentation Category

IRD Object Name	Composer Block Name
ANI object	Branching. Expression Builder contains a DNIS variable under Workflow Variables > System. Also see ANI predefined application variable in Figure 14 on page 32.
DNIS object	Branching. Expression Builder contains a DNIS variable under Workflow Variables > System. Also see ANI predefined application variable in Figure 14 on page 32
Date object	Branching. You can also use the _genesys.IsSpecialday Functional Module as described in Appendix B, “Functional Modules,” on page 113.
Day of Week object	Branching. You can also use the _genesys.session.day Functional Module.
Time object	Branching. You can also use the _genesys.timeinZone Functional Module.
Generic object	Branching. See “Expression Builder” on page 32.

Table 17: IRD Treatment Category

IRD Treatment	Composer Equivalent	Composer Busy Treatment?
Record user announcement	Create User Announcement	no
Delete user announcement	Delete User Announcement	no
Cancel call	Disconnect	no
IVR	IVR	yes
Pause	Pause	yes

Table 17: IRD Treatment Category (Continued)

IRD Treatment	Composer Equivalent	Composer Busy Treatment?
Play announcement	Play Message	yes
Text to speech	Play Message	yes
Play announcement and collect digits	User Input	yes
Text to speech and collect digits	User Input	yes
Verify digits	Digit verification in User Input	yes
Collect digits	User Input	yes
Play Application	Play Application	Play Application
Busy	Play Sound	yes
Fast busy	Play Sound	yes
Music	Play Sound	yes
Ringback	Play Sound	yes
Silence	Play Sound	yes
RAN	Play Sound	yes
Set default destination	Set Default Route	no

Table 18: IRD Multimedia Category

Composer Route Block Name	Equivalent IRD Block/Object
Create E-mail object	Create E-mail block
Send E-mail object	Send E-mail block
Create SMS object	Create SMS block
Send SMS object	Send SMS block
Queue Interaction object	Queue Interaction block
Route Interaction object	Route Interaction block

Table 18: IRD Multimedia Category (Continued)

Composer Route Block Name	Equivalent IRD Block/Object
Stop Interaction object	Stop Interaction block
External Service (ESP) object	External Service block

Table 19: SMS Category

IRD Object Name	Composer Block Name
Create SMS Out	Create SMS block
Send SMS Out	Send SMS block

Other Functionality

Table 20 lists other IRD functionality that can also be found in Composer.

Table 20: Statistics

IRD Functionality	Composer Functionality
List Objects	List Object Manager. See Composer's List Object Manager (see page 37).
Statistics Dialog Box	Statistics Manager and Builder. See Composer’s Statistics Builder (see page 36).
Generic Segmentation object	Expression Builder (see page 32)
Generic Segmentation object	Skill Expression Builder (see page 38)

Appendix

B Functional Modules

To introduce you to the Genesys Functional Modules found under URS Functions in Expression Builder (see Figure 15 on page 33), this appendix presents some general information with emphasis on the _genesys category (see Figure 59 on page 114).

Note: For more detailed information, consult the *Genesys 8.0 SCXML Technical Reference*. Select Help > Contents within Composer.

This appendix contains the following sections:

Blocks Using the Functional Modules

The Functional Modules are accessed through Expression Builder, which opens from the following Flow Control blocks (see Table 7 on page 104):

- Assign—Assign Data property
- Branching—Conditions property
- ECMAScript —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 the URS functions for SCXML-based strategies.

URS Functions

When `URS Functions` is selected in Expression Builder (see page 32), there are various data subcategories, which can be expanded (see Figure 58).

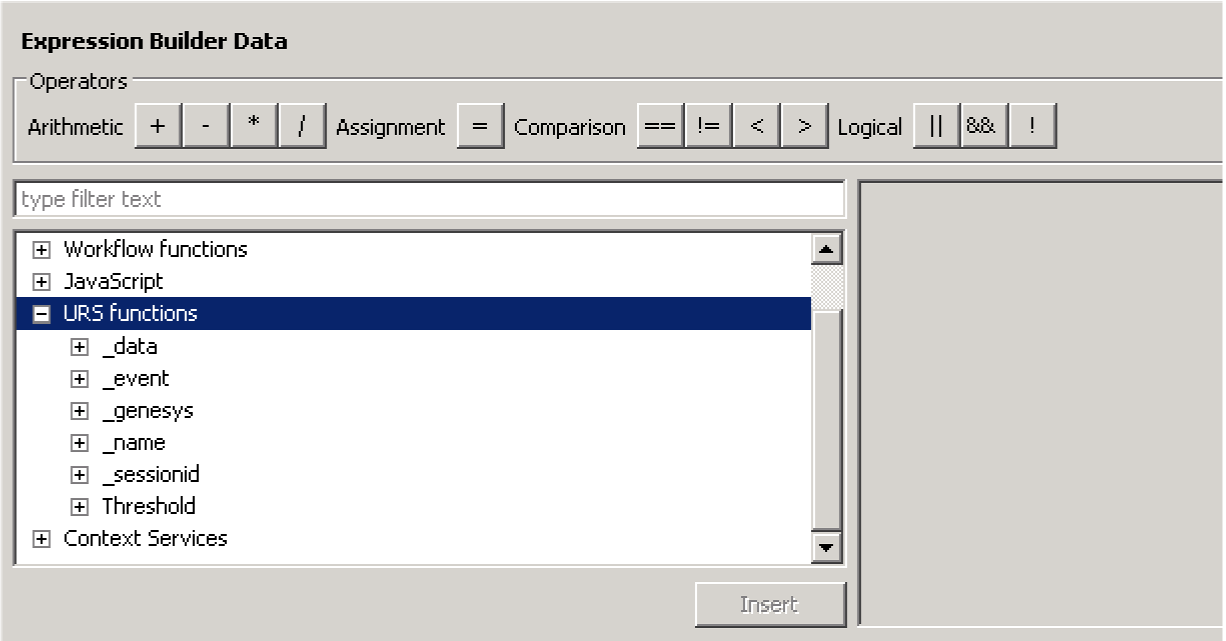
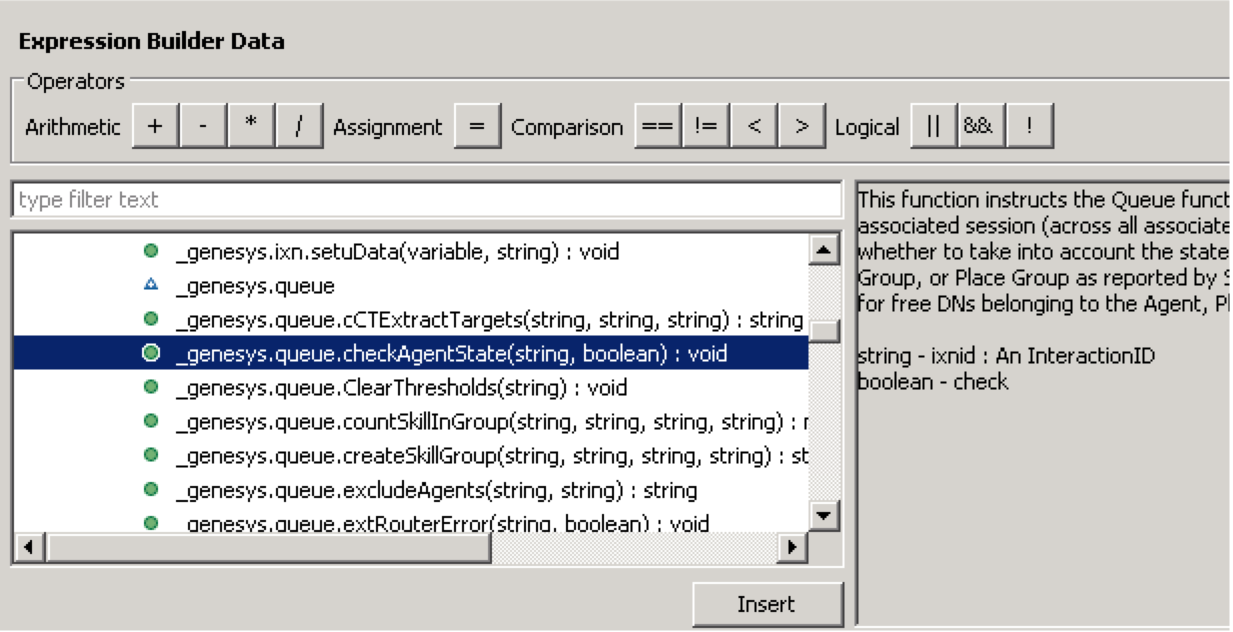


Figure 58: URS Functions in Expression Builder

Expand one of these categories and URS functions appear for selection. For example, expand `_genesys.queue` (see Figure 59).


Expression Builder Data
Operators
Arithmetic
+
-
*
/
Assignment
=
Comparison
==
!=
<
>
Logical
||
&&
!
type filter text
_genesys.ixn.setuData(variable, string) : void
_genesys.queue
_genesys.queue.cCTExtractTargets(string, string, string) : string
_genesys.queue.checkAgentState(string, boolean) : void
_genesys.queue.ClearThresholds(string) : void
_genesys.queue.countSkillInGroup(string, string, string, string) : r
_genesys.queue.createSkillGroup(string, string, string, string) : st
_genesys.queue.excludeAgents(string, string) : string
_genesys.queue.extRouterError(string, boolean) : void
Insert

This function instructs the Queue funct
associated session (across all associate
whether to take into account the state
Group, or Place Group as reported by S
for free DNs belonging to the Agent, Pl

string - ixnid : An InteractionID
boolean - check

Figure 59: URS Queue Functions in Expression Builder

Find online documentation in Composer on the Genesys-supplied Functional Modules by selecting `Help > Contents > Universal Routing 8.0 SCXML API Reference`.

Categories Under URS Functions

Table 21 below summarizes each of the categories under `URS Functions` in Expression Builder as shown in Figure 58 on page 114.

Table 21: Expression Builder Data Subcategories

Expression Builder Category	Description
_genesys	Use for access to Genesys platform-related objects, properties, and functions. Every SCXML session instance running in the Genesys Universal Routing platform has a global root object from which an application has access to the all Genesys platform-related objects, properties, and functions. This object is maintained by the URS platform. The name of the object is "_genesys". For more information on the _genesys object module, see Table 22 on page 117.
_data	Used to access data for objects that are created based on the "orchestration" logic (<datamodel> tag). Currently these are as follows: • _data._dest (the Route Point address from which a session was started from) • _data._orig (the address that originated the interaction and the request)
_event	Use for access to event properties.Genesys fully supports the SCXML Event Model, both internal session events and external events. If the event has properties, the `_event` system variable contains the event's properties in the "_event.data" location.
_name	The variable "name" is bound at load time to the name of the SCXML state machine, which is specified in the "name" attribute of the <scxml> element.
_sessionid	The variable "sessionid" is bound at load time to the system-generated id for the current SCXML session. It remains bound until the session terminates.

_genesys Data Subcategory

In Table 21, the _genesys Data Subcategory is where you will find modules that access Genesys-platform related objects, properties, and functions. When _genesys is selected, the third column in Expression Builder displays a long list of _genesys-prefixed entries (see Figure 15 on page 33). Following the _genesys. prefix will be one of the following: dialog, ixn, queue, resource, server, session, stat, or voice.

For example:

- _genesys.dialog
- _genesys.ixn.GetMediaIntValue('media')
- _genesys.queue.ClearThresholds('ixnid')
- _genesys.resouce.deviceType.CFGCP'
- _genesys.server
- _genesys.session.activeServerName('name')

The next section summarizes these properties.

_genesys Object Properties

Table 22 summarizes the dialog, ixn, queue, resource, session, server, stat, and voice properties for the _genesys object. associated with each object, which you may want to use in your routing application.

,

Table 22: Expression Builder _genesys Data Subcategory

_genesys Property	Description
_genesys.dialog	Use to get information about dialog (treatment)-related properties for an SCXML session. Here you will find parameter elements for the following action elements to be used as input attributes. • <collect> • <play> • <playandcollect> • <playandverify> • <runtreatments>
_genesys.ixn	Use to get information about interaction-related properties for an SCXML session. An interaction represents the various types of communications between a resource and a customer: • Conversation-based communication between a customer and given contact center or enterprise resources using a single logic media (certain media may support more than one media type over an interaction (voice and video) (for example, a voice call or a chat session). This type of interaction involves a series of information being communicated back and forth. • Message-based communication with a customer (for example, an incoming e-mail or sms or an outgoing e-mail or sms). • Web-based communication with a customer (for example, web page activity). The following types of properties are maintained with an interaction: • Interaction state information (state, reason for the state, and so on). • Customer ID—It will be the link to the associated customer view information. • Application attached data. • Media-specific customer identification data. • Media-specific characteristic data (media type, bandwidth, current dialog application being executed, and media resources being used (for example, TTS engine)). • Resources and media devices involved in the interaction.

Table 22: Expression Builder _genesys Data Subcategory (Continued)

_genesys Property	Description
_genesys.queue	Use for logic to request the appropriate resource for some processing and return the appropriate address information for the resource. The current URS functionality (target selection, queuing, prioritization, and so on) is associated with this module. Here you will find an approximate SCXML version of many Universal Routing Server functions that fall under the “Miscellaneous” and “Target Manipulation” categories in the *Universal Routing 8.0 Reference Manual,* such as _genesys.FMname.selectTargets and _genesys.FMname.getSkillInGroup.
_genesys.resource	Use to get information about resource-related properties for an SCXML session. A resource is an entity in the business which is involved in helping customers with the services they need. This resource can either be directly involved with the customer through an actual interaction (an IVR, website, knowledge management system, and so on) or be indirectly involved with the customer by doing the processing on behalf of the customer (for example, an agent). A resource can be either a human (for example, an agent) or a device (for example, an IVR).
_genesys.server	Use to get information about the Genesys server on which an SCXML session is running.Every SCXML session instance running in the Universal Routing platform has a global root object from which an application (such as a routing application created in Composer) has access to the platform server information. This object is maintained by the UR platform. The name of the object is “_genesys.server”. Note; _genesys.session below accesses the _genesys.session.ActiveServerName(‘name’) module.
_genesys.session	Use to get information about an SCXML session. Every SCXML session instance running in the Universal Routing/Orchestration platform will have an object with a set of common orchestration logic properties. These properties are maintained by the orchestration platform, but they can be set or updated by the orchestration logic itself. They are also used by the orchestration platform for orchestration logic reporting and management functionality. The name of the object is “_genesys.session”. Here you will find an approximate SCXML version of certain Universal Routing Server functions that fall under the “Miscellaneous,” “Date and Time,” and “Configuration Options” categories in the *Universal Routing 8.0 Reference Manual.*
_genesys.stat	Use to get statistical information about an SCXML session. Here you will find an approximate SCXML version of certain Universal Routing Server functions that fall under the “Statistical” category in the *Universal Routing 8.0 Reference Manual.*
_genesys.voice	Use to get information about voice-interaction related properties for an SCXML session.

More information is available in the `SCXML Expressions` > `Object Module` section of the *Genesys 8.0 SCXML Technical Reference* online help by selecting `Help` > `Contents` within Composer. Consult that help system for information on the properties. Also see:

- `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`

IRD Functions

The Functions chapter in the *Universal Routing 8.0 Reference Manual* groups functions available through the IRD Function object into the following categories:

- Specialized Functions
- CallInfo Functions
- Configuration Options Functions
- Data Manipulation Functions
- Date and Time Functions
- Force Functions
- List Manipulation Functions
- Miscellaneous Functions
- Reporting Functions
- Statistical Functions
- String Manipulation Functions
- Target Manipulation Functions

In some cases, Genesys provides a Functional Module through Expression Builder with a similar name to an IRD function (see Table 23 on page 120). For example, the Functional Modules `_genesys.FMname.getSkillInGroup` is similar to the getSkillInGroup IRD function.

Functional Modules and IRD Functions

The first column in Table 23 lists each URS function available for selection in the IRD Function object. Columns two and three indicate its SCXML and/or Functional Module equivalent, and support status.

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
ACDQ	interaction.voice.acdq property	supported
acfgdata	See Threshold under URS Functions in Expression Builder. See Figure 60 on page 136.	supported
ActiveServerName	_genesys.session.activeServerName	supported
AddAgentState		not supported
AddStackFrame		not supported
Alarm	Use <log> with the following attributes set: • label = Alarm Number • expr = Alarm Message • level = 5 for alarm	supported
ANI	interaction.voice.ani property	supported
AnswerCall		not supported
ApplyBusinessRule		not supported
ApplyIVRTreatment		not supported
Attach	_genesys.ixn.setuData()	supported
BlockDN	_genesys.queue.reserveTarget()	supported
BusinessData	interacdtion.udata property	supported
BusinessDataINT	interaction.udata property	supported
BusinessSkill		not supported
callage	See Threshold under URS Functions in Expression Builder. See Figure 60 on page 136.	supported
CallID	interaction.voice.callid property	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
CallsDistributed	_genesys.stat.sData() parameters: • object = either VQ.name, R, RP • statistic = "CallsDistributed"	supported
CallsEntered	_genesys.stat.sData() parameters: • object = either VQ.name, R, RP • statistic = "CallsEntered"	supported
CallStrategy		not supported
CallsWaiting	_genesys.stat.sData() parameters: • object = any target related object • statistic = "CallsWaiting"	supported
CallType	interaction.FMObjectname.type property	supported
Cat	Part of ECMAScript string functions – either "+" or String.concat()	supported
CCTExtractTargets	_genesys.queue.cCTExtractTargets()	supported
CED	interaction.voice.ced property	supported
Char		not supported
CheckAgentState	_genesys.queue.checkAgentState()	supported
ClearTargets	<queue:cancel>	supported
ClearTargetThresholds		not supported
ClearThresholds	_genesys.queue.clearThresholds()	supported
ConnID	interaction.voice.connid property	supported
CountAgentsInTargetList		not supported
CountSkillInGroup	_genesys.queue.countSkillInGroup()	supported
CreateSkillGroup	_genesys.queue.createSkillGroup()	supported
Database_object	<session:fetch> with application server	supported
Date	Standard ECMAScript Date object	supported
DateInZone	_genesys.session.dateInZone()	supported
Day	Standard ECMAScript time and date objects	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
DayInZone	Developer would use xdate = _genesys.session.dateInZone() and then xday = xdate.getDay()	supported
Debug		not supported
Default	<queue:default>	supported
Delay	<state> <send with a delay>	supported
DeleteAttachedData	_genesys.ixn.deleteuData()	supported
DeleteKey		not supported
DeliverCall		not supported
DeliverToIVR		not supported
DelStackFrame		not supported
Dest	_data._dest property	supported
DistributedPercentage	_genesys.stat.sData() parameters: • object = either VQ.*name, RP* • statistic = "DistributedPercentage"	supported
DistributedWaitingTime	_genesys.stat.sData() parameters: • object = either VQ.name, RP • statistic = "DistributedWaitingTime"	supported
DNIS	interaction.voice.dnis property – or interaction.contactedaddr property	supported
Email_object		not supported
Error		not supported
Even		not supported
EvenEx		not supported
ExcludeAgents	_genesys.queue.excludeAgents()	supported
ExcludeLocations		not supported
Exit	<final> element	supported
ExpandGroup		not supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
ExpandWFActivity		not supported
ExtrouterError	_genesys.queue.extRouterError()	supported
ExtrouterStatus	_genesys.queue.extRouteStatus()	supported
FindServiceObjective	_genesys.queue.findServiceObjecctive() - will deprecate over time – push to application server.	supported
Force		not supported
GetAvgStatData	_genesys.stat.getAvgData() – List of target and statistic – returns the average.	supported
GetConfigOption	_genesys.session.getConfigOption()	supported
GetCurrentSwitch	interaction.location.media_server property	supported
GetCurrentTServer	interaction.location.control_server property	supported
GetCustomerSegment	interaction udata property (“CustomerSegment”)	
GetDataBaseInfo	<session:fetch> and application server	supported
GetIntegerKey	ECMAScript string manipulation code replaces it	supported
GetLastError	This information is in the error.xxxx event – The developer must write code to save.	supported
GetLastErrorInfo	This information is in the error.xxxx event – The developer must write code to save.	supported
GetMaxStatData	_genesys.stat.getMaxData() – List of targets and statistic s– returns the maximum.	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?

GetMaxSubList	ECMAScript string manipulation code replaces it
	function maxSort(a, b)
	{
	  var aA;
	  var aB;
	  aA = a.split(“:”);
	  aB = b.split(“:”);
	   if(aA[1] > aB[1])
	     return -1
	   if(aA[1] < aB[1])
	     return 1
	   return 0
	}
	function getMaxSubList(iStr)
	    var sArray;
	    var sortA;
	    var retS = null;
	    sArray = iStr.split(“|”);
	    sortA = sArray.sort(maxSort);
	    tA = sortA[0].split(“:”);
	    for (i=0; i i<sortA.length; i++) {
	        tt = sortA[i].split(“:”);
	        if (tA[1] == tt[1]) {
	            retS = retS + sortA[i];
	        } else {
	            return retS;
	        }
	    }
	}
		supported

GetMediaType	interaction.getMediaIntValue(_genesys.ixn.interactions[x].MediaFM.media)	supported
GetMediaTypeFromName		not supported
GetMediaTypeName	interaction..MediaFM.media property	supported
GetMinStatData	_genesys.stat.getMinData()– List of targets and statistics – returns the minimum.	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

<table>
<tr><th>URS/IRD Function</th><th>SCXML and Functional Module Equivalent</th><th>Supported?</th></tr>
<tr><td>GetMinSubList</td><td>ECMAScript string manipulation code replaces it
function minSort(a, b)
{
  var aA;
  var aB;
  aA = a.split(“:”);
  aB = b.split(“:”);
   if(aA[1] < aB[1])
      return -1
   if(aA[1] > aB[1])
      return 1
   return 0
}
function getMinSubList(iStr)
    var sArray;
    var sortA;
    var retS = null;
    sArray = iStr.split(“|”);
    sortA = sArray.sort(maxSort);
    tA = sortA[0].split(“:”);
    for (i=0; i i<sortA.length; i++) {
         tt = sortA[i].split(“:”);
         if (tA[1] == tt[1]) {
              retS = retS + sortA[i];
         } else {
              return retS;
         }
    }
}</td><td>supported</td></tr>
<tr><td>GetObjectProperty</td><td></td><td>not support</td></tr>
<tr><td>GetPriority</td><td></td><td>not supported</td></tr>
<tr><td>GetQueueID</td><td></td><td>not supported</td></tr>
<tr><td>GetRemoteAccessCode</td><td></td><td>not supported</td></tr>
<tr><td>GetRoutingPoint</td><td>_data._dest property</td><td>supported</td></tr>
</table>

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
GetServiceObjective	interaction.udata property (“ServiceObjective”)	supported
GetServiceType	interaction.udata property (“ServiceType”)	supported
GetSkillInGroup		not supported
GetStringKey	ECMAScript string manipulation code replaces it	supported
GetTenantID		not supported
GetWebServiceInfo	<session:fetch> and application server	supported
Increment		not supported
IncrementPriority	_genesys.queue.incrementPriority()	supported
InteractionData	interaction.udata property	supported
InteractionDataINT	interaction.udata property	supported
InVQWaitTime	_genesys.stat.sData() parameters: • object = VQ.name • statistic = “InVQWaitTime”	supported
IsSituation		not supported
IsSpecialDay	_genesys.session.isSpecialDay()	supported
IsSpecialDayEx	_genesys.session.isSpecialDay()	supported
JumpToStrategy		not supported
KeepQueue		not supported
KVListGetArray	<session:fetch> and using the returned ECMAScript objects (JSON) Internal only for GetWebServiceInfo	supported
lcfgdata	part of the threshold definition of <queue:target> (official) for skill expression – for expression language internally.	supported
ListGetDataCfg	_genesys.session.getListItemValue()	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?

ListGetInteger	supported through – ECMAScript functionality
	function ListGetInteger(ind, iStr)
	  sArray = iStr.split("|");
	  if (ind < sArray.length) {
	    sItem = sArray[ind];
	    aItem = sItem.split(":");
	    if (aItem[1].isNaN()) {
	      return 0;
	    } else {
	      return aItem[1];
	    }
	  }
	 return 0;
	}
		supported

ListGetKey	supported through – ECMAScript functionality
	function ListGetKey(ind, iStr)
	  sArray = iStr.split("|");
	  if (ind < sArray.length) {
	    sItem = sArray[ind];
	    aItem = sItem.split(":");
	    return sItem[0];
	  }
	 return 0;
	}
		supported

ListGetSize	supported through – ECMAScript functionality
	function ListGetSize(ind, iStr)
	 sArray = iStr.split("|");
	 return sArray.length;
	}
		supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?

ListGetString	supported through – ECMAScript functionality
	function ListGetString(ind, iStr)
	sArray = iStr.split("|");
	if (ind < sArray.length) {
	sItem = sArray[ind];
	aItem = sItem.split(":");
	if (aItem[1].isNaN()) {
	return aItem[1];
	}
	}
	return 0;
	}
		supported

ListLookup	Covered by <session:fetch> and the application server	supported
ListLookupCfg	_genesys.session.listLookupValue()	supported
Monitor		not supported
MultiplyTargets		not supported
MultiSkill		not supported
NMTExtractTargets	_genesys.queue.nMTExtractTargets()	supported
NotDistributedPercentage	_genesys.stat.sData() parameters: • object = VQ.name, RP • statistic = "NotDistributedPercentage"	supported
NotDistributedWaitingTime	_genesys.stat.sData() parameters: • object = VQ.name, RP • statistic = "NotDistributedWaitingTime"	supported
NPA	interaction.ixn.nPA()	supported
NPANXX	interaction.ixn.nPANXX()	supported
OldSkill		not supported
OnCallAbandoned	The interaction.abandoned can be used to all the developer to create there own logic for processing this condition. So this function is not needed any more.	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
OnRouteError	_genesys.queue.onRouteError()	supported
Orig	_data._orig property	supported
PACCode	interaction.voice.ced property	supported
Peg	use <log> with the following attributes set: • label = Peg name • expr = increment • level = 6 for peg	supported
PegValue	This is the same as SData so we will use _genesys.stat.sData()	supported
Pop		not supported
PositionInQueue	_genesys.stat.sData()	supported
Print	use <log> with the following attributes set: • label = print • expr = message • level = 5 for debug	supported
Priority	covered by <queue:submit>	supported
PriorityTuning	_genesys.queue.priorityTuning()	supported
Push		not supported
PutToPersistentQueue		not supported
PutToWorkbin		not supported
Rand	Part of ECMAScript (math.random())	supported
ReleaseCall		not supported
RequestedSkillExpression		not supported
RequestService		not supported
ResetBusyTreatments		not supported
ResetStatAdjustment	_genesys.queue.resetAdjustment()	supported
Return		not supported
RouteCall		not supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
Routed	_genesys.queue.routed()	supported
Router	_genesys.platform.server.name and _genesys.platform.server.cluster properties	supported
Routing_object		not supported
SData	_genesys.stat.sData()	supported
SelectDN	<queue:submit>	supported
SelectTargets	_genesys.queue.selectTargets()	supported
SelectTargetsByThreshold	_genesys.queue.selectTargetsByThreshold	supported
SendEMailAcknowledgement		not supported
SendEMailAutoResponse		not supported
SendEvent		not supported
SendRequest		not supported
ServerStatus	_genesys.session.serverStatus()	supported
SetBusyTreatment	<queue:submit> and <dialog:runtreatments>	supported
SetCallOption	_genesys.session.setOptions()	supported
SetDelayedAttach		not supported
SetDNIS	interaction.ixn.setDNIS()	supported
SetDNISOverride	interaction.ixn.setDNIS()	supported
SetInPersistentQueues		not supported
SetIntegerKey		not supported
SetInteractionAge	_genesys.queue.setInteractionAge()	supported
SetLastError		not supported
SetMandatoryTreatment		not supported
SetOutPersistentQueues		not supported
SetRouteType		not supported
SetRunTimeMode		not supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
SetStatAdjustment	_genesys.queue.setAdjustment()	supported
SetStatistic		not supported
SetStatUpdate	Periodically attach statistic to interaction – Use <send delay> when the timer event is received <transition> the application can get the appropriate statistic (_genesys.stat.sData) and then put it into _genesys.ixn.interaction[].udata.	supported
SetStringKey		not supported
SetTargetThreshold		not supported
SetTenant	_genesys.session.setTenant()	supported
SetThreshold		not supported
SetThresholdEx	The developer can define a global varilable/property, which contains the global threshold definition that can be used on any <queue:submit> action and assign the <target> threshold attribute to this variable.	supported
SetTranslationOverride	_genesys.queue.translationOverride()	supported
SetTreatmentMode		not supported
SetVirtualQueue	Internal only – part of <queue:submit>	supported
SetVQPriority	<queue:update with priority attribute>	supported
SetVQThreshold		not supported
StartTreatmentCollectDigits	<dialog:collect>	supported
StartTreatmentPlayAnnouncement	<dialog:play>	supported
StartTreatmentPlayAnnouncementAndDigits	<dialog:playandcollect>	supported
StartTreatmentPlayApplication	<dialog:start>	supported
StartTreatmentRecordUserAnnouncement	<dialog:createann>	supported
StartTreatmentVerifyDigits	<dialog:playandverify>	supported
StrAsciiBreak	Part of ECMAScript String.indexOf()??	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
StrAsciiTok	Part of ECMAScript String.split() with some extra code to get what you want.	supported
StrChar	Part of ECMAScript String.indexOf()	supported
StrGetChar	Part of ECMAScript String.charAt()	supported
StrLen	Part of ECMAScript String.length	supported
StrReplace	Part of ECMAScript requires some code using string related methods.	supported
StrStr	Part of ECMAScript requires some code using string related methods.	supported
StrSub	Part of ECMAScript String.substring()	supported
StrTargets	Part of ECMAScript – Composer to had easy building of these target string.	supported
StrToLower	Part of ECMAScript String.toLowerCase()	supported
StrToUpper	Part of ECMAScript String.toUpperCase()	supported
SuspendForDN	<transition> for queue.submit.done event	supported
SuspendForEvent		not supported
SuspendForTreatmentEnd	<transition> for the dialog and treatment related events	supported
TargetComponentSelected	This data is returned in the queue.submit.done event – If the application wants to use it after the event, then it needs to be stored.	supported
TargetObjectSelected	This data is returned in the queue.submit.done event – If the application wants to use it after the event, then it needs to be stored.	supported
Targets	<queue:submit>	supported
TargetSelected	This data is returned in the queue.submit.done event – If the application wants to use it after the event, then it needs to be stored.	supported
TargetSelectionTuning	_genesys.queue.targetSelectionTuning()	supported
Time	Standard ECMAScript time and date objects	supported
TimeBehind		not supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
TimeDifference	Standard ECMAScript time and date objects – and code. xtimeDiff = date2.getTime() – date1.getTime()	supported
TimeInZone	Developer would use xtime = _genesys.session.dateInZone() and then xtime = xdate.getHours() + “ : ” + xdate.getMinutes()	supported
Timeout		not supported
TimeStamp	Not needed because it was primarily used for TimeDifference function and now we are using ECMAScript time and data objects for this.	supported
Translate		not supported
Treatment_object		not supported
TreatmentBusy	<dialog:playsound>	supported
TreatmentCancelCall	<dialog:stop>	supported
TreatmentCollectDigits	<dialog:collect>	supported
TreatmentDeleteUserAnnouncement	<dialog:deleteann>	supported
TreatmentIVR	<dialog:remote>	supported
TreatmentMusic	<dialog:playsound>	supported
TreatmentPlayAnnouncement	<dialog:play>	supported
TreatmentPlayAnnouncement AndDigits	<dialog:playandcollect>	supported
TreatmentPlayApplication	<dialog:start>	supported
TreatmentRecordUser Announcement	<dialog:createann>	supported
TreatmentRingBack	<dialog:playsound>	supported
TreatmentSetDefaultRoute	<dialog:setdialogdefaultdest>	supported
TreatmentSilence	<dialog:playsound>	supported
TreatmentTextToSpeech	<dialog:play> <prompts type=”tts”>	supported

Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
TreatmentTextToSpeechandDigits	<dialog:playandcollect> <prompts tyhpe=”tts”	supported
TreatmentVerifyDigits	<dialog:playandverify>	supported
TRoute		not supported
UData	interaction.udata property	supported
UDataINT	interaction.udata property	supported
Update	_genesys.ixn.setuData()	supported
UpdateB		not supported
UpdateBusinessData	_genesys.ixn.setuData()	supported
UpdateBusinessSkill		not supported
UpdateInteractionData	_genesys.ixn.setuData()	supported
UpdateS		not supported
UpdateScript	_genesys.ixn.setuData() (key name - MyScript) – The value of this key-value pair will be the configuration layer DB ID of the script object to be used. This ID will be retrieved by Composer and set in the session logic to set the pair in the interaction.udata property using the genesys.ixn.setuData() function.	supported
UpdateSkill		not supported
UseAgentState	_genesys.queue.useAgentState()	supported
UseAgentStatistics	_genesys.queue.useAgentStatistics()	supported
UseCapacity	_genesys.queue.useCapacity()	supported
UseCustomAgentType	_genesys.queue.useCustomType()	supported
UseCustomDNType	_genesys.queue.useCustomType()	supported
UseCustomPlaceType	_genesys.queue.useCustomType()	supported
UseDNType	_genesys.queue.useDNType()	supported
UseMediaType	_genesys.queue.useMediaType()	supported
UseQuota		not supported

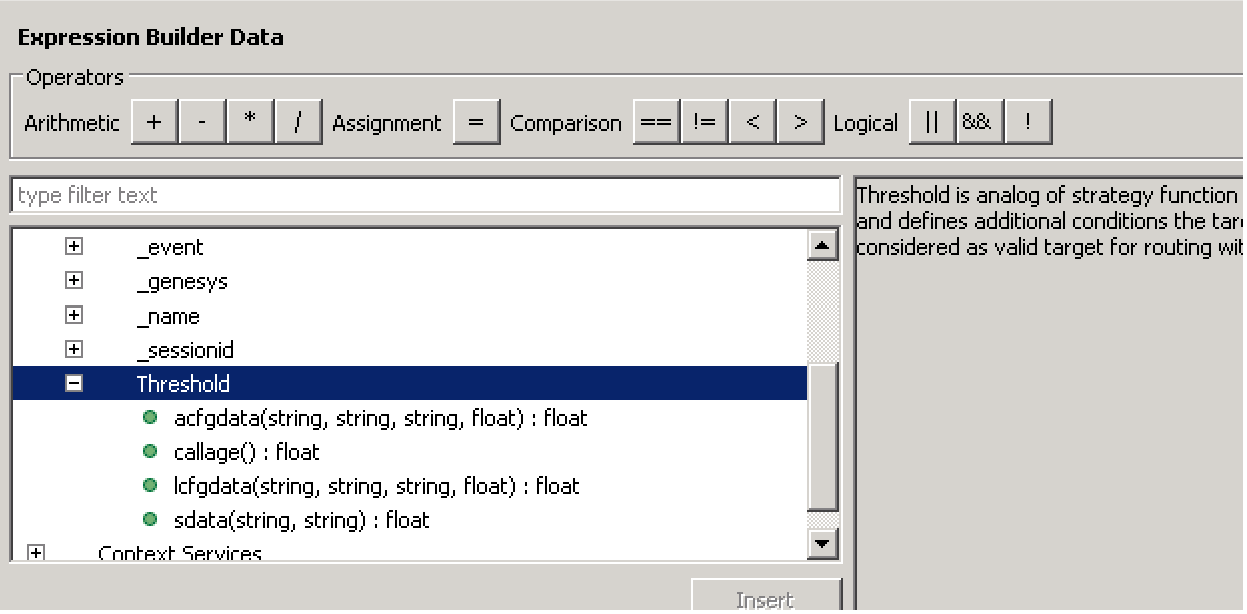
Table 23: URS/IRD Functions and SCXML Functional Module Equivalent (Continued)

URS/IRD Function	SCXML and Functional Module Equivalent	Supported?
VQSelected	This data is returned in the queue.submit.done event – If the application wants to use it after the event then it needs to be stored.	supported
Wait	<queue:submit>	supported
WakeCallUp		not supported
WakeUpData		not supported

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 via Composer’s Target block, 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. Figure 60 shows the threshold functions in Expression Builder.



Threshold is analog of strategy function
and defines additional conditions the tar
considered as valid target for routing wit

Figure 60: Threshold Functions

For complete information on these functions, see the *Universal Routing 8.0 Routing Application Configuration Guide*.

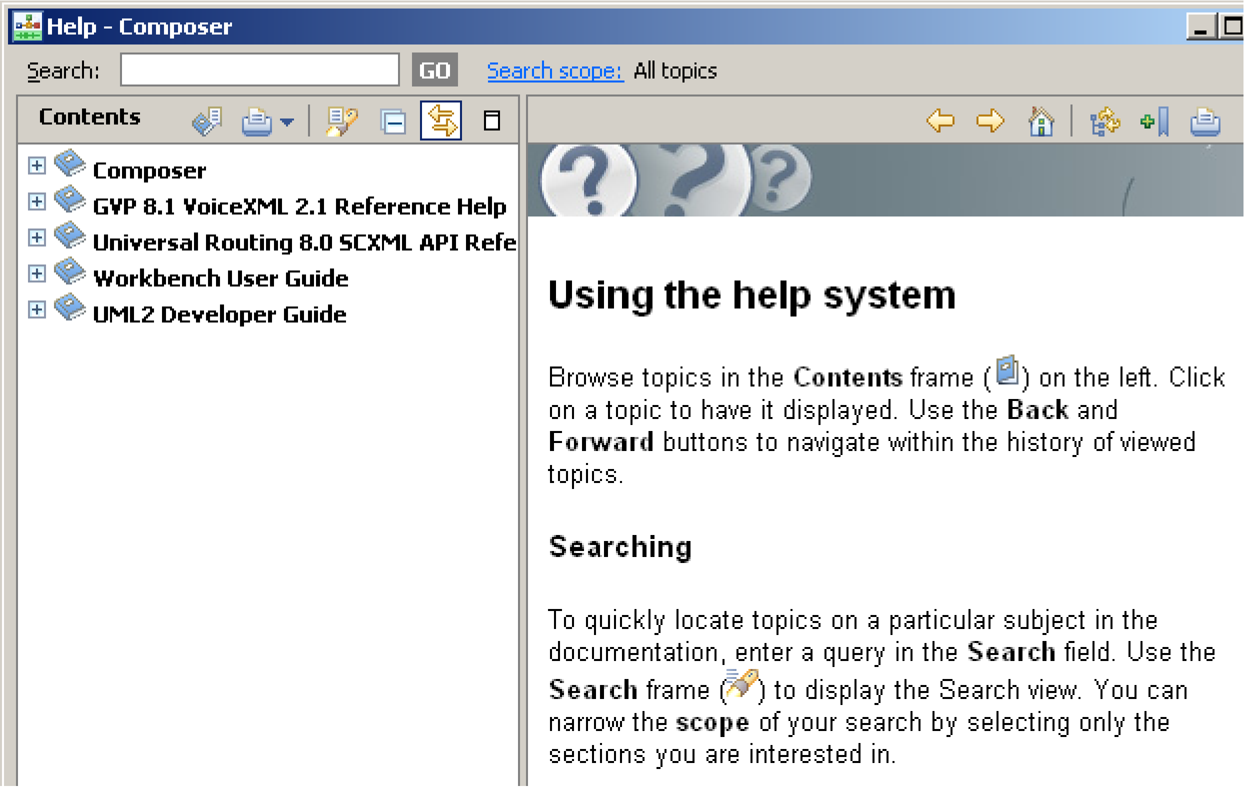
Supplements

Related Documentation Resources

The following resources provide additional information that is relevant to this software. Consult these additional resources as necessary.

Composer Help

- *Composer 8.0.4 Help,* which provides integrated help information about using Composer to develop voice and routing applications. Selecting `Help` > `Help Contents` brings up the window shown in Figure 61.



Using the help system

Browse topics in the **Contents** frame ( ) on the left. Click on a topic to have it displayed. Use the **Back** and **Forward** buttons to navigate within the history of viewed topics.

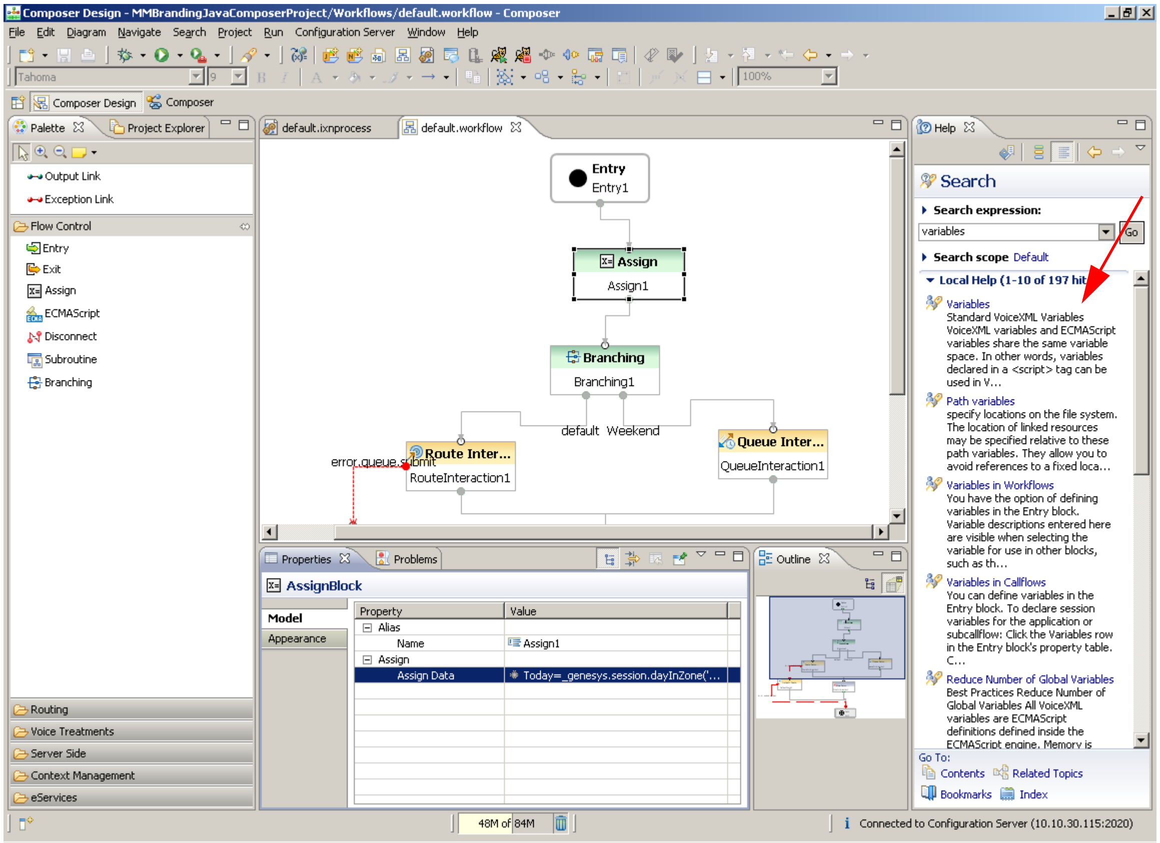
Searching

To quickly locate topics on a particular subject in the documentation, enter a query in the **Search** field. Use the **Search** frame ( ) to display the Search view. You can narrow the **scope** of your search by selecting only the sections you are interested in.

Figure 61: Composer Help Dialog Box

- Double-click Composer.
- Or, since Composer is based on Eclipse, you should familiarize yourself with basic Eclipse concepts by referring to the *Workbench User Guide* shown in Figure 61.

Selecting `Help` > `Search` opens a search pane on the right side of the Composer window. Figure 62 shows example search results.



Property	Value
⊟ Alias	
Name	Assign1
⊟ Assign	
Assign Data	Today=_genesys.session.dayInZone('...

Search expression: variables Go

Search scope Default

Local Help (1-10 of 197 hit

Variables
Standard VoiceXML Variables VoiceXML variables and ECMAScript variables share the same variable space. In other words, variables declared in a <script> tag can be used in V...

Path variables
specify locations on the file system. The location of linked resources may be specified relative to these path variables. They allow you to avoid references to a fixed loca...

Variables in Workflows
You have the option of defining variables in the Entry block. Variable descriptions entered here are visible when selecting the variable for use in other blocks, such as th...

Variables in Callflows
You can define variables in the Entry block. To declare session variables for the application or subcallflow: Click the Variables row in the Entry block's property table. C...

Reduce Number of Global Variables
Best Practices Reduce Number of Global Variables All VoiceXML variables are ECMAScript definitions defined inside the ECMAScript engine. Memory is

Connected to Configuration Server (10.10.30.115:2020)

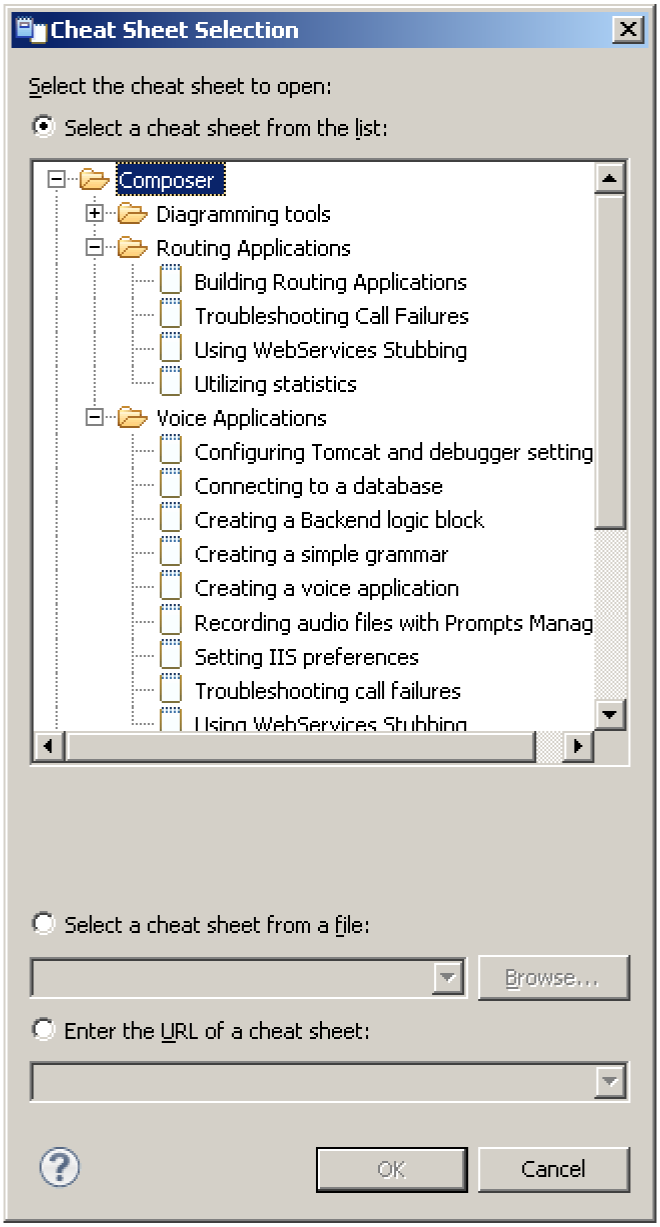
Figure 62: Help Tab for Searching

Composer 8.0 Deployment Guide

- This guide describes how to install Composer and perform post-installation configuration. It also describes software requirements, system requirements, launching Composer for the first time, and uninstallation.

Cheat Sheets

- Selecting `Help` > `Cheat Sheets` opens a dialog box where you can expand `Composer` and select tutorials to quickly get started with the concepts. This includes tutorials for pre-configurations like for the SIP Phone settings, creating first voice application (see Figure 63).



Cheat Sheet Selection

Select the cheat sheet to open:

● Select a cheat sheet from the list:

- Composer
  - Diagramming tools
  - Routing Applications
    - Building Routing Applications
    - Troubleshooting Call Failures
    - Using WebServices Stubbing
    - Utilizing statistics
  - Voice Applications
    - Configuring Tomcat and debugger setting
    - Connecting to a database
    - Creating a Backend logic block
    - Creating a simple grammar
    - Creating a voice application
    - Recording audio files with Prompts Manag
    - Setting IIS preferences
    - Troubleshooting call failures
    - Using WebServices Stubbing

○ Select a cheat sheet from a file:

○ Enter the URL of a cheat sheet:

Figure 63: Cheat Sheets

Management Framework

- *Framework 8.0 Deployment Guide,* which provides information about configuring, installing, starting, and stopping Framework components.
- *Framework 8.0 Genesys Administrator Help,* which provides information about configuring and provisioning contact center objects by using the Genesys Administrator.
- *Framework 8.0 Configuration Options Reference Manual,* which provides descriptions of the configuration options for Framework components.

- *Framework 8.0 Stat Server User's Guide*, which describes the configuration, installation, and start procedures relevant to deploying Stat Server.

SIP Server

- *Framework 8.0 SIP Server Deployment Guide,* which provides information about configuring and installing SIP Server.

Universal Routing

- *Universal Routing 8.0 SCXML API Reference*. This help file, available via **Help** > **Contents** (see Figure 61 on page 137), groups URS extensions to the SCXML executable content by Genesys-supplied Functional Module.
- *Orchestration Server 8.0 Deployment Guide*. Contains deployment information for Genesys Orchestration Server, which offers an open standards-based platform with an SCXML engine enabling intelligent distribution of interactions throughout the enterprise. Orchestration Server interprets the top-level SCXML document created as a result of an interaction process diagram created in Composer.

eServices/Multimedia

- *Universal Contact Server 8.0 Context Services Services User's Guide*, which provides information on the Universal Contact Server database of customer-related, service, and interaction-centric data (current and historical). Composer's Context Services blocks use this database.

Genesys Voice Platform

- *Genesys Voice Platform 8.1 Deployment Guide,* which provides information about installing and configuring Genesys Voice Platform (GVP).
- *Genesys Voice Platform 8.1 User's Guide,* which provides information about configuring, provisioning, and monitoring GVP and its components.
- *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help* (see Figure 61 on page 137), which provides information about developing Voice Extensible Markup Language (VoiceXML) applications. It presents VoiceXML concepts, and provides examples that focus on the GVP Next Generation Interpreter (NGI) implementation of VoiceXML.
- *Genesys Voice Platform 8.1 Legacy Genesys VoiceXML 2.1 Reference Manual*, which describes the VoiceXML 2.1 language as implemented by the Legacy GVP Interpreter (GVPi) in GVP 7.6 and earlier, and which is now supported in the GVP 8.1 release.

- *Genesys Voice Platform 8.1 CCXML Reference Manual,* which provides information about developing Call Control Extensible Markup Language (CCXML) applications for GVP.
- *Genesys Voice Platform 8.1 Troubleshooting Guide,* which provides information about Simple Network Management Protocol (SNMP) Management Information Bases (MIBs) and traps for GVP, as well as troubleshooting methodology.
- *Genesys Voice Platform 8.1 Configuration Options Reference,* which replicates the metadata available in the Genesys provisioning GUI, to provide information about all the GVP configuration options, including descriptions, syntax, valid values, and default values.
- *Genesys Voice Platform 8.1 Metrics Reference,* which provides information about all the GVP metrics (VoiceXML and CCXML application event logs), including descriptions, format, logging level, source component, and metric ID.

Voice Platform Solution

- *Voice Platform Solution 8.1 Integration Guide,* which provides information about integrating GVP 8.1, SIP Server 8.0, and, if applicable, IVR Server.

Open Standards

- *W3C Voice Extensible Markup Language (VoiceXML) 2.1, W3C Recommendation 19 June 2007,* which is the World Wide Web Consortium (W3C) VoiceXML specification that GVP NGI supports.
- *W3C Voice Extensible Markup Language (VoiceXML) 2.0, W3C Recommendation 16 March 2004,* which is the W3C VoiceXML specification that GVP supports.
- *W3C Speech Synthesis Markup Language (SSML) Version 1.0, Recommendation 7 September 2004,* which is the W3C SSML specification that GVP supports.
- *W3C Voice Browser Call Control: CCXML Version 1.0, W3C Working Draft 29 June 2005,* which is the W3C CCXML specification that GVP supports.
- *W3C Semantic Interpretation for Speech Recognition (SISR) Version 1.0, W3C Recommendation 5 April 2007,* which is the W3C SISR specification that GVP supports.
- *W3C Speech Recognition Grammar Specification (SRGS) Version 1.0, W3C Recommendation 16 March 2004,* which is the W3C SRGS specification that GVP supports.

Genesys

- *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and computer-telephony integration (CTI) terminology and acronyms used in this document.
- *Genesys Migration Guide*, which ships on the Genesys Documentation Library DVD, and which provides documented migration strategies for Genesys product releases. Contact Genesys Technical Support for more information.
- Release Notes and Product Advisories for this product, which are available on the Genesys Technical Support website at `http://genesyslab.com/support`.

Information about supported hardware and third-party software is available on the Genesys Technical Support website in the following documents:

- *Genesys Supported Operating Environment Reference Manual*
- *Genesys Supported Media Interfaces Reference Manual*

Consult these additional resources as necessary:

- *Genesys Hardware Sizing Guide,* which provides information about Genesys hardware sizing guidelines for the Genesys 7.x and 8.x releases.
- *Genesys Interoperability Guide,* which provides information on the compatibility of Genesys products with various Configuration Layer Environments; Interoperability of Reporting Templates and Solutions; and Gplus Adapters Interoperability.
- *Genesys Licensing Guide,* which introduces you to the concepts, terminology, and procedures relevant to the Genesys licensing system.
- *Genesys Database Sizing Estimator 7.6 Worksheets,* which provides a range of expected database sizes for various Genesys products.

Document Conventions

This document uses certain stylistic and typographical conventions—introduced here—that serve as shorthands for particular kinds of information.

Document Version Number

A version number appears at the bottom of the inside front cover of this document. Version numbers change as new information is added to this document. Here is a sample version number:

80fr_ref_06-2008_v8.0.001.00

You will need this number when you are talking with Genesys Technical Support about this product.

Screen Captures Used in This Document

Screen captures from the product graphical user interface (GUI), as used in this document, may sometimes contain minor spelling, capitalization, or grammatical errors. The text accompanying and explaining the screen captures corrects such errors *except* when such a correction would prevent you from installing, configuring, or successfully using the product. For example, if the name of an option contains a usage error, the name would be presented exactly as it appears in the product GUI; the error would not be corrected in any accompanying text.

Type Styles

Table 24 describes and illustrates the type conventions that are used in this document.

Table 24: Type Styles

Type Style	Used For	Examples
Italic	• Document titles • Emphasis • Definitions of (or first references to) unfamiliar terms • Mathematical variables Also used to indicate placeholder text within code samples or commands, in the special case where angle brackets are a required part of the syntax (see the note about angle brackets on page 144).	Please consult the *Genesys Migration Guide* for more information. Do *not* use this value for this option. A *customary and usual* practice is one that is widely accepted and used within a particular industry or profession. The formula, $x + 1 = 7$ where x stands for . . .

Table 24: Type Styles (Continued)

Type Style	Used For	Examples
Monospace font (Looks like `teletype` or `typewriter text`)	All programming identifiers and GUI elements. This convention includes: • The *names* of directories, files, folders, configuration objects, paths, scripts, dialog boxes, options, fields, text and list boxes, operational modes, all buttons (including radio buttons), check boxes, commands, tabs, CTI events, and error messages. • The values of options. • Logical arguments and command syntax. • Code samples. Also used for any text that users must manually enter during a configuration or installation procedure, or on a command line.	Select the `Show variables on screen` check box. In the `Operand` text box, enter your formula. Click `OK` to exit the `Properties` dialog box. T-Server distributes the error messages in `EventError` events. If you select `true` for the `inbound-bsns-calls` option, all established inbound calls on a local agent are considered business calls. Enter `exit` on the command line.
Square brackets ([ ])	A particular parameter or value that is optional within a logical argument, a command, or some programming syntax. That is, the presence of the parameter or value is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information.	`smcp_server -host [/flags]`
Angle brackets (< >)	A placeholder for a value that the user must specify. This might be a DN or a port number specific to your enterprise. **Note:** In some cases, angle brackets are required characters in code syntax (for example, in XML schemas). In these cases, italic text is used for placeholder values.	`smcp_server -host <confighost>`

Index