Predictive Routing Deployment and Operations Guide

Deploying the Composer Strategy Subroutines

5/6/2025

Contents

Deploying the Composer Strategy Subroutines

Genesys Predictive Routing (GPR) provide subroutines components that are integrated with your Genesys Routing solution. The subroutines are placed within an existing Composer workflow, where they add agent scoring and best-match functionality that enables you to fine-tune the routing of a specific interaction to the agent who can best handle it, based on the KPIs you want to optimize.

Genesys Predictive Routing provides pre-configured subroutines for use with either:

- Interaction Routing Designer (IRD) and Universal Routing Server (URS)
- Composer, Orchestration Server (ORS), and URS

These strategy subroutines are *static strategies*, that is, they run automatically once they have been set up. They respond to configured conditions without needing ongoing adjustments.

> ## Important
>
> - If you would like to evaluate Genesys Predictive Routing for use with schedule-based routing (using Genesys Workforce Management), service-level routing, or business-objective routing, contact Genesys Professional Services for a review of your routing environment.
>
> - For descriptions of how the subroutines handle agent-surplus and interaction-surplus scenarios, see Interaction Flows.

Out-of-the-Box Composer/ORS/URS Workflow Subroutines

The Composer strategy subroutines IP provides a complete Composer project, including all the workflows necessary to perform predictive routing, and including an example workflow. The Composer strategy subroutines make use of the underlying functionality provided by the URS subroutines described above.

When you extract the **ScoreBasedRoutingComposer_v4.zip** file, which contains the Composer subroutines, you get a top-level folder called **ScoreBasedRoutingComposer_v4**, with various subfolders, including the following:

The **Workflows** subfolder contains the following workflow files:

- Activate GPR.workflow

- GetActionFilters.workflow
- GetCustomerRecord.workflow
- GetScoringAuthToken.workflow
- GPRIxnCompleted.workflow
- Main.workflow
- SetScoreMaps.workflow

The **src** subfolder contains a subsubfolder called **subroutines**, which contains the following SCXML files:

- ClearTargets.scxml
- InteractionAccept.scxml
- QueueCancel.scxml
- QueueQuery.scxml
- SuspendForEvent.scxml

The **src-gen** subfolder contains the following SCXML files:

- Activate GPR.scxml
- GetActionFilters.scxml
- GetCustomerRecord.scxml
- GetScoringAuthToken.scxml
- GPRIxnCompleted.scxml
- IPD_GPRDemo_GPRMain.scxml
- Main.scxml
- SetScoreMaps.scxml

The ActivateGPR subroutine (called ActivatePredictiveMatching in earlier releases), after you insert it into your Composer workflow, triggers the ActivatePredictiveRouting_v<*version_number*> URS subroutine.

The GPRIxnCompleted (called PMIxnCleanup in earlier releases) and (in releases prior to 9.0.007.00) PMIxnCleanup subroutines can be inserted into the Composer workflow in the appropriate places. They perform the same functions of attaching matching user data and cleaning up the URS global map as the comparable GPRIxnCompleted and PrrIxnCleanup subroutines.

Other Files

The following files included in the URS Strategy Subroutines component IP are required for the Composer Strategy Subroutines component to function correctly.

- **SetIdealAndReadyConditionforOCS**: An RBN file included in the URS Strategy Subroutines component IP, which is required for the Composer Strategy Subroutines component to function correctly.

- **objects.kvlt**: A text file containing objects required by the strategy subroutines. You must import thais file as well as the RBN file for the strategy subroutines to work correctly.

Installing the Composer/ORS/URS Strategy Subroutines

The following is a high-level overview of the steps required to deploy the Composer/ORS/URS Strategy Subroutines:

1. Configuring URS to Support Predictive Routing.

2. Import the subroutines. For a list of the subroutines, with descriptions of their functionality, see Composer/ORS/URS Strategy Subroutines.

3. Define the entry points in your Composer workflow for the appropriate subroutines. See the complete integration instructions for Composer for specific recommendations and configuration information.

4. Set appropriate values for the strategy subroutine configuration options, located in the Predictive_Route_CfgData Transaction List object.

5. Configure the parameters for the subroutines used in your environment.

6. Test that the subroutines are correcting directing interactions to agents.

The following section provides detailed instructions for setting up your subroutines:

- Composer/Orchestration Server (ORS)/Universal Routing Server (URS)

Configuring URS to Support Predictive Routing

There are two main steps to configure URS to work with Predictive Routing:

1. Create the **static_strategy** configuration option in the **[default]** section of the URS Application object and set its value to empty. You can set this option either on the level of individual routing points or on the tenant/URS level. Depending on where you set it, the option works slightly differently. This option takes effect immediately and does not require that you restart URS.

2. Configure the http log for URS to check scoring requests and responses. To do this, set the **verbose** option in the **[web]** section to 3.

3. Set the **run_verbose** option in the **[default]** section to 0.

4. Set the **vqtime** option in the **[default]** section to 13:2048.

Strategy Subroutine Integration for ORS/Composer

Predictive Routing comes with a pre-configured Composer strategy subroutine. The Composer subroutine ActivatePredictiveMatching is a wrapper around the URS subroutine

ActivatePredictiveRouting and calls the latter via URS web API call. You can use it as-is, edit it, or create your own based on the model of the pre-configured subroutine.

Design Considerations

The subroutine that comes with Predictive Routing has been created to work efficiently. It:

1. Minimizes the use of IRD subroutines invoked by the Composer workflow.

2. Reuses existing IRD subroutines as much as possible to avoid maintenance and upgrade of multiple versions.

3. Uses the same configuration options as the strategy subroutines for URS, specifically those configured in the Predictive_Route_DataCfg `Transaction List` object.

Functional Blocks and their Behaviors

The ActivatePredictiveMatching subroutine contains a number of functional blocks:

- The Entry block handles the setAgtScores timeout, which is set once before the agent score memory maps are set.

- The Score request content **format_as_map** can be set to set to `false`, which returns the agent scores as a JSON array with entries ordered by decreasing score, or to `true`, which returns scores as a dictionary, where the keys are employee IDs for the targeted agents and values are their scores for the interaction.

- The WebRequestScore block is configured to use a single retry in the event of an error. After one retry, if the retry fails, an error is logged and recorded in the attached data and it returns a value of `false`.

- The AgentScores ECMA script block parses the returned JSON array and prepares a JSON object with entries ordered by score. In case of JSON object parsing error the script assigns variables varResult.success = "false" and varResult.message = "Parsing of agent scores failed".

- The AgentScoreMaps subroutine invocation block includes the **parCustomerId** parameter, which maintains consistency with the IRD version.

- The PMSetIdealAndReady SCXML State block invokes the IRD SetIdealAndReadyConditionForORS strategy through an HTTP request to URS. Details of the invocation can be found below.
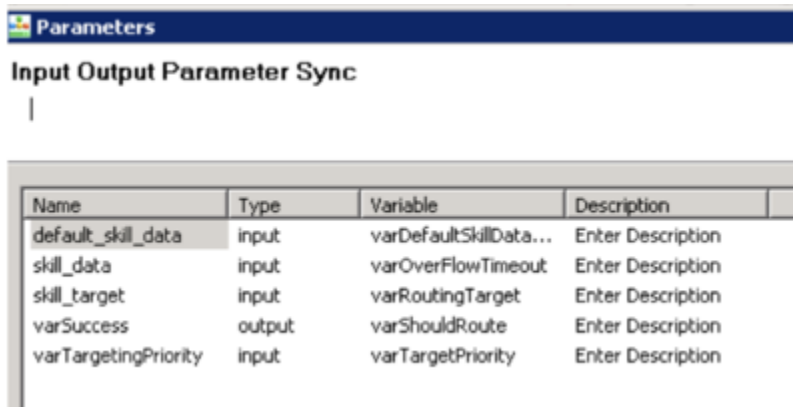
After a target is returned from the Target block in the ActivatePredictiveMatching subroutine, the PMIxnCompleted and PMIxnCleanup subroutines are invoked. They work essentially as the PrrIxnCompleted and PrrIxnCleanup IRD subroutines do, except that the PMIxnCleanup subroutine takes advantage of new memory map cleanup functionality implemented in URS 8.1.400.37, and therefore does not explicitly remove agent score data from the memory map.

Importing and Configuring the Composer Workflows

To integrate the GPR Composer subroutines into your routing environment, you must import the workflows.

1. In Composer, select **Window > Preferences > Composer > SCXML Templates**.

2. In the resulting **Templates** dialog box, click the **Import...** button.

3. Navigate to the folder containing the SCXML file, select it, and click **Open**.

The main workflow should invoke the ActivateGPR subroutine with the parameters shown in the following image:


> ## Important
>
> The input variables **default_skill_data**, **skill_data**, and **skill_target** are equivalent to the IRD implementation (above). The **varTargetPriority** value is the targeting priority to be used. The output is a Boolean value indicating success if `true`, `false` if targeting fails.

The implementation and invocation of the ORS SetIdealAndReadyConditionForORS subroutine are a key part of the Composer integration. The implementation within this IRD subroutine provides the URS callbacks for the SetIdealAgent and SetReadyCondition functions. This enables the use of virtual queues for reporting and agent reservation within the main workflow implementation.

The workflow performs scoring authentication in the GetScoringAuthToken subroutine workflow, which obtains the url, api key, and so on from the Predictive_Route_DataCfg Transaction List object, just as in the IRD implementation. The agent scores are obtained from AICS, and URS memory maps are populated with agent ID, connid, and agent score. Again, this is equivalent to the IRD implementation. Next the workflow invokes the SetIdealAndReadyConditionForORS subroutine. After completion, normal workflow targeting proceeds.

The SetIdealAndReadyConditionForORS IRD subroutine uses the IRD function, RequestRouter, shown below, to invoke the SetIdealAgent and SetReadyCondition functions through an internal URS HTTP interface, directed at ##SELF, the URS on which the call is being handled. The Context is the `connid` to be used.
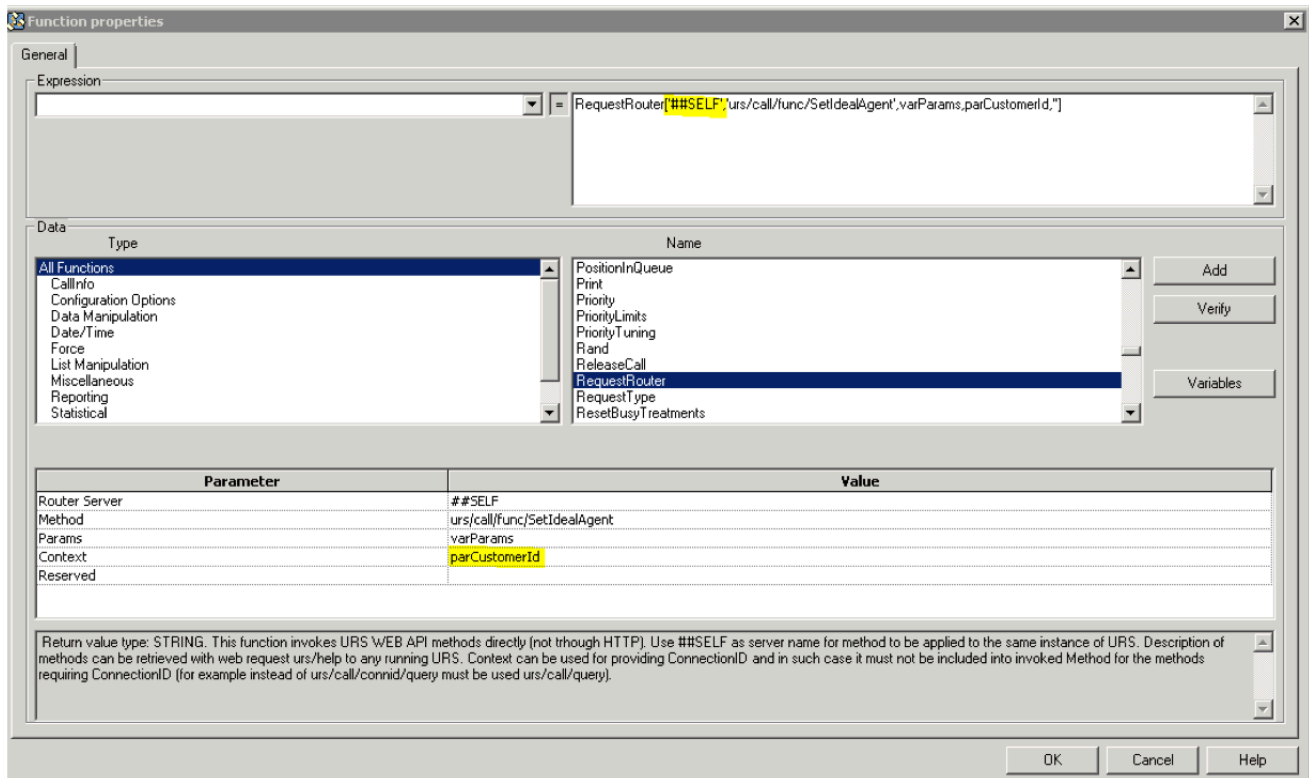
The example shows the invocation of the SetIdealAgent URS function. The equivalent is performed for the SetReadyCondition function as well.

Invoking the SetIdealAndReadyConditionForORS from a Composer workflow details are shown below.
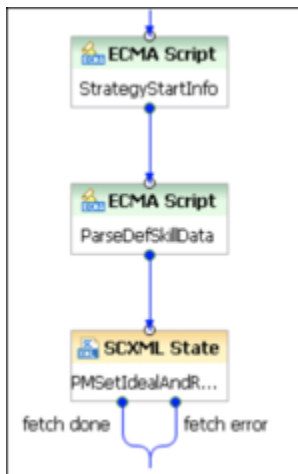
The two ECMA Script blocks prepare the required information for the invocation, which occurs in the SCXML State block.

Setting the Configuration Options

Set values for the following configuration options in the Predictive_Route_DataCfg Transaction List

Object.

All About How the Composer Workflow Blocks Work

Details of each block follow.

StrategyStartInfo

This block sets the following variables as below varConnID, varURSRequestURL, and ursRequestTimeout are User variables.

```
try {

        // get the connid of this interaction

        varConnID = _genesys.ixn.interactions[system.InteractionID].voice.connid;

        // set urs call start request

        var ursHost = _genesys.session.getConfigOption('null', "hostname",
         _genesys.session.lookupseq.StartFromRouter);
        var ursHttpPort = _genesys.session.getConfigOption('null', "http_port",
        _genesys.session.lookupseq.StartFromRouter);
                varURSRequestURL =  'http://' + ursHost + ':' + ursHttpPort +'/urs/call/'
+varConnID + '/exec';

        // fetch timeout in seconds for strategy - TODO - add to xlist

        ursRequestTimeout = 15;

} catch (error) {
        __Log('###DWS StrategyStartInfo error l = ' + uneval(error)  );
}
```

ParseDefSkillData

The User variable varDefaultSkillDataString is populated from the default_skill_data input variable to the ActivatePredictiveMatching.workflow subroutine, as show below.

```
try {

        var vKeys = Object.keys( default_skill_data );

        // want "prr-ixn-timestamp:64407|predictor:qaart-predictor|AgentScore:Y"

        var vKeyCount = vKeys.length;
        for ( var iKey = 0; iKey < vKeyCount; iKey++ )
        {
                varDefaultSkillDataString  = varDefaultSkillDataString + vKeys[iKey]  +':'
+   default_skill_data[vKeys[iKey]];
                if ( iKey < vKeyCount - 1)  varDefaultSkillDataString =
varDefaultSkillDataString + '|';
        }

} catch (error) {
        __Log('###PRRDemo ActivatePredictiveRouting PatseDefSkillData  error  = ' +
uneval(error)  );
```

```
}
```

PMSetIdealAndReady

The onentry element performs the invocation with the following parameters:

- skill_target and skill_data parameters: Equivalent to the IRD implementation. Input variables to the ActivatePredictiveMatching.workflow subroutine.

- varPredictorCfg is also the equivalent to the IRD implementation, retrieved from the Predictive_Route_DataCfg Transaction List object for the defined predictor.

```
<onentry>
        <session:fetch requestid="ursCallRequestId" srcexpr="varURSRequestURL" method="'get'"
timeout="ursRequestTimeout" >
                <param name="tenant" expr="_genesys.session.tenant"/>
                <param name="strategy" expr="'SetIdealAndReadyConditionForORS'"/>
                  <param name="TARGET" expr="skill_target"/>
                <param name="TARGET_TIME_TO_WAIT" expr="skill_data"/>
                <param name="CustomerId" expr="parCustomerId"/>
                <param name="DefaultSkillData" expr="varDefaultSkillDataString"/>
                <param name="PredictorCfg" expr="varPredictorCfg"/>
        </session:fetch>
</onentry>
```

The transitions from the state are as shown below. The session.fetch.done event body sets the varSuccess User variable to true.
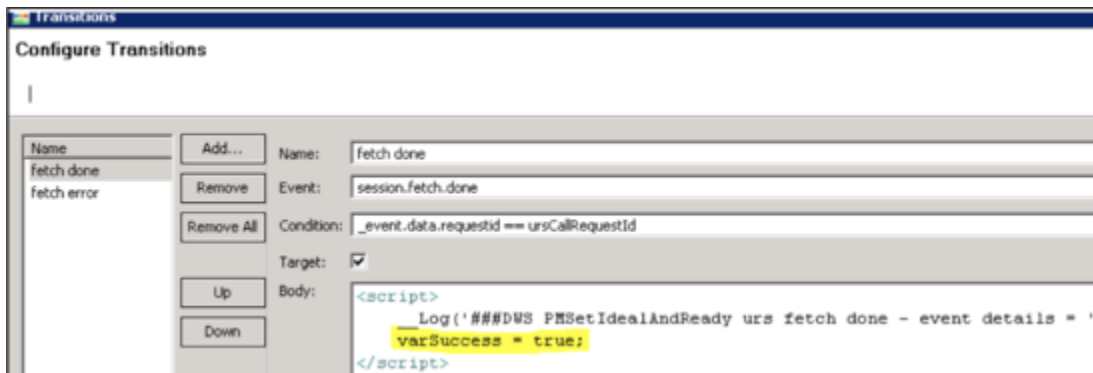
The error.session.fetch event body leaves the varSuccess User variable at the default value of false.
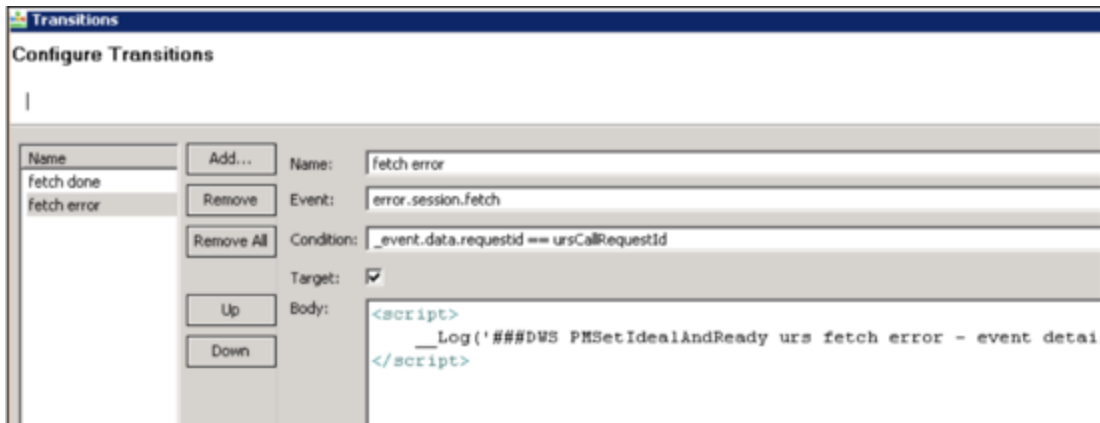
Once this block has completed, the subroutine exits to the main workflow where normal targeting occurs if the invocation was successful. If not, the sample main workflow still performs normal targeting, but skips the PMIxnCompleted and PMIxnCleanup as shown below.
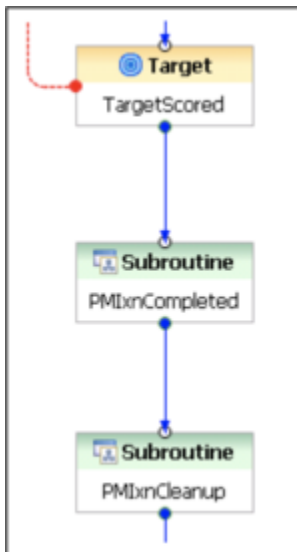
The PMIxnCompleted input is simply the resource selected returned from the Target block.

The PMIxnCleanup input parameters are the varInteractionId, which is the connid of the call being processed, and varResourceSelected, which is the same as for PMIxnCompleted.