Oracle® Real-Time Decisions

Getting Started with Oracle RTD

ORACLE

Version 2.2 Document Revised February 2007

Getting Started with Oracle RTD

Preface

Oracle Real-Time Decisions (Oracle RTD) enables you to develop adaptive enterprise software solutions. These adaptive solutions continuously learn from business process transactions while they execute and optimize each transaction, in real time, by way of rules and predictive models.

About this document

This document contains information and examples to help you get started using Oracle RTD. The examples in this guide assume the reader has installed Oracle RTD on a Windows system.

Intended audience

This document is designed to help technical users of Oracle RTD get acquainted with the capabilities, terminology, tools, and methodologies used to configure Inline Services.

How to use this guide

This document is divided into the following sections:

Section 1: About Oracle RTD provides background on Oracle RTD.

Section 2: Creating an Inline Service explains how to build an Oracle RTD inline service.

Section 3: Simulating Load for Inline Services provides information about using the Load Generator to simulate the run-time operation of the system.

Section 4: Enhancing the Call Center Inline Service expands the functionality of the Inline Service.

Section 5: Closing the Feedback Loop further enhances the Inline Service to add a self-learning model that uses predictive methods to update the Inline Service.

Document conventions

Convention	Description
`monospace`	Indicates source code and program output.
bold	Indicates portions of the user interface, such as labels, tabs, and menus.
italic	Italics are used to highlight the first use of terms.
'quote'	Indicates input required from the user.
	Indicates additional information that may make the task easier.
	Indicates additional information about the subject.
	Indicates actions that may result in loss of data or errors.

Section 1: About Oracle RTD

Oracle RTD provides a new generation of enterprise analytics software solutions that enable companies to make better decisions in real time at key, high-value points in operational business processes.

Oracle RTD easily integrates with enterprise applications both on the front end (such as CRM applications) and on the back end (such as enterprise data stores). Oracle RTD also includes other helpful load testing and debugging tools.

1.1 Terminology

Oracle RTD consists of five components:

- Decision Studio
- Real-Time Decision Server
- Decision Center
- Administration (JMX)
- Load Generator

Inline Service refers to the configured application that is deployed.

Inline Services are configured and deployed using Decision Studio and analyzed and updated using Decision Center. Inline Services run on Real-Time Decision Server.

An Inline Service can gather data and analyze characteristics of enterprise business processes on a real-time and continuous basis. It also leverages that data and analysis to provide decision-making capability and feedback to key business processes.

Elements are one of the following types of objects:

1. Application: The application object identifies the application level settings for models, and any parameters needed for the Inline Service.
2. Performance Goals: Performance Goals identify the Key Performance Indicators (KPIs) that the Inline Service is designed to track and optimize.
3. Choices: Choices represent the offers that will be presented through the Inline Service or the attributes to be tracked by the self learning model.
4. Rules: Rules are graphically configured rules used to target segments of population, decide whether a choice is eligible or score a particular choice.
5. Decisions: Decisions score and weigh the eligible choices and present the one that optimizes performance goals.
6. Selection Functions: Selection Functions can be used by decisions as a custom way to make a choice.
7. Entities: Entities represent the actors in the system.
8. Data sources: Data Sources retrieve data from tables or stored procedures.
9. Integration Points: Integration Points are the places where the Inline Service touches outside systems, either with data coming in or advice going out. There are two classes of Integration Points: Informants and Advisors. Informants receive data from outside systems, whereas Advisors receive data and also send advice back to outside systems.
10. Models: Self-learning, predictive models optimize decisions and analyze data.
11. Statistics collectors: Statistic Collectors are special models that track statistics about entities.
12. Categories: Categories are used to segment data for display in Decision Center.

1.2 About Decision Studio

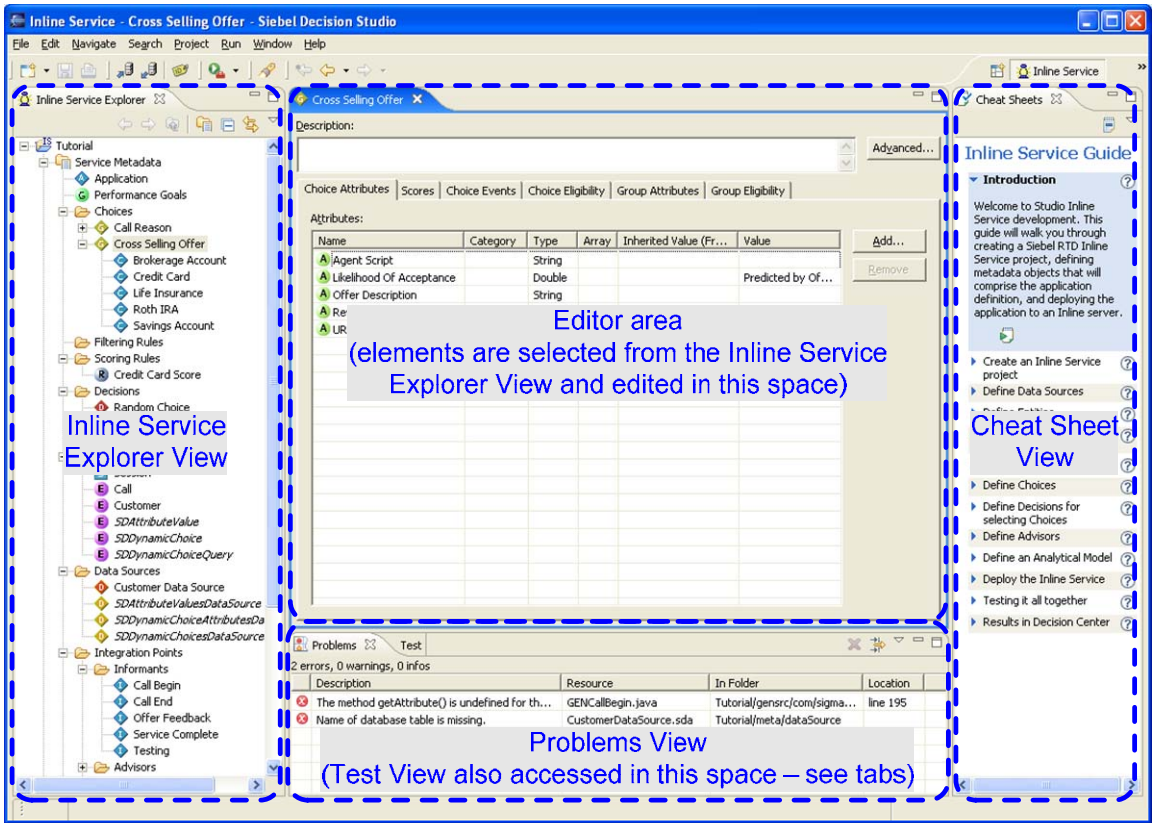
Decision Studio is a graphical development tool for configuring Inline Services, the services that allow you to monitor activity, gather statistics, and make recommendations.

Decision Studio is fully integrated with Eclipse, an open source Java IDE produced by the Eclipse Foundation. Decision Studio exists as a standard plug-in to the Eclipse environment. If you are using Eclipse, you have the advantage of using the environment for additional development and advanced features. If you are not familiar with Eclipse, it is completely transparent to using Decision Studio. Eclipse and Decision Studio both have online help available through the Help menu.

Decision Studio allows you to work with an Inline Service from several *perspectives*. A perspective defines the initial set and layout of *views* and *editors* for the perspective. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources. Perspectives control what appears in certain menus and toolbars.

To select or change to a different Perspective (such as Inline Service, Java, Resource, and so on), click the **Window** menu in Decision Studio and choose **Open Perspective**, then choose from the list of available Perspectives. The default Perspective when starting Decision Studio for the first time is **Inline Service**. We will use this Perspective in this tutorial. In general, this will be the Perspective you use to develop Inline Services.

The default **Inline Service** perspective contains four views and an editor area:



1.2.1 Inline Service Explorer View

The **Inline Service Explorer View** gives a view of the overall project structure. When you start a new Inline Service, this view is populated by all the elements of the Inline Service you have chosen.

1.2.2 Problems View

The **Problems View** shows validation (`.sda`) and compilation errors (`.java`) as you build your Inline Service. If you double-click a validation error, **Problems View** opens the metadata/element-editor at the point of the error. If you

double-click a compilation error, Problems View opens the generated source code (`.java` files) at the point of the error. You should *not* edit generated source code files directly; instead, fix related metadata/element problems, which will then regenerate and recompile the source code.

1.2.3 Test View

The Test View allows you to test your Inline Services against the server as you build them.

1.2.4 Cheat Sheets View

The Cheat Sheets View provides step-by-step instructions for common tasks. After installation, it is located on the right-hand side of the window.

Tip: You may want to close the Cheat Sheets View to give more editor space. The Cheat Sheets are unused in this tutorial.

1.2.5 Editor area

The center area of the Inline Service Perspective is the *editor area,* and shows an editor that is specific to the node on the project tree you have highlighted. To change to a new editor, double-click the element from the Inline Service Explorer View you wish to edit.

1.2.6 Arranging Views and Resizing Editors

Tabs on the editors indicate the name of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes. Tabs may have a toolbar that provides functionality.

You can drag the views and editors of a perspective to any space on the screen. Views and editors will resize themselves to fit the area in which they are placed. Occasionally, portions of an editor (where you do your main work) or view will become covered by other views, or resized to an area that is not convenient to use. To resize the editor or view, either close other open views and the remaining will automatically resize, or maximize the editor or view by double-clicking the editor tab.

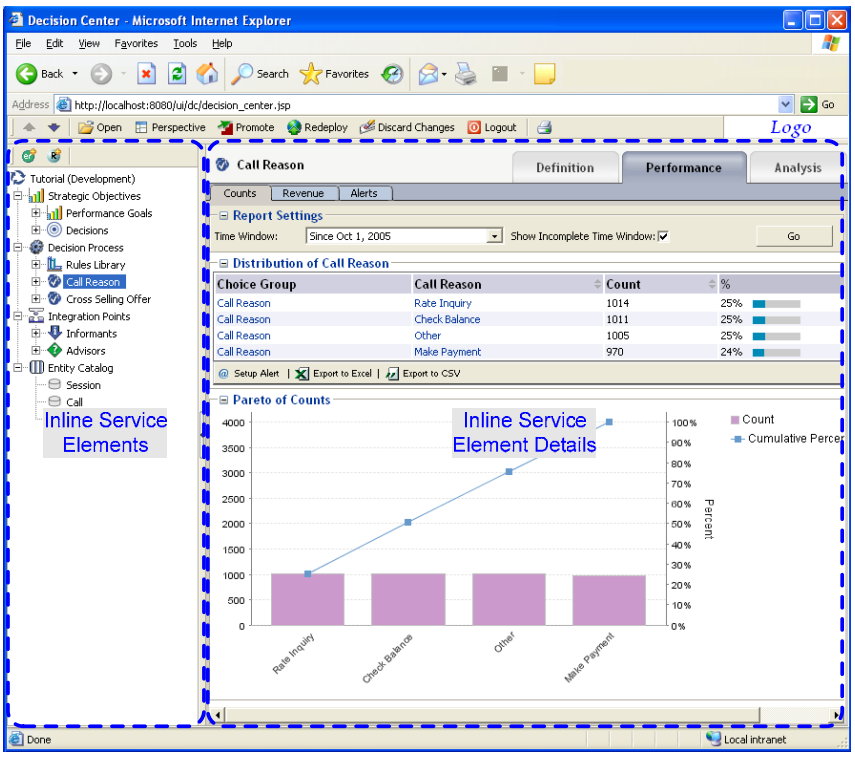
Both Editors and Views can be toggled between Maximize and Minimize by double-clicking the tab, or by using the right-click menu item.

To show additional Views or open Views that were closed, click the Window menu in Decision Studio and choose Show View, then choose from the list of available Views.

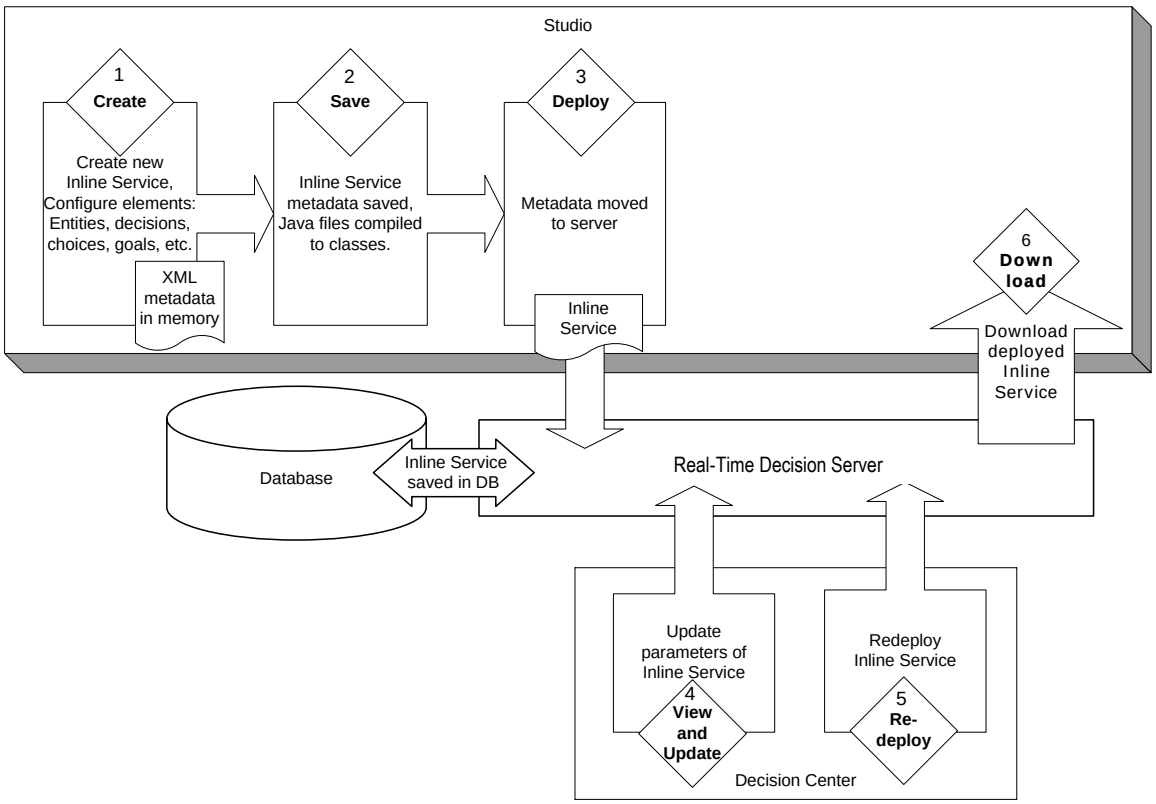
1.3 About Decision Center

Decision Center is a Web-based application that allows the business analyst to monitor and optimize deployed Inline Services. From Decision Center, you can view statistics gathered from the models and fine-tune campaigns such as cross-selling, as well as adjust how decisions are made.

The Decision Center user interface displays Inline Services in two panes. The left pane shows the list of Inline Service elements, while the right pane displays detailed information related to the selected element.



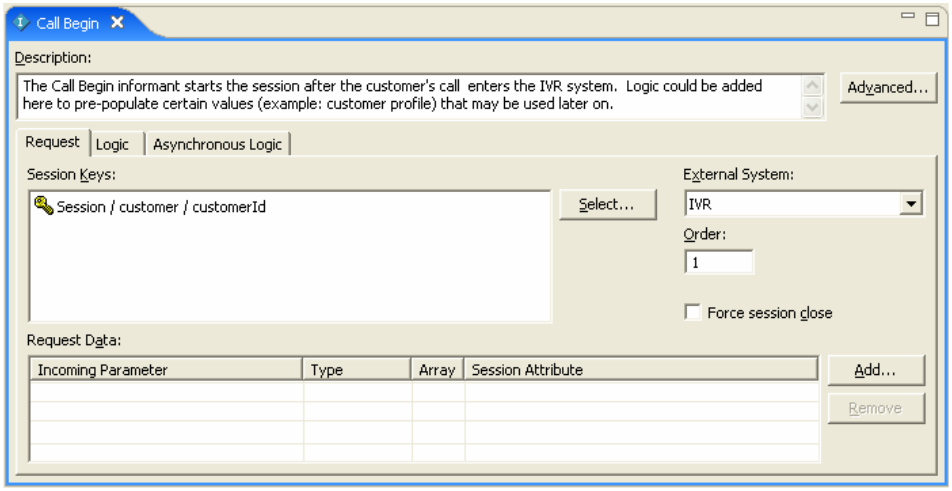
1.4 Overview of the Inline Service lifecycle



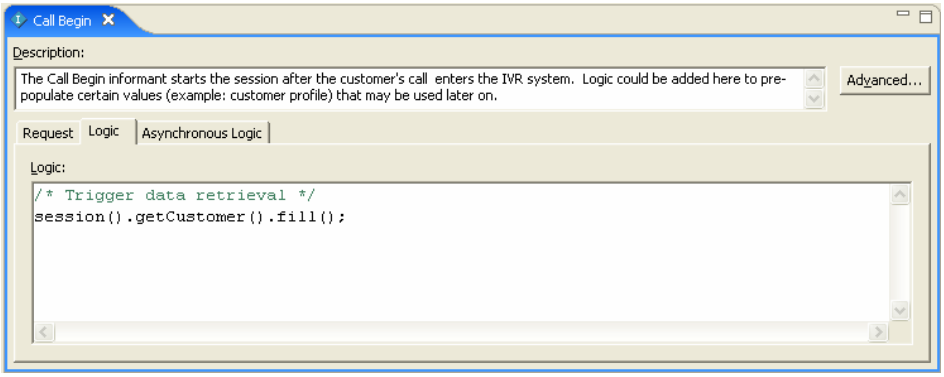
Inline Services are created using Decision Studio. The following steps outline the overall process by which Inline Services are created, deployed, and downloaded:

1. Create: Using Decision Studio, elements are created and configured. Configuration often consists of checking attributes or settings that apply to your business situation. Examples of elements are: Choice Groups, Performance Goals, Decisions, Informants, Advisors, Entities, and Data Sources.

   Some elements allow the use of Java scriptlets in Logic and Asynchronous Logic attributes. For instance, an Informant element is shown below. This element is named 'Call Begin.' In addition to the Description and the Advanced button, there are three tabs, each with a set of attributes for the Informant.



   In the Logic tab of this 'Call Begin' Informant, we can write optional Java code to perform specific tasks:



   As elements are created and saved, XML metadata is created in memory that describes the object.

2. Save: By saving the Inline Service in Decision Studio, the metadata is written to an Inline Service directory on the local file system, in XML files with the extension *.sda. The metadata for the Informant 'Call Begin' is saved in a file called `CallBegin.sda`, and the content is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sda:RTAPType xmlns:sda="http://www.sigmadynamics.com/schema/sda"
internalName="CallBegin" lastModifiedTime="1133228616435" name="Call
Begin" schemaVersion="20050818" forcesSessionClose="false" order="1.0">
  <sda:description>The Call Begin Informant starts the session after the
customer's call  enters the IVR system.  Logic could be added here to
pre-populate certain values (example: customer profile) that may be used
later on.</sda:description>
  <sda:system ref="Ivr"/>
  <sda:sessionKey path="customer.customerId" relativeTo="session"/>
  <sda:requestMapper internalName="RequestMapper">
    <sda:entity type="ApplicationSession" var="session"/>
```

```
    <sda:dataSource type="RequestDataSource" var="result">
      <sda:arg>
        <sda:path localVarName="session" path="request"
relativeTo="local"/>
      </sda:arg>
    </sda:dataSource>
  </sda:requestMapper>
  <sda:requestData internalName="RequestDataSource">
    <sda:param internalName="message" dataType="object"
objectType="com.sigmadynamics.client.wp.SDRequest"/>
    <sda:request>
      <sda:resultSet/>
    </sda:request>
  </sda:requestData>
  <sda:body>
    <sda:java order="0">/* Trigger data retrieval
*/&#xD;&#xA;session().getCustomer().fill(); </sda:java>
  </sda:body>
  <sda:postOutputBody/>
</sda:RTAPType>
```

The attributes that were assigned to the element in Decision Studio, such as Session Key and External System, are represented here. Note that the Java scriptlet is also inserted into the body of the XML file.

As Inline Service elements are added, configured, and saved, Decision Studio automatically generates the necessary Java code and compiles them into Java class files. Two classes of Java code are generated. The first set is the base Java files used by the Inline Service; these files are named the element id preceded by GEN. For example, the CallBegin element will produce a file called `GENCallBegin.java`.

The second set of Java files is created to allow overriding of the generated code. These files are named the same as the element id. For instance, the CallBegin element will produce a file named `CallBegin.java`. Note that by default, the Java class `CallBegin` simply extends the class `GENCallBegin`.

When the Inline Service is compiled, the generated code is used unless we specifically instruct that the override code be used. To do this, update and move the override Java file (for example, `CallBegin.java`) from the generated source files folder:

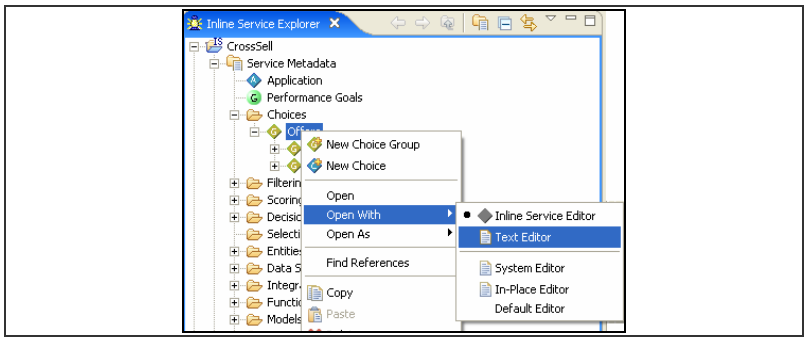
`[InlineServiceProjectRootFolder]\`**`gensrc`**`\com\sigmadynamics\sdo\`

to the override source files folder:

`[InlineServiceProjectRootFolder]\`**`src`**`\com\sigmadynamics\sdo\`

Decision Studio will now compile using the override Java file instead of the generated Java file.

Tip: The XML for any Inline Service object can be viewed with Decision Studio's built-in Text Editor. Right-click an Inline Service object in the Inline Service Explorer View and in the Open With menu, select Text Editor. To switch back to normal editor format, select the option Inline Service Editor. Note: You should not edit the XML (`*.sda`) files directly to modify the Inline Service objects; instead, use the corresponding Inline Service Editors.



3. Deploy: The Inline Service is deployed to the Real-Time Decision Server using Decision Studio. The Management Service on the server receives the metadata and compiled Inline Service files, stores the Inline Service in the database, and loads the Inline Service into memory. The Inline Service can now be utilized to process requests, view reports, and so on.

4. View and Update: Reports and learnings are viewed through the browser-based Decision Center interface. Selected elements and parameters of your Inline Service can be updated from Decision Center. Updated Inline Services are not available for run-time use until they are redeployed.

5. Redeploy: If updates are made to the Inline Service in Decision Center, the changes are made available for use by redeploying the Inline Service in Decision Center. The Management Service will regenerate all necessary metadata and Java files, recompile the Inline Service, store the Inline Service in the database, and load it in memory.

6. Download: Using Decision Studio, you can download a deployed Inline Service from the server. Downloading involves copying the Inline Service that resides in the database and placing all of the metadata, Java, and class files into a Decision Studio project on the hard drive. This is useful if you were not the original developer of the Inline Service and thus do not have the metadata files. Even if you had originally developed and deployed the Inline Service from Decision Studio, if your business process allows other users to make changes and redeploy the Inline Service through Decision Center, then to make additional changes to the Inline Service in Decision Studio, you would first need to download the latest version from the server.

Section 2: Creating an Inline Service

This section is designed to demonstrate how to build an Inline Service that acts as an Observer. Observer Inline Services are aimed at analyzing characteristics of target process on a real-time and continuous basis. An Observer Inline Service guides business users in their analysis of those various business events and how they change over time.

The Inline Service for this tutorial is based around a credit card company's call center. The Inline Service will collect data about the customer and the call center operational system and will analyze information about the call and its resolution.

The goal of this Inline Service is to analyze the patterns about calls, reasons for calling, and customers. In later sections, we will extend the capability of this Inline Service to provide recommendations to the CRM system on cross selling offers and then to add feedback to the service on the success of its recommendations.

2.1 Overview of the tutorial

An Inline Service is created using the Studio graphical development tool. In general, an Inline Service is created in the following fashion:

- A project is started in Decision Studio.
- Elements are added to that project and then configured to meet your business needs.
- Logic is added in the form of Java scriptlets to certain elements that perform operations.
- The Inline Service is deployed to the Real-Time Decision Server, where it runs.
- Results from the Inline Service are viewed through Decision Center.

In this tutorial the following elements are added and configured:

1. *Application:* The *Application* element establishes any application level settings that are needed, as well as defines security for the Inline Service. An Application element is automatically created for every Inline Service.
2. *Performance Goals:* Performance Goals represent organizational goals composed of metrics that are optimized using scoring. For instance, revenue and call duration are performance metrics. An organizational goal would be to maximize revenue while minimizing call duration.
3. *Data source:* The data source element acts as a provider of data from an outside data source. The structure and format of data from data sources can vary. For example:
   - Rows and columns of a RDBMS table
   - Output values and result row sets from a stored procedure

   A data source is a provider of data that you can map to Entity elements to supply the data for those elements.

   For example, in this tutorial we add a data source that connects to a table in the database. This table contains customer data.
4. *Entity:* The *Entity* is a logical representation of data that can be built from one or more data sources. Entities serve the following purposes:
   - To organize the data into objects for organizational, analytical, and modeling purposes.
   - To allow relatively easy and intuitive access from Java code of data from various sources.
   - To hide the details by which the data is obtained so that those details can change without requiring the logic to change.
   - To hide the mechanisms by which the data is obtained to save the user of this data from needing to deal with the APIs that are used to obtain the data.

- To support sharing of objects when objects need to be shared. For example, an object representing a service agent could be used in multiple sessions.

Attributes of an entity can be *key* values. The entity key is used to identify an instance of an entity.

For example, in this tutorial we create an entity to represent a customer. The attributes of that entity are mapped to the data source for values. The customer ID is chosen as the key for the customer entity.

Later we will also create an entity that represents a Call.

5. *Session Entity*: A special entity of an Inline Service is the *Session* entity. The Session entity represents a container for attributes that are specific to a particular defined Session. The *Session key* identifies the start and end of the Session.

   Entities that have been defined can be associated with the session by being made attributes of the Session Entity. Only Entities that are Session attributes can have their keys marked as session keys.

   For example, in this tutorial we add the Customer entity to the Session entity as an attribute, and then we choose the Customer key value, Customer ID, as a Session key.

6. *Informant:* An Informant is an Integration Point within the Inline Service that identifies the business interactions as they occur and triggers business logic that continuously identifies relevant statistical patterns in the data. Informants watch a process; they do not interact with it.

   In this tutorial we first create a testing Informant, and then create an Informant that gathers completion of service data from a CRM system.

   Later in the tutorial we create an Informant that provides feedback to the Inline Service on the success or failure of the predictions of the model.

7. *Choice Groups:* Choice Groups are useful for organizing choices. Choice Groups can be used in one of two ways: they provide a way to organize the observations that are collected and analyzed; they are also a way to organize the feedback we will give to the business process through the Advisor Integration Points.

   For example, in this tutorial we first create Choice Group that organizes the reason for calls. When we extend the Inline Service to include an Advisor, a Choice Group is used to organize cross sell offers that are recommended to the service center agent.

8. *Models:* Built in analytical models allow self-learning and automatic analysis. Models can be used to simply analyze data or to make recommendations to the business process.

   In this tutorial we create a model that analyzes the reasons for calls, and then later a model which helps to determine the most likely cross sell offer to be accepted by the customer.

9. *Decision:* A Decision is used by an Advisor to determine eligible Choices, score those Choices dynamically, weight the scoring according to segments of the population, and present to best-fit choice.

10. *Advisors:* Advisors extend the capability of the Inline Service by allowing a return of information into the business process. Advisors are tightly related to Choice Groups and Rules.

    In this tutorial we will create a Choice Group of offers that can be made to callers to the credit card service center. The Advisor calls on a Decision to determine the best offer for the caller based on information about the call and caller. The Advisor passes that cross sell recommendation to the front end application, so that the call center agent can make the offer.

2.2 A note about naming and descriptions

Element names and descriptions are used extensively in Decision Center, the user interface for business users. Therefore, it is very important that as you create elements you take the time to name them intuitively and to write good descriptions for all elements.

2.2.1 Before you begin

Before you begin, ensure that Real-Time Decision Server is started. See *Installation and Administration of Oracle RTD* for more information about how to start Real-Time Decision Server.

2.2.2 How to configure the Application element

1 Open Decision Studio by running *RTD_HOME*`\eclipse\eclipse.exe`. After Decision Studio opens, use File > New > Inline Service Project to begin a new project.

Note: This tutorial assumes you are using a new installation, with the original preferences set. If Decision Studio or Eclipse has been used in the past, you may want to switch to a new workspace. To switch to a new workspace, use File > Switch Workspace and choose a new workspace folder.

2 Enter the name for the Project, `Tutorial` and choose the Basic Application template. Click Finish. If you are asked about upgrading the Inline Service, select Yes. The Inline Service project is created in the Inline Service Explorer. By default, the files for the project/Inline Service are stored in the Decision Studio workspace, in a folder with the same name as the project (for example, `C:\Documents and Settings\Win_User\Oracle RTD Studio\Tutorial`).

3 In Studio, expand the Tutorial and Service Metadata folders. Double-click the Application element to bring up the element editor. In the element editor, type a description for the Tutorial Inline Service.

4 On the Permissions tab, use Add to add users to the application permissions. To choose a Real-Time Decision Server, click Select Server (by default `localhost:8080`). The Connect to a Server window appears. Enter the server Host Name and Port Number, then enter the User Name and Password of a user with Oracle RTD administrative permissions (permission `0`). Oracle RTD authentication and users were set up during Oracle RTD installation and configuration; see *Installation and Administration of Oracle RTD* for more information.

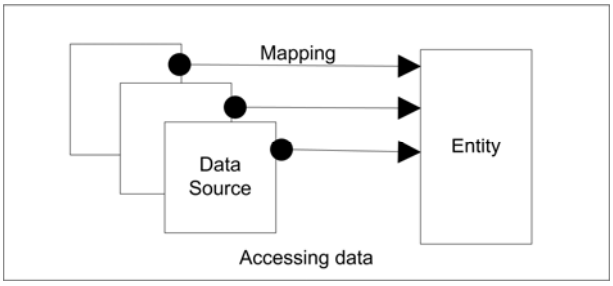
5 Click Connect. After you are logged on to the server, select the box Show users and use Get Names to display a list of users on the server. Your user name should be listed. Choose your user name and click OK.

6 Select your user and click under Granted to grant privileges. You will notice that some privileges imply other privileges. For instance, if you choose Deploy Service from Studio, you automatically get open, read and deploy privileges in Decision Center. Select all privileges.

2.3 Accessing data

In order to access organizational data, we will configure two elements:

- Data source: The data source is the element that represents the structure of the data in the database.
- Entity: The entity is a logical representation of data that can be populated by one or more data sources or contextual data retrieved by an Informant.



2.4 Creating a data source

1 Select Data Sources in the Inline Service Explorer and right-click it. Select New SQL Data Source. Enter the data source name, `Customer Data Source`, and click OK. The data source Editor appears.

2 Under Description, add the following description for the data source: `Customer data from a database table`.

Tip: Good descriptions are very important. These descriptions are used in Decision Center and are essential for business users to identify components of reports and analysis.

Note: You may notice that there are some other data sources already defined. These are part of the Inline Service framework and are not used in this tutorial.

2.4.1 Importing the outputs for a data source

The outputs of a data source are the columns that are retrieved from the database. Outputs do not have to include all the columns in the table.

1 Click Import. Import Database Table appears. You server should appear next to Server. Click Next to connect to the server. Select Table or View appears.

2 Select the SDDS Data Source and the Table Name, `CrossSellCustomers`. This table was created and populated by the default standard installation.

3 Click Finish.

4 All of the columns of the table are imported into the Output columns table.

5 Set the input column for the data source. The input is the column that you will be matching on to retrieve the data record. In this case, we can select the column name `Id` from the Output columns table and click the right arrow (→) to move `Id` to the Input columns table. The data type is Integer.

6 Set the output columns for the data source. In the Output columns table, select and use Remove to remove all except the following columns.

Name	Type
Age	Integer
HasCreditProtection	String
Language	String
LastStatementBalance	Double
MaritalStatus	String
NumberOfChildren	Integer
Occupation	String

7 Save your work using File > Save All. If there are errors in your work, you will receive notification in the Problems View.

Note: You can use Import to import the column names and data types to the Outputs for the data source. Remove any columns you will not be using with Remove.

2.5 Creating an entity

Now that we have the data source defined, we can proceed to define a corresponding Entity. Entities are the objects that are used by the other elements in the configuration. Entities provide a level of abstraction from sources of data such as Data Sources or Informants. A single entity can have data coming from many data sources, or even computed values. For now, we will create a simple entity that maps directly to the structure of the data source.

1 In the Inline Service Explorer, locate the group Entities. Right-click and use the menu item New Entity. Enter the name `Customer` and click OK. The Entity Editor appears. Add `Customer entity` for Description.

Note: Object IDs are automatically made to conform to Java naming conventions: variables are mixed case with a lowercase first letter; classes are mixed case with an uppercase first letter. If you have spaces in your label name, they will be removed when forming the object ID.

You can use the  icon on the Inline Explorer task bar to toggle between the label of the object and its object ID.

Tip: Good descriptions for entity attributes are of particularly high importance. Make sure you add a good description for every entity.

2 Use Import to import the attributes names and data types from 'Customer Data Source'. Leave the option Build data mappings for the selected data source checked.

3 In the column Default Value of the Definition tab, click to get an insertion point and add a default value for each attribute. Values for String data types will be automatically surrounded by double quotes.

Name	Type	Default Value
age	Integer	35
hasCreditProtection	String	NO
language	String	English
lastStatementBalance	Double	1000
maritalStatus	String	Single
numberOfChildren	Integer	0
occupation	String	Student

2.5.1 About additional entity properties

You can modify additional settings about the attributes of an entity. For example, in more complex Inline Services, you may want to define categories of attributes. To do this, create a category element and assign it using the Category on the attribute's Properties. To view the properties of an attribute, select the attribute in the Definition tab, then right-click and choose Properties from the menu.

You may also want to indicate that an attribute should not be used for learning. For example, if you have the phone number of the customer it does not make sense to have analytics on the number, so in that case you would deselect Use for Analysis.

The Show in Decision Center option is used to control whether the attribute is visible in Decision Center. This is useful when an attribute has only technical meaning and no direct or interesting business meaning.

2.5.2 Adding an entity key

In order to fully map the entity object to the data source, we need an entity attribute to map to the key value of the data source and complete the mapping.

1 On the Definition tab of the Customer Entity, use Add Key to add a key attribute. Add Key appears. Enter `customerId`, add a description for the key value, change the data type to `Integer`, and click OK.

2 Save your work using File > Save All. You may see several errors in the Problems View – this is expected because the mapping definition of the Customer entity attributes to its data source is incomplete. Proceed to the next section in order to complete the mapping definition.

2.6 About the Session entity

The Session is the root of runtime data for a unit of a process. Data is kept in memory during the duration of the session. In order to track data about an Entity, we associate it with the Session entity that is part of the Oracle RTD framework. To associate the Entity to the session, make it an attribute of the Session entity. A key is chosen for the session. When a unique instance of that key is detected, the session begins.

As an example, consider a call center process being tracked by Oracle RTD. The Session contains entities that represent the Caller and the Agent. For the duration of the session (in other words, the call in this case) the data defined by those entities and the interaction between them is kept in memory and available for analysis and decision making.

2.6.1 Adding an attribute to the Session entity

1. In the Inline Service Explorer, double-click **Session** under **Entities**.
2. From the **Definition** tab, use **Add Attribute**. Enter an attribute name, `customer`, then add a **Description**. Note that the initial data type is type String. We'll change this in the next step.
3. From **Data type**, scroll down to **Other** and select it. A **Type** selection dialog appears. Under **Entity Types**, choose `Customer`. Click **OK**.

2.6.2 Creating a session key

1. In **Session Keys from Dependent Entity**, click **Select**.
2. Expand the tree to see the keys of all entities associated with the Session. Expand **customer** and select **customerId** as a session key by checking the box. Click **OK**.

Tip: Oracle RTD supports multiple session keys to enable the tracking of a session when different systems are sending Informants and Advisors to the same Inline Service. In this tutorial and in many real installations only one session key is needed.

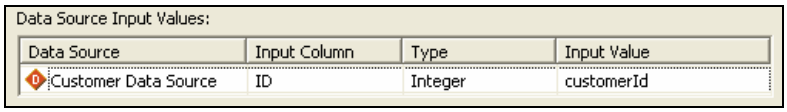
2.6.3 Mapping the entity to the data source

We associate the Customer entity with the Customer Data Source through mappings defined in the Entity object editor. Our mapping of the Customer entity's attributes to the Customer Data Source *output* columns was automatically done when the attributes were imported from the Customer Data Source – see section 2.5. We need to now map the *input* column value for the Customer Data Source in the Customer entity.

Open the Customer Entity and select the **Mapping** tab. Entities editors are identified by an 'E' icon 

.

1. Since we used Import, the Customer Data Source attributes are already mapped to the Customer entity attributes. For attribute 'Age', the **Source** value should look like `Customer Data Source / Age` (or `Customer Data Source.Age` if the Show Object ID icon is selected). If you had added additional attributes beyond the import, they are mapped by clicking  under **Source** and locating the data source attribute.
2. We need to identify the input column values for each Data Source (in this case, Customer Data Source) in the **Attributes** table. The input columns for the data source are the identifier (the 'where' clause in a SQL select statement) by which records are retrieved. In the Customer Data Source, we had only one input column, `Id`, thus in this **Mapping** tab, we will see only one entry in the **Data Source Input Values** table, located below the **Attributes** table. In the **Input Value** cell of this entry, click on  to bring up an Edit Value dialog.
3. For this inline service, we will select the Customer entity's key. Choose **Attribute or Variable**. Expand **Customer**, choose **customerId**, then click **OK**.



4. Save the Inline Service using **File** > **Save All**.

2.7 Creating an Informant

Informants are a type of Integration Point that can send a message to the Real-Time Decision Server containing information about a specific unit in a process.

To test the Inline Service at this stage, we will create a testing Informant that prints out the age of the customer. To view the actual printed statement, we will need to deploy the Inline Service to the Real-Time Decision Server and then call the Informant.

If we get a number back, we will know that the entity, mapping, and data source are working.

2.7.1 Adding an Informant

1. In the Inline Service Explorer, go to **Integration Points** and then select **Informants**. Right-click and select **New Informant** from the menu. Enter an object name, `Testing`, then OK.
2. In the **Testing** Editor, add a description under **Description**.
3. Click **Advanced** next to Description. Deselect **Show in Decision Center**. This will make this Informant invisible to the business users of the Decision Center. Click OK.

2.7.2 Adding testing logic to the Informant

On the 'Testing' Informant Editor, select the **Request** tab. Informants are identified by an 'I' icon 

.

1. To add a session key, click **Select** under **Session Keys**. Choose **customerId** from **Customer**. Click OK.

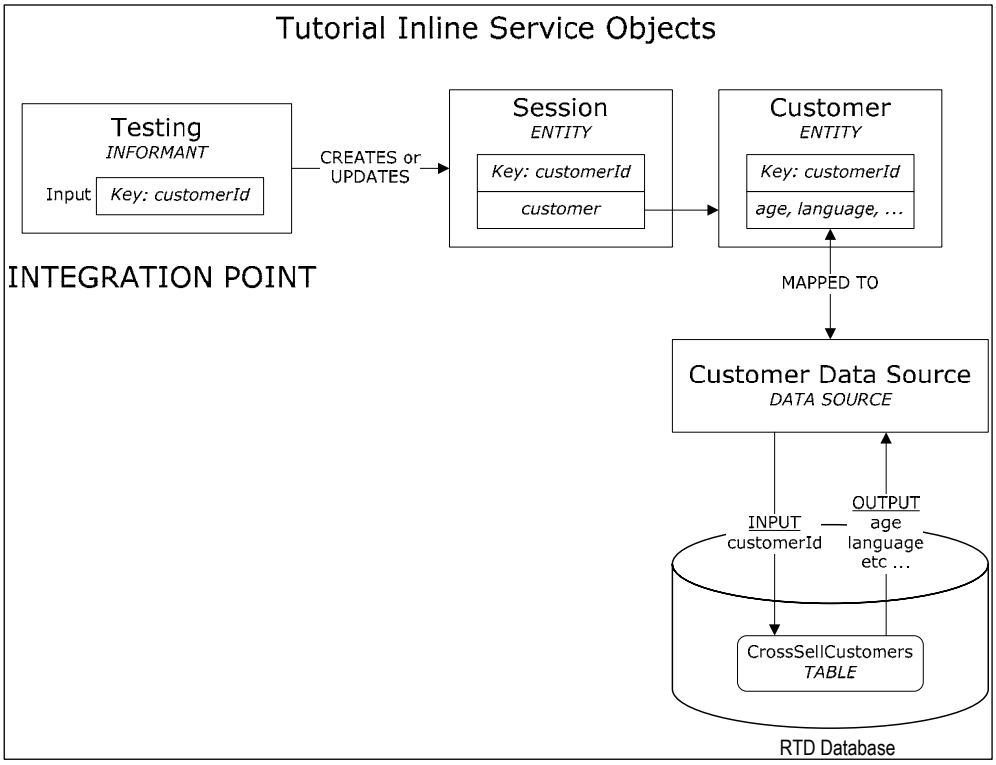
> **Note:** When you configure an entity in Decision Studio, a class is generated. The generated classes have a property, getter, and setter for each attribute.

2. Choose the **Logic** tab. Under **Logic** add the following scriptlet:

```
logInfo("Customer age = " + session().getCustomer().getAge() );
```

The logInfo call allows us to output information to the **Log** sub tab of the **Test view** and also the server log file (usually in *RTD_HOME*\log). We will use the session to access the Customer object and get the contents of the age attribute.

3. Now we should be ready to deploy. Save the configuration using **File** > **Save All**.
4. The following diagram shows how the 'Testing' Informant will access customer data when the Informant is called and a session created.



2.8 Testing the Inline Service

To test the Inline Service, we deploy it, call the Informant with test data, and use the Test View to observe the results. Because Informants do not return value to their callers, the results will be seen in the Log tab of Test View.

2.8.1 Deploying the Inline Service for testing

1 Click the Deploy icon  on the taskbar to deploy the Inline Service.

Note: You can also use the menu item Project > Deploy to deploy your Inline Service.

2 Use the Select button to select the server where you want to deploy. Deploy to the location of your Real-Time Decision Server. This defaults to localhost, as does the default configuration of the installation. Use the drop-down list to select a deployment state, `Development`. Check Terminate Active Sessions (used for testing). Click Deploy.

Deployment takes anywhere from about 10 seconds to a few minutes. A message 'Tutorial deployed successfully' will appear below the Inline Service Explorer when deployment is complete.

Note: The reason we terminate active sessions is that we want to make sure we are testing against the latest deployed Inline Service. If there were active sessions and we used the same session id (in this case, it is also the customerId), testing of the Informant would be against an earlier version of the deployed Inline Service. If we terminate the currently active sessions, then we are guaranteed to be testing against the latest deployed Inline Service, regardless of the session id used.

3 In the Test View at the bottom of the screen, select Testing as the Integration Point to test.

Enter a value for customerId by typing 7 in the field. Click Send  .

4 Select the Log tab within Test View to see the results. Every printout coming from a `logInfo` command will be printed out with a timestamp.

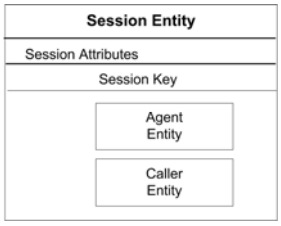
Your results should look similar to this:

```
11:53:54,102 Customer age = 38
```

2.9 Adding functionality

We will now create an entity to hold information specific to the call. This is contextual information about the nature of the interaction with the customer. The data in this entity will come from Informants or be computed, but it will not be retrieved from any database.

First we create an entity to represent a call, then an Informant that gathers data from calls. Choices are created as the targets of our analysis of the calls. In our case we are interested in focusing our analysis on the reasons for the calls.



Using this entity, we will explore the factors related to the reasons for calls, like the call lengths for each call reason, the most likely customer characteristics for these calls, and so on. In order to gather and analyze the call reasons gathered by the Informant, a self-learning analytical model will be added and reports will be displayed in Decision Center.

2.9.1 Creating a call entity

1 In the Inline Service Explorer, select the group Entities. Right-click and use the menu to New Entity. Enter the object name `Call` and click OK.

2 For each attribute listed in the table below, do the following:

- On the Definition tab of the Entity Editor click Add Attribute. Add Attribute appears. Enter the values from the table and click OK.
- Click in Type. Choose the proper data type for each attribute using the pull down.

Name	Type
agent	String
length	Integer

Name	Type
reason code	Integer

3 In the Inline Service Explorer double-click Session under Entities.

4 From the Definition tab, click Add Attribute. Enter an object name, `call`. Note that the default type is String. We will change the default type in the next step.

5 From Data type, scroll down to Other. From Other select Entity types and then Call as the type. Add a Description for 'call'. Click OK.

6 Save the changes to the Inline Service using File > Save All.

2.9.2 Creating the Call Begin Informant

We will now create three Informants that will be called by the CRM application: 'Call Begin', 'Service Complete', and 'Call End'. The first, 'Call Begin,' will start the session. In this Informant, we could optionally preload and cache certain session attribute values so they can be accessed more quickly later. For example, we may want to preload the customer's profile if this information will be used later on and the loading of this information is expected to be slow due to database calls or other constraints.

Note that it is not necessary to preload session attribute values as they are automatically loaded whenever they are needed. For example, when we want to print the customer's Age, as the 'Testing' Informant did in the previous section (2.7.2), the Real-Time Decision Server will automatically populate the entire session's Customer entity attribute and return the Age value. In this Tutorial Inline Service, our 'Call Begin' Informant will simply start the session, but will not pre-populate any session attribute values.

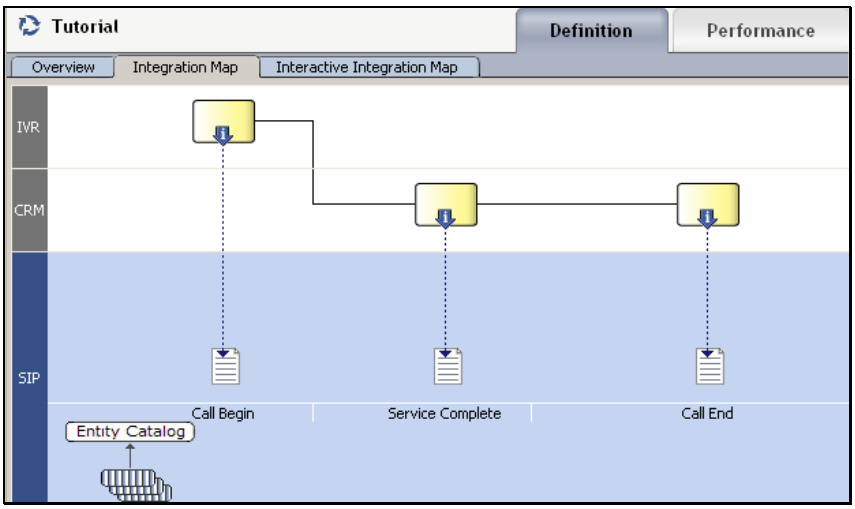
1 In the Inline Service Explorer, under Integration Points, select the group External Systems. Right-click and use the menu to New External System. Object Name appears. Name the system `IVR` and click OK. Give the element a description. Save this object.

2 In the Inline Service Explorer, select the group Informants. Right-click and use the menu to New Informant. Object Name appears. Name the Informant `Call Begin` and click OK.

3 Using the Informant Editor, enter a description for 'Call Begin'.

4 To add a session key to the 'Call Begin' Informant, click Select adjacent to Session Keys in the Request tab. Choose customerId. Click OK.

5 While still in the Request tab, choose `IVR` from the External System drop-down list and enter `1` in the Order box. Do not select Force session close. The External System and Order determine the display layout and order in Decision Center's Integration Map (see section 4.1.7). When we have finished defining the three Informants and deployed the Inline Service, the Integration Map in Decision Center will look like the following figure:



6 In the Logic tab, add the following code:

```
/*
Prepopulate customer data during start of call even though the
information may not be used until much later.  This is not explicitly
necessary since the server will automatically retrieve the information
whenever logic in the Inline Service needs it.
*/
//session().getCustomer().fill();
int CustomerID = session().getCustomer().getCustomerId();
logInfo("Integration Point - CallBegin: Start Session for customerID = "
+ CustomerID);
```

7 Save the changes to the Inline Service using File > Save All.

2.9.3 Creating the Service Complete Informant

The second Informant will report on call information such as the agent that handled the call, the length of the call, and the reason for the customer's call. This Informant is called by the CRM application when the call center agent has responded to the customer's need, or in other words, when service is complete. The data that is gathered by the Informant will populate the Call entity.

1 In the Inline Service Explorer, under Integration Points, select the group External Systems. Right-click and use the menu to New External System. Object Name appears. Name the system CRM and click OK. Give the element a description. Save this object.

2 In the Inline Service Explorer, select the group Informants. Right-click and use the menu to New Informant. Object Name appears. Name the Informant `Service Complete` and click OK.

3 Using the Informant Editor, enter a description for 'Service Complete'.

4 To add a session key to the 'Service Complete' Informant, click Select adjacent to Session Keys in the Request tab. Expand Customer and choose customerId. Click OK.

5 While still in the Request tab, choose CRM from the External System drop-down list and enter 2 in the Order box. Do not select Force session close.

6 To add the additional pieces of data to the Informant, do the following *for each incoming parameter* listed in the table below:

- On the Request tab of the Informant Editor, use the Add button. Enter the name and then select the data type using the drop-down list. Click OK.
- Under Session Attribute, click the ellipsis  to use Assignment. Follow the drop-down list to 'call' and then the call attribute that matches the incoming item.

Incoming Parameter Name	Type	Session Attribute
agent	String	call / agent (or call.agent if the Show Object ID icon is selected)
length	Integer	call / length (or call.length if the Show Object ID icon is selected
reason code	Integer	call / reason code (or call.reason code if the Show Object ID icon is selected)

7 In the Logic tab, add the following code:

```
logInfo("Integration Point - Service Complete");
logInfo("  Reason Code: " + session().getCall().getReasonCode());
logInfo("  Agent: " + session().getCall().getAgent());
logInfo("  Call Length: " + session().getCall().getLength());
```

8 Save the changes to the Inline Service using File > Save All.

2.9.4 Creating the Call End Informant

The third Informant will close the session and could be the last Informant called by the CRM application. In this Tutorial Inline Service, we will only use this Informant to close the session, but in a real system, you might perform additional processing and/or trigger learning for a model.

1 In the Inline Service Explorer, select the group Informants. Right-click and use the menu to New Informant. Object Name appears. Name the Informant `Call End` and click OK.

2 Using the Informant Editor, enter a description for 'Call End'.

3 To add a session key to the 'Call End' Informant, click Select adjacent to Session Keys in the Request tab. Expand Customer and choose customerId. Click OK.

4 While still in the Request tab, choose CRM from the External System drop-down list and enter 5 in the Order box. The reason we set the Order to 5 is because we will add two more integration points (an Advisor and another Informant) later in this tutorial.

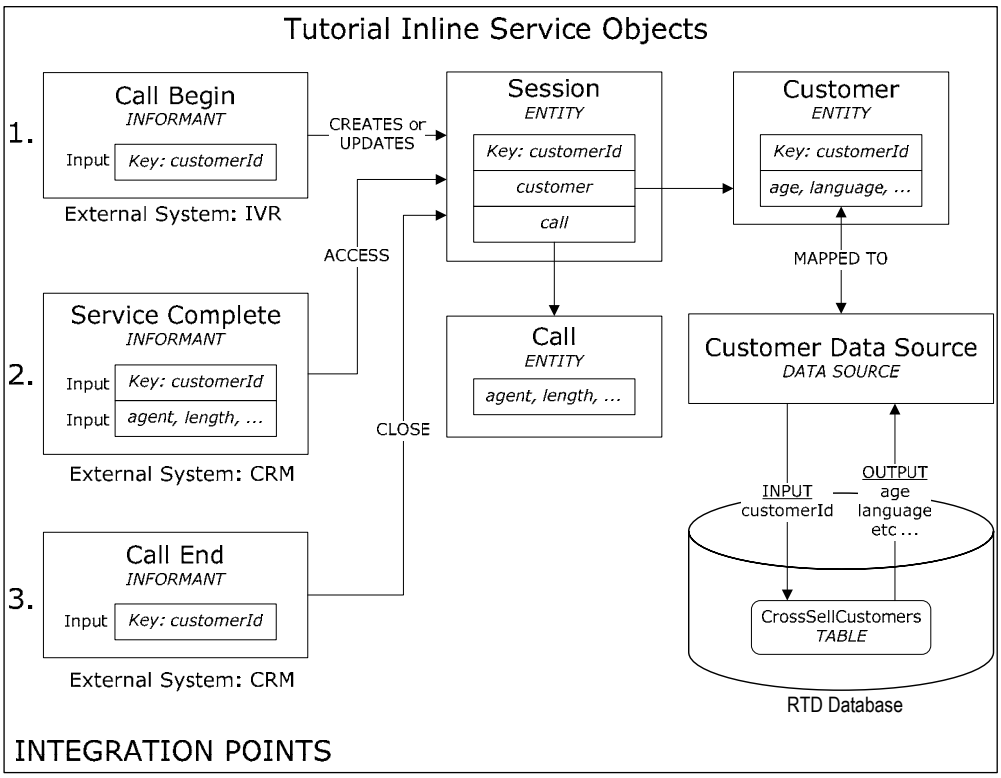
5 Make sure the option Force session close is selected. Choosing this option will explicitly close the session once the Informant has been called and its logic processed. Note that if we do not explicitly close a session, the session will automatically close after some period of time – the default is 30 minutes and can be changed using JConsole.

6 In the Logic tab, add the following code:

```
logInfo("Integration Point - CallEnd");
logInfo("**************************");
```

7 Save the changes to the Inline Service using File > Save All.

8 The following diagram shows how the three Informants access and update the same session.



2.9.5 Testing the Informants

We will now test a simple scenario where three Informants you just created are called, corresponding to 1) start of a call, 2) service completion, and 3) end of the call. We will use the Test View to call the Informants and view the log messages we had placed in the logic portions of the Informants.

1 Deploy to the server. Use Deploy  on the taskbar to deploy the Tutorial Inline Service. Remember to select Terminate Active Sessions (used for testing).

2 In the Test View, located in the bottom portion of Decision Studio, select the Integration Point `Call Begin`. For the request input `customerId`, enter an integer value, say 7. Click Send  to send the request to the server. In the Log tab within the Test View, you should see a message indicating that the 'Call Begin' integration point was called.

Repeat the process for the other two integration points, 'Service Complete' and 'Call End,' in order and with the input values as shown in the following table. Examples of what you should see in the Log tab is also shown in this table:

Integration Point	Request Inputs	Log tab
Call Begin	customerId: 7	`09:15:41,753 Integration Point - CallBegin: Start Session for customerID = 7`

Service Complete	customerId: 7
	agent: John
	length: 21
	reason code: 18
		`09:17:51,845 Integration Point - Service Complete`
		`09:17:51,845 Reason Code: 18`
		`09:17:51,845 Agent: John`
		`09:17:51,845 Call Length: 21`

Call End	customerId: 7	`09:20:17,342 Integration Point - CallEnd` `09:20:17,342 **********`

3 Note that you could have called the Informants in any order. The 'Call Begin' Informant is not needed to start a session and 'Call End' is not needed to end the session. If we had only

called the 'Service Complete' Informant, the session would still be started correctly and the response would have been the same, although the session would then remain open. We are working with three Informants in this tutorial to demonstrate the different locations of a call center process that could make calls to the Real-Time Decision Server.

Tip: Troubleshooting

If there were errors in compilation, a dialog in Decision Studio will show these errors in the Problems View. Double clicking the error will take you to the editor of the element that has the error.

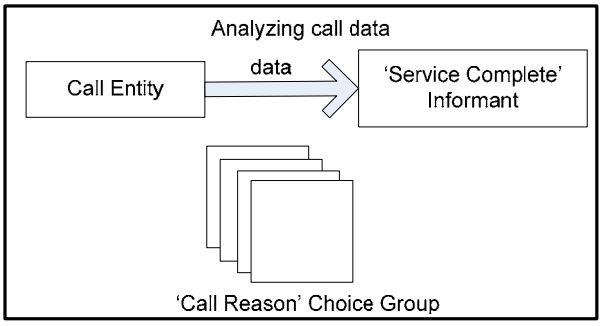
Make sure that the server with which you are communicating is "localhost." Decision Studio will remember in the drop-downs the values previously entered, and the default may not be 'localhost'.

2.10 Analyze Call Reasons

In the previous sections, we created three Informants, the second of which – 'Service Complete' – sends call information from the CRM application to the Real-Time Decision Server. One of the pieces of call information is the call reason – why the customer called. In this section, we will analyze the call reasons registered through the use of choices and a model. The objective is to able to view basic reports on call reasons – how many of each reason were recorded and how/if there were correlations between each call reason and session attributes.

2.10.1 About using choices for analysis

Choices are used to create targets for analysis. In our case, we are first interested in focusing our analysis on the reasons for the calls. We will first create a choice group for the call reasons. Then, we will define an attribute for this choice group – 'code'. The individual choices created within this group will then inherit the attribute definition, although the values can differ for each choice.



2.10.2 Adding a choice group

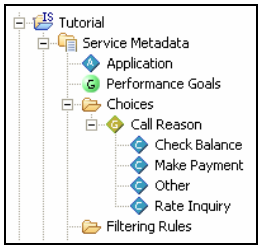
1. In the Inline Service Explorer, select the folder Choices inside Service Metadata. Right-click and select the item New Choice Group. Name the group `Call Reason` and click OK. Add a description.
2. In the Choice Group Editor for 'Call Reason', in the Choice Attributes tab, click Add next to the Attributes table. Name it `code`. Select data type `Integer`. Select Overridable. Add the description `Choice codes`.

Note: We made this a choice attribute as opposed to a group attribute. The difference between the two is that choice attributes are meant to be given values for each of the choices in the hierarchy, while group attributes are only given to the current group.

3 To create choices underneath the group, right-click the choice group 'Call Reason' in the Inline Service Explorer and select New Choice. Add a Choice, `Check Balance`.

Repeat for the following choices: `Make Payment`, `Rate Inquiry` and `Other`. Add a description for each.

4 In the Inline Service Explorer, under Choices, expand the Call Reason group to show the choices.



5 For *each of the four choices*:

Select the Choice in the Inline Service Explorer. In the Editor for that choice, under the Attribute Values tab, for the attribute 'code', set the Attribute Value as shown in the following table:

Choice	Attribute Value
Check Balance	17
Make Payment	18
Other	20
Rate Inquiry	19

6 Save the changes to the Inline Service using File > Save All.

2.10.3 About the analytical model

A self-learning analytical Model is created to perform the automatic analysis of the reasons for calls. This model will track the reason for each call and correlate all the session attributes with these outcomes. Decision Center uses this model to build reports and send alerts.

2.10.4 Adding an analytical model

1 In the Inline Service Explorer, select the group Models. Right-click and use the menu to New Choice Model. Name the model `Reason Analysis` and click OK. Make sure to use a Choice Model, and not a Choice Event Model.

2 Deselect Use for prediction.

3 To indicate that the target of analysis is the 'choice' model attribute, select the Choice tab and choose `Call Reason` from the Choice Group drop-down list.

4 On the Editor, on the Learn Location tab, select On Integration Point.

5 Use Select to choose 'Service Complete' from the list.

6 Save the changes to the Inline Service using File > Save All.

2.10.5 Adding logic for selecting choices

When the 'Service Complete' Informant is received, we need to select the choice that represents the corresponding reason for the call. We will do so by adding reasons to the model's choice array using the method of the Choice Model `addToChoice`.

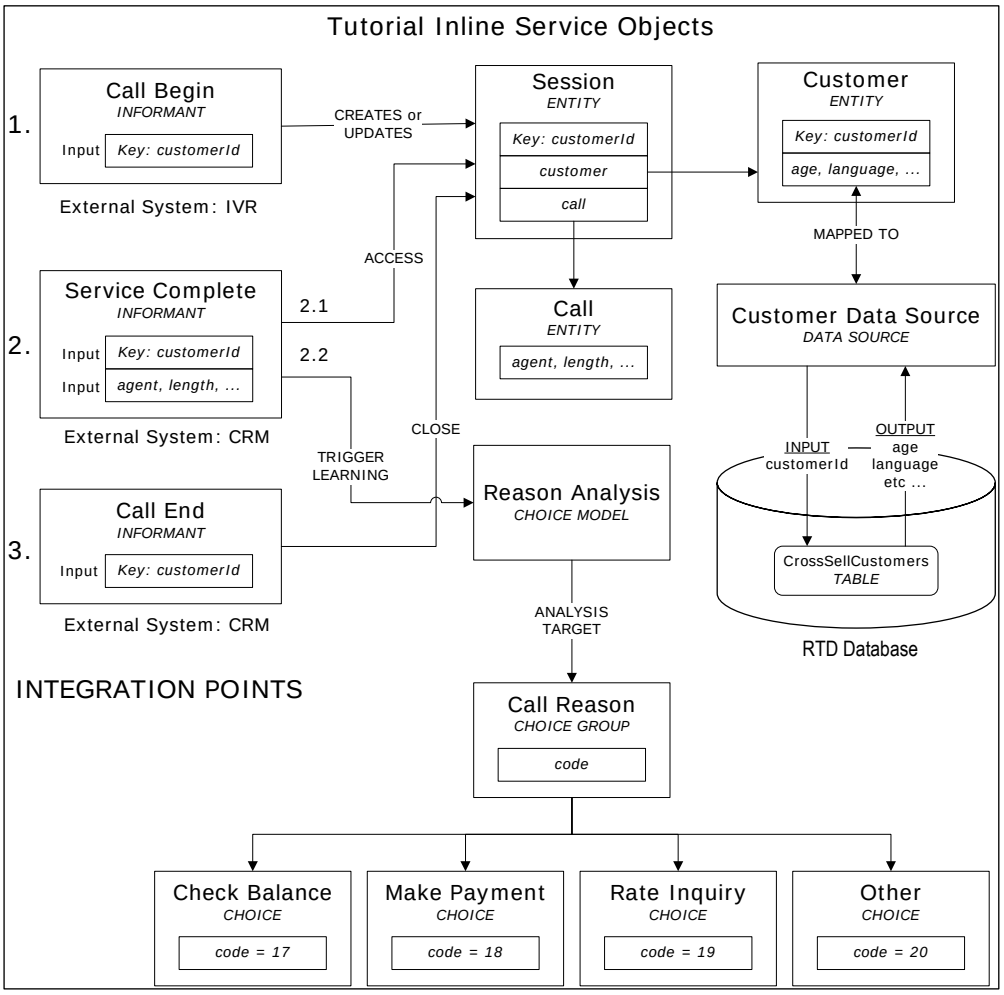
1 In the Inline Service Explorer, expand Integration Points. Under Informant select the 'Service Complete' Informant.

2 Select the Logic tab and enter the following logic. This adds the Object ID of the Choice that represents the reason for call to the model.

```
logInfo ("Integration Point – Service Complete. ");
logInfo ("  Reason Code: " + session().getCall().getReasonCode());
logInfo ("  Agent: " + session().getCall().getAgent());
logInfo ("  Length: " + session().getCall().getLength());


int code=session().getCall().getReasonCode();
switch (code) {
     case 17:
            ReasonAnalysis.addToChoice("CheckBalance");
            logInfo ("  CheckBalance was added to the model");
        break;
     case 18:
            ReasonAnalysis.addToChoice("MakePayment");
            logInfo ("  MakePayment was added to the model");
            break;
     case 19:
            ReasonAnalysis.addToChoice("RateInquiry");
            logInfo ("  RateInquiry was added to the model");
        break;
      default:
            ReasonAnalysis.addToChoice("Other");
            logInfo ("  Other was added to the model");
            break;
}
```

3 Save the configuration using File > Save All.

4 The following diagram shows how the model 'Reason Analysis' is updated when the Informant 'Service Complete' is called.



2.10.6 Testing it all together

1 Deploy the configuration to the server. Make sure there are no errors in deployment or compilation.

2 Use the Test view to test the Integration Point. Select 'Service Complete' and set values for the different arguments: customerId = 7, agent = John, length = 21, and reasonCode = 18.

Click Send. You should see results similar to the following:

```
13:57:29,794 Integration Point - Service Complete
13:57:29,794  Reason Code: 18
13:57:29,794  Agent: John
13:57:29,794  Call Length: 21
13:57:29,794  MakePayment was added to the ReasonAnalysis model
```

When this Informant with the shown input values is called, the Call entity, which is an attribute of the Session, is populated with information about the agent, length of call, and reason code. The Informant logic then determines that since the reason code was 18, then the choice 'Make Payment' will be added to the 'Reason Analysis' model. That is, the count for choice 'Make Payment' will have been increased by one. Along with the count, the model also tracks all of the session attributes and correlation with the choices.

Change the values and test a few times to see that the correct Object ID is being added to the model for other reason codes.

Section 3: Simulating Load for Inline Services

This section of the tutorial contains step-by-step instructions for utilizing the Load Generator to simulate the run-time operation of the system. In general, the Load Generator is used in three situations:

1. Performance: To characterize the performance of the system under load, measuring response time and making sure the whole system, including back-end data feeds, can cope with the expected load.

2. Initialize: To initialize the learnings and statistics with some significant data for demonstration purposes.

3. Process: To process off-line batch sources – for example, when exploring previously collected data at the initial stages of a project.

3.1 Performance under load

To evaluate performance under load, we will create a load-simulator script that calls the integration points defined in the Inline Service: 'Begin Call', 'Service Complete', and 'Call End'. The script will call this series of three integration points multiple times, each time with different customer id's, call reason codes, agent names, and call lengths. The 'Reason Analysis' model will learn on each of these iterations and we will be able to see the analysis results in Decision Center reports.

For this tutorial, we can think of Load Generator as simulating the CRM application making multiple iterations of integration point calls to the Real-Time Decision Server. The Load Generator Script (saved as an xml file) we will create will contain the definition of this simulation.

Note: When defining the Load Generator script, all references to inline service objects must be in the form of object IDs, not labels. To view the object IDs in Studio, use the object ID toggle icon  on the Inline Service Explorer taskbar. For example, the ID for the Informant 'Service Complete' is 'ServiceComplete', the id for the Informant parameter 'reason code' is 'reasonCode' and so forth.

3.1.1 Creating the Load Generator script

1 Open Load Generator by running *RTD_HOME*`\scripts\loadgen.cmd`. Then, click Create a new Load Generator script.

2 You can press F1 to read the online help for this tool and explanations for the parameters that are not explained in this tutorial.

3 Select the General tab and enter the following parameters:

Parameter	Explanation	Value
Client ConfigurationFile	A properties file that indicates the protocol to be used to communicate with the server, what server to talk to and through what port. The default is to communicate using HTTP to the local server using port 8080. The default file is suitable for our needs.	*RTD_HOME*`\client\clientHttpEndPoints.properties`
Graph refresh Interval in Seconds	This parameter only affects the user interface. It determines the refresh rate for the UI graph and counters. The default is to refresh every 2 seconds.	2

Parameter	Explanation	Value
Inline Service	This is the name we gave the Inline Service we created in the previous section.	Tutorial
Random Number Generator Seed	The seed used to generate random numbers. Default is -1.	-1
Think Time	Think Time is the time in between transactions. In a session oriented load simulation you would give different numbers here. For this tutorial we will explore the maximum throughput, sending as many sessions as possible. Values for Think Time can be fixed or a range of values.	Fixed Global Think Time
Constant	A fixed constant for think time in between transactions.	0
Number of concurrent scripts to run	This is the number of sessions active at any given point, running in parallel. In this case we will just run one session at a time.	1
Maximum number of scripts to run	The total number of session that will be created. Load Generator will stop sending events after this number has been reached.	2000
Enable Logging	Checkbox to enable/disable loadgen counters log. This log maintains a history of loadgen performance data, including the number of requests sent by Load Generator, number of errors, the average and peak response times of a request, etc. If deselected, the remaining three logging parameters (Append to Existing File, Log File, Logging Interval in Seconds) are ignored.	Unchecked
Append to Existing File	Checkbox to indicate whether to overwrite or append to an existing log file each time a loadgen script is run.	Unchecked
Log File	File path to an `ascii` file. This is the location where the Load Generator log will be written, in tab-delimited format.	*RTD_HOME*`\log\loadgen.csv`
Logging Interval in Seconds	This parameter only affects the Load Generator log. It determines the interval for writing to the log. The default is 10 seconds.	10

In the path names, make sure to replace *RTD_HOME* with the path where you installed Real-Time Decision Server (for example, `C:\OracleBI\RTD`).

Note that parameters related to sessions cannot be changed in the middle of execution. More precisely, they can be changed, but their changes will not affect execution until the Load Generator script is stopped and restarted.

4 Save the configuration. It is customary to save Load Generator Script in a folder named `etc` within the inline service project folder. If you had created the Tutorial inline service in the default workspace, the path would be similar to: `C:\Documents and Settings\Win_User\Oracle RTD Studio\Tutorial\etc`. Name the script (an xml file) anything you like (for example, `TutorialLoadgen.xml`).

5 To define the values for the parameters to the Integration Point, click the Variables tab. Variables allow an Integration Point's parameter values to be drawn from different sources.

Note: It is possible that not all the tree is visible on the left. To make it all visible, you can drag the bar dividing the two areas.

6 Right-click Script and select Add Variable. Name it `var_customerId`. In Contents, select Integer Range from 1 to 2000, sequential. This definition will create a variable that is computed once per session and goes from 1 to 2000 sequentially, that is, the first session will have var_customerId = 1 and the last one will be 2000. Right-click Script and select Add Variable three more times for a total of four variables:

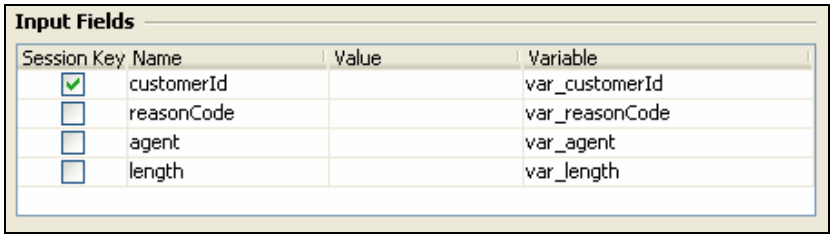
Parameter	Content type	Variable Value
var_customerId	Integer Range	Minimum = 1, Maximum = 2000, Access type = Sequential
var_reasonCode	Integer Range	Minimum = 17, Maximum = 20, Access type = Random
var_agent	String Array	To add a string to the array, right-click on the table area and select Add Item. Then select (double-click) the newly created row to get a cursor and type the name to be used. Press the Enter key to register the value for that row. Add a few sample values of agent names (for example, John, Peter, Mary, and Sara).
var_length	Integer Range	Minimum = 75, Maximum = 567, Access type = Sequential. This will be used as the length of the call in seconds.

7 Select the Edit Script tab, then right-click the left area and select Add Action. We will add three actions, each corresponding to an integration point. We need the actions to be in the right order – 'CallBegin', 'ServiceComplete', and finally 'CallEnd'.

8 For the first action, set the type to `Message` and the Integration Point name to `CallBegin`. In Input Fields, right-click and choose Add item to add an input field. Double-click in the space under Name and enter the value `customerId`; press Enter to register the new value. In the Variable column for customerId, choose `var_customerId` from the drop-down list. Select Session Key to identify this field as the session key.

9 In the Edit Script tab, right -click the left area and select Add Action. Set the type to `Message` and the Integration Point name to `ServiceComplete`. In Input Fields, right-click and chose Add item to add an input field. Set the Name to `customerId`, the Variable to `var_customerId`, and select Session Key.

10 For the action 'ServiceComplete', add three additional fields (reasonCode, agent, and length) and set their names and variable values to that shown in the following figure.



Again, the names here must match *exactly* with the incoming parameter IDs for the 'ServiceComplete' Informant as seen in Decision Studio. You can use the  icon on the Inline Service Explorer task bar in Decision Studio to toggle between the label of the object and its object ID.

11 In the Edit Script tab, right-click the left area and select Add Action. Set the type to `Message` and the Integration Point name to `CallEnd`. In Input Fields, right-click and chose Add item to add an input field. Set the Name to `customerId`, the Variable to `var_customerId`, and select Session Key.

12 Once again, save the Load Generator configuration script. Our Load Generator script now contains calls to three integration points. Make sure the order of the actions in the Edit Script tab is correct: 'CallBegin', 'ServiceComplete', and 'CallEnd'. If not in this order, right-click the actions to move items up or down. Then, save the script again.

13 Go to the Run tab and press the Play  button. Allow Load Generator to complete.

Note: There is a Pause button and a Stop button. The difference between these two is that Pause remembers the sequences and will continue from the point it was paused, whereas Stop resets everything.

Tip:

Troubleshooting

Look at the Total Errors in the Run tab. If the number is above 0, look at the server output window. There may be an indication of the problem. Common mistakes are:

1. The Inline Service has not been deployed.
2. There is a spelling or case mistake in the name of the Inline Service or the Integration Point.
3. The server is not running.

If the Total Number of Requests stays at 1 and does not grow, there may be a mistake in the definition of the loadgen script. Some things to look for:

1. In Integer Range variables, make sure the Minimum is below the Maximum.
2. Make sure that the mapping of values sent in messages to variables is correct. For example, if a variable name is changed, the mapping needs to be redone.
3. Make sure the Client Configuration file is correct.

3.1.2 Viewing analysis results in Decision Center

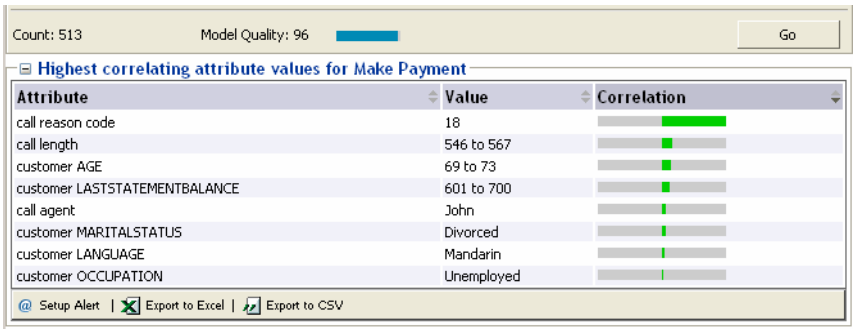
You can use the Decision Center to check what has been learned by the models after running the Load Generator.

1 Open Decision Center by opening a Web browser and going to the URL `http://server_name:8080/ui`. Log in as a user with administrative permissions (permission `0`).

2 Click Open Inline Services. The Select Inline Service window appears. Select Tutorial, then expand Call Reason and select one of the Choices, such as Make Payment. In the right

pane, navigate to the Analysis tab and then the Best-fit subtab. This report summarizes the number of times this call reason was seen, and correlations between this call reason and attribute values.

3 You will see something interesting. The call reason code has an unexpectedly strong correlation to the Make Payment.



Since we generated the call data randomly with Load Generator variables, we would not expect to have any significant correlations. In this case, however, the call reason code (sent by the 'ServiceComplete' Informant) absolutely determines the call reason (see logic written in section 2.10.5).

To remove this induced-correlation, we should exclude this attribute from being used as an input to the model. Another type of attribute we might exclude from models is a customer's full telephone number. After all, it is unlikely that correlations can be found between unique telephone numbers and the reason he/she called. On the other hand, there may be correlations between the area codes of customers and the call reasons, so this would be an attribute that we would not exclude from the model. In the next section, you will exclude the 'reason code' attribute from the model and re-run the Load Generator script.

3.1.3 Excluding the attribute

4 In Decision Studio, open the Tutorial project.

5 Expand Service Metadata > Models, then select Reason Analysis from the Inline Service Explorer.

6 Go to the Attributes tab. In the lower table, titled Excluded Attributes, click Select to choose an attribute to exclude. Expand the Session node, then expand the call entity and select reason code.

7 Save all and redeploy to the localhost server.

8 You can now re-run the Load Generator script.

If you use Decision Center now to look at the counts, you will notice that they include the events from both runs of the Load Generator. This happens because we did not reset the models between the two times we ran the Load Generator script.

3.2 Resetting the Model learnings

Use the JConsole administration tool to reset the Model learnings.

1 If you are using OC4J or WebLogic, open JConsole by running *JAVA_HOME*`\bin\jconsole.exe`. If you are using WebSphere, run the batch script you created during JConsole configuration. See *Installation and Administration of Oracle RTD* for more information about accessing JConsole.

2 Click the Remote tab. Then, enter the appropriate port number (typically `12345`) and the administrator credentials you created during installation and click Connect.

3 Click the MBean tab, then go to the OracleRTD > InlineServiceManager > Tutorial > Development > Loadable MBean.

4 Click the Operations tab, then use the deleteAllOperationalData() operation to remove all operational data, including the study, for this Inline Service.

5 To see the new results in Decision Center, run the Load Generator script again.

3.2.1 Summary of the Inline Service

We have so far created a simple but fully functional Inline Service. We did so by starting with the definition of the data environment, the data source and entity for the customer, and then the entity for the current call data. After testing the basic functionality, we created several Integration Points and a model to perform the analysis. Logic was added to determine the reasons of the customer calls and to record occurrence of the different reasons in a model for analysis purposes. We then used Load Generator to simulate requests against the Real-Time Decision Server and the Tutorial inline service. The results are then viewed in Decision Center.

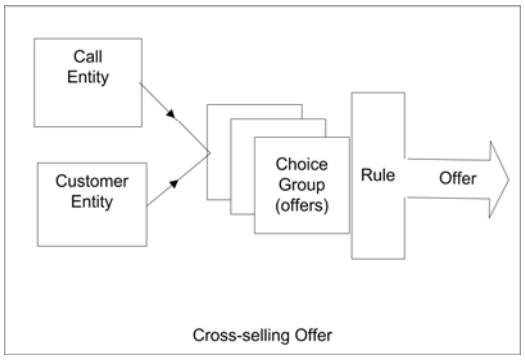
Section 4: Enhancing the Call Center Inline Service

In Section 2, we created an Inline Service that tracks and analyzes incoming data related to the reason for calls to a credit card call center. In Section 3, we used Load Generator to simulate client requests (through Informant calls) to our Inline Service.

In this section, we will enhance the Tutorial Inline Service to provide cross-selling advice to the CRM application. The process enhancement is this: after the agent has finished processing the customer's call in the normal way ('Service Complete' Informant called), the agent then takes the opportunity to present an offer to the customer. In section 5, we will track the customer's response to the offer, and then use what was learned from the responses in presenting offers to other customers.

4.1 About using choice groups and scoring to cross sell

We will create a choice group of offers that can be extended to customers calling the service center. Choice scores are based on cost in order to support our Performance Metric of minimizing cost. Next, an Advisor is created to pass that cross sell recommendation to the CRM application, so that the call center agent can extend the offer.



4.1.1 Creating an offer inventory using choice groups

1. In the Inline Service Explorer, select the group **Choices**. Right-click and choose New **Choice Group**. Name the group `Cross Selling Offer` and click **OK**.
2. Expand **Choices** and select/open the newly created group. Add a description.
3. In the **Choice Attributes** tab, click **Add** next to the **Attributes** table. Add the following attributes, making sure to select **Send to client** and **Overridable**:

Attribute name	Data type	Send to client	Overridable
Offer Description	String	√	√
URL	String	√	√
Agent Script	String	√	√

Note: These attributes are sent to the client because they are needed by the client (call center agent) to present the offer.

The attributes should be overridable because their values will be different for each actual offer. Cross Selling offers will be represented by choices in this choice group.

In a real Inline Service, we are likely to see several levels of choice groups before we get to actual offers. Each choice group provides a logical group for offers, and may have attributes or business rules that apply uniformly to a group of offers.

4 In the Inline Service Explorer, under Choices, select the Cross Selling Offer choice group and add five choices with their attributes, as shown in the table below.

For each of the choices:

- Right-click Cross Selling Offer in the Inline Service Explorer and select New Choice. Add the following choices: Credit Card, Savings Account, Life Insurance, Roth IRA, and Brokerage Account.
- In the Inline Service Explorer, under Choices, expand the Cross Selling Offer Group to show the choices.
- For each of the five choices:
  1. Select the Choice in the Inline Service Explorer. In the Editor for that choice, add a description.
  2. On the Attribute Values tab you will see the three attributes: Agent Script, Offer Description, and URL. Using Attribute Value, add the attribute values from the following table:

Choice name	Agent Script	Offer description	URL
Brokerage Account	Would you like to try our new brokerage account?	Brokerage Account offer	http://www.offer.com/offer1.html
Credit Card	Would you like to try our new credit card?	Credit Card offer	http://www.offer.com/offer2.html
Life Insurance	Would you like to try our new life insurance?	Life Insurance offer	http://www.offer.com/offer3.html
Roth IRA	Would you like to try our new Roth IRA?	Roth IRA offer	http://www.offer.com/offer4.html
Savings Account	Would you like to try our new savings account?	Savings Account offer	http://www.offer.com/offer5.html

5 Save the configuration using File > Save All.

4.1.2 Configuring Performance Goals

1. In the Inline Service Explorer, double-click the **Performance Goals** element to open the editor. Use the **Add** button to add a Performance Metric. Name the metric `Cost`. Click OK.
2. In **Optimization**, choose `Minimize` and make the metric **Required**.

> Note: If you have more than one performance metric, you must use the Normalization Factor to normalize the values. For instance, if you had another metric called "Minimize hold time" measured in seconds, the normalization factor would be how many minimized seconds are worth a dollar (revenue) to your organization.

3. Save the configuration using **File** > **Save All**.

4.1.3 Scoring the choices

Each product costs the company an average amount to maintain on a yearly basis. The cost in dollars is the score for that product.

1. In the Inline Service Explorer, under **Choices**, select and open the **Cross Selling Offer** choice group. In the **Scores** tab, click on **Select Metrics** and choose the performance metric **Cost**. This sets up the choice group with a Cost score. The actual score values will be set on a per-choice basis.
2. Score values do not have to be constants. In many cases, score for one type of customer can differ by a significant amount from another customer type. We can express such differences through the use of formulas or **scoring rules**. For example, the Cost to maintain a credit card account may be less for customers who are age 40 or under. We will define this logic in a **Scoring Rule** and then assign this rule to the Cost score for the Credit Card offer.
3. In the Inline Service Explorer, right click the folder **Scoring Rules** and select **New Scoring Rule**. Name the scoring rule `Credit Card Score`. The editor for this new rule opens.
4. Click the **Add conditional value** button  to set up a rule condition in addition to the default. A new row will appear, where the left cell is a two-sided rule and the right cell is the score value. The logic is as follows: "If the left cell evaluates to true, then the value in the right cell will be returned, otherwise use the value in the second row of the rule." Click the left side of the rule and then on the ellipsis  . An **Edit Value** dialog appears. Select **Attribute**, expand **session attributes** > **customer** and select **Age**, then click OK. Click the condition operator  , then click the lower-right corner triangle and select the less than or equal to symbol (<=). Click in the right half of the rule and type the number `40`. In the Then cell, type the number `130`. In the second row, select and type in the number `147` for the value. The full rule should look like the following:

Condition	Value
If All of the following 1. session / customer / AGE <= 40	Then 130.0
Otherwise...	The value is: 147.0

Save the Credit Card Score scoring rule. For the other offers, we will set constant values for the Cost score.

5. For each of the choices under the choice group **Cross Selling Offer**, open the **Scores** tab. In the Score column for the Cost metric, enter the values shown in the following table. To set the

Cost score for the Credit Card choice, click the ellipsis in the Score column, then select Function or rule call as the Value Source. In the Function to Call drop-down list, select Credit Card Score.

Choice	Cost Score
Brokerage Account	150
Credit Card	"Credit Card Score" scoring rule: 130 if age <=40, otherwise 147
Life Insurance	140
Roth IRA	145
Savings Account	135

Since our Performance Goal is to minimize costs, it is clear that the Savings Account offer (score =135) will be chosen unless the customer's age is 40 or below (score = 130), in which case the Credit Card offer will be chosen. In later sections of the tutorial, we will add another Performance Goal, Maximize Revenue, to see how these two competing performance metrics are optimized by the platform.

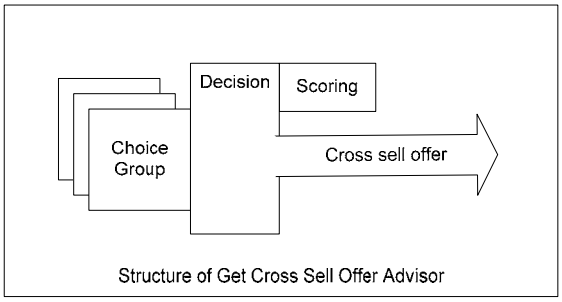
6 Save the configuration using File > Save All.

4.1.4 About Advisors

When an external system needs a decision to be made on its behalf, it calls an Advisor. Here, we create the Advisor that will send back to the CRM application an offer selected for a specific customer.

The Advisor's internal structure includes a Decision which associates it with one or more Choice Groups. These Choice Groups contain the offers that are to be made. The result of the decision is the result sent to the Advisor.

An Advisor has two decisions, one for normal processing and the other for the control group. The control group serves as a baseline to show performance gains achieved by Oracle RTD.



Structure of Get Cross Sell Offer Advisor

4.1.5 Creating the Decisions

1 In the Inline Service Explorer, select the group Decisions. Right-click and choose New Decision. Name the Decision `Select Offer` and click OK.

2 Add a description for 'Select Offer'. On the Selection Criteria tab in the Decision editor, locate Select Choices from. Use Select to select the Cross Selling Offer from the list and click OK.

3 For our control group, we will have a decision that chooses an offer randomly. Create a new Decision and name it `Random Choice`.

4 Add a description for 'Random Choice'. On the Selection Criteria tab in the Decision editor, locate Select Choices from. Use Select to select the Cross Selling Offer from the list and click OK.

5 Check the Select at random box.

Note: The Control Group acts as a baseline so that the business user can compare the results of the predictive model against the pre-existing business process. It is important to correctly define the Control Group decision to truly reflect the decision as it would have been made if Oracle RTD was not installed. For example, in a cross-selling application for a call center, if agents randomly selected an offer before Oracle RTD was introduced, then the Control Group Decision should return a random selection.

6 Save the configuration using File > Save All.

4.1.6 Creating the Advisor

1 In the Inline Service Explorer, under Integration Points, select the group Advisors. Right-click and choose New Advisor. Name the element `Get Cross Sell Offer` and click OK.

2 To add a session key to the Get Cross Sell Offer Advisor, use Select under Session Keys in the Editor and choose customerId from Customer. Click OK.

3 Under External System, select CRM. For Order, enter 3.

Recall that we had set the Order for Informant 'Service Complete' to 2. We are preparing to call the Get Cross Sell Offer Advisor after this Informant, and thus the order number 3. Note that the Order is only used in Decision Center's integration map to help graphically describe the application process; Order does not force integration points to execute in any particular sequence.

4 On the Response tab, select a Decision for both the normal processing and the control group. Select `Select Offer` for the Decision and `Random Choice` for the Control Group Decision.

5 In the Default Choices section, use Select to choose Life Insurance from the list and click OK. This will make the selected Offer the default response for this Advisor. This default will be used when there is any problem in the computation (for instance, if there is a timeout).

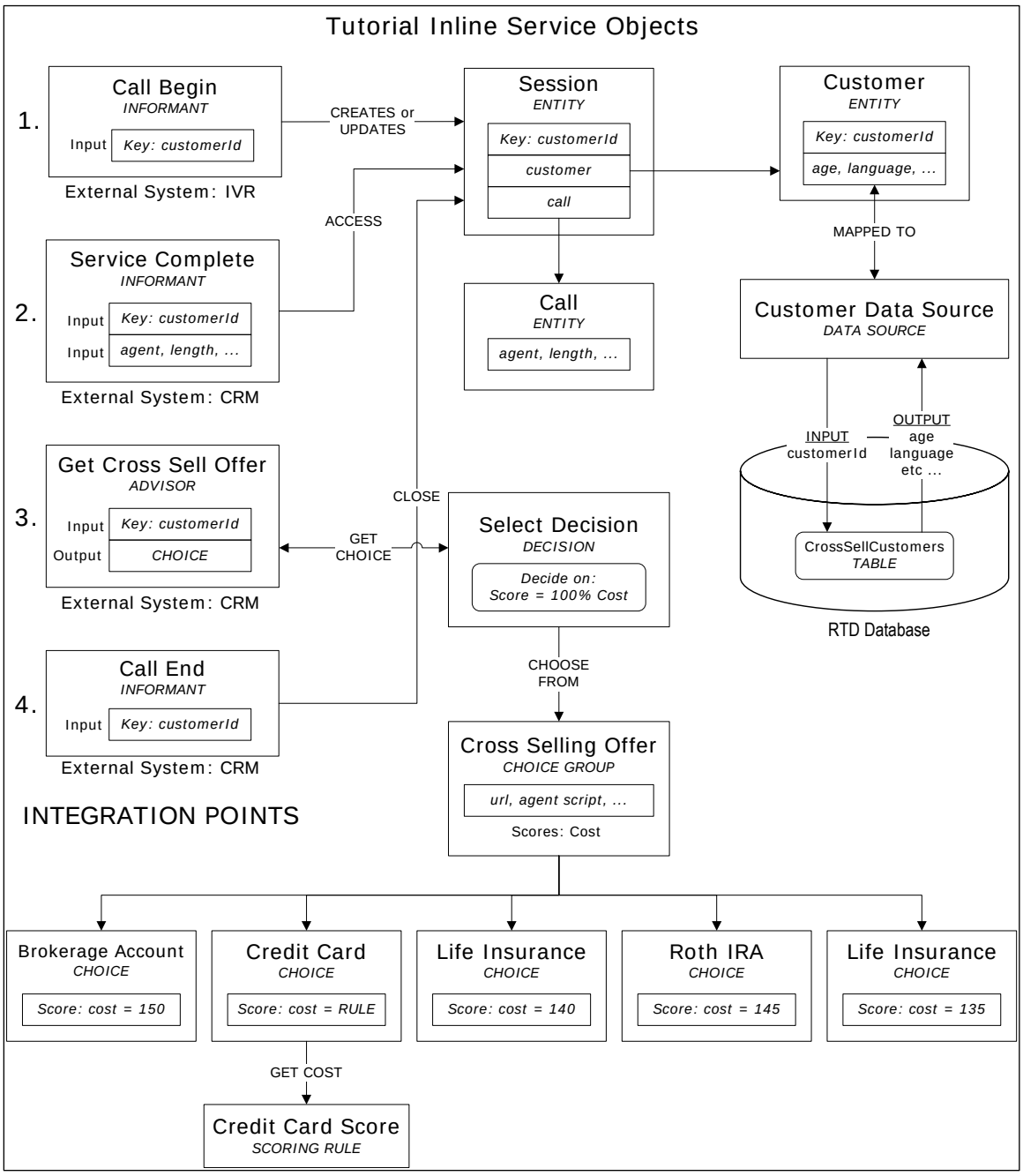
6 In the Asynchronous Logic tab, enter the following code:

```
logInfo("Integration Point - Get Cross Sell Offer");

logInfo("  Customer age = " + session().getCustomer().getAge() );
// 'choices' is array returned by the 'Select Offer' decision

if (choices.size() > 0) {

  //Get the first offer from array
  Choice offer = choices.get(0);

  logInfo("  Offer presented: '" + offer.getSDOLabel() + "'");

}
```

If we had entered the code in the Logic tab, it would have been executed before the decision was made on which offer to return, and we would not be able to print the name of the offer returned. In the above code, we print the customer's age and the presented offer name. Recall that because we are minimizing on Cost, only the offers Savings Account and Credit Card will be presented, depending on the age of the customer.

7 Save the Inline Service. Click the Deploy button. Select Terminate Active Sessions (used for testing) to remove any session that is still currently active. Deploy.

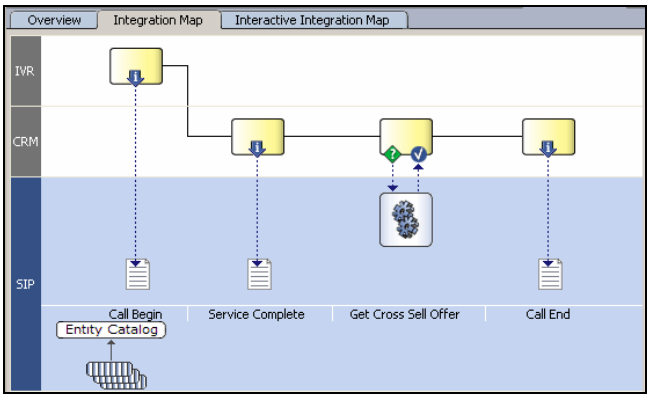
8 The following diagram shows how the Get Cross Sell Offer Advisor retrieves an offer from the Cross Selling Offer choice group, based on the performance goal Cost.



4.1.7 Viewing the integration map

1. Open Decision Center by opening a Web browser and going to the URL `http://server_name:8080/ui`. Log in using the default administrator credentials you created during installation. Real-Time Decision Server must be started for Decision Center to run.
2. Click **Open an Inline Service**.
3. Choose the **Tutorial** Inline Service.

4 On the left-hand tree, click the root node **Tutorial**. In the right pane, on the **Definition** tab, click to view the **Integration Map** subtab. You should see something similar to the following:



5 The following symbols are used on the Integration Map to indicate integration points, processing, entities and information flow.

Symbol	Significance
	Processing on the Real-Time Decision Server
	Advisor call
	Information provided to the Real-Time Decision Server
	Informant Call

4.1.8 Testing the Advisor

1 In Decision Studio, use the **Test** view to send a request integration point. Select the Service Complete Informant and fill in some values for the parameters. For example: customerId = 7, agent = John, length = 21, reason code = 18 (others: 17, 19, or 20).

2 Click **Send**  and confirm in the **Log** subtab that the message was sent. This Informant call creates a new session based on the customer id and registers the customer's call reason, agent's name, and call length.

3 Now select the **Get Cross Sell Offer** Advisor, leaving the **customerId** as it is, as we want to continue with the same session. Click **Send**.

The selected offer and its attributes are returned and displayed in the **Response** pane in the **Test** View.

In the **Log** subtab in the **Test** View, for customerId = 7, you should see something similar to:

```
00:24:40,764 Integration Point - Get Cross Sell Offer
00:24:40,764   Customer age = 38
00:24:40,764   Offer presented: 'Credit Card'
```

4 Repeat steps 1 to 3 with different values for customerId and other parameters. Notice that the Credit Card offer is returned if the customer's age is 40 or below, and the Savings Account offer is returned for all other ages. This is expected because so far, we have only optimized on the Cost performance metric, thus the lowest cost offer is either Savings Account or Credit Card, depending on the customer's age (see the Credit Card Score scoring rule in section 4.1.3).

5 In the Trace subtab of the Test view, you will find a description of the sequence taken to arrive at the offer, from determining which offers are eligible to computing scores for each offer, and finally choosing the offer that met the performance goal (minimize Cost).

Section 5: Closing the Feedback Loop

In the previous section, we added an Advisor that returns an offer to the CRM application so the call center agent can present it to the customer. Once presented to the customer, we want to track whether the customer has accepted the offer and thus close the loop on the offer presentation/acceptance process. The feedback loop can be closed in different ways and at different times. It is not unusual to know the results only days or weeks after a decision or offer is made. Even then, in many cases, only the positive result is seen, but not the negative. Feedback can come directly from customers, from the agents handling the call, from operational systems that handle service, fulfillment or billing, or even from batch processes.

The way the feedback loop is closed with an Inline Service is by notifying the Real-Time Decision Server through the use of Informants.

5.1 About the use of events to track success

In most cases, there are different events in the lifetime of an offer that are interesting from the point of view of tracking success. For example, the events in the life of a credit card offer may be:

- Offer presented
- Customer showed interest
- Applied for the card
- Received the card
- Used the card

An argument could be made that only when the customer uses the credit card is there any real success. The goal is to bring more customers that not only show interest, apply and get the card, but for them to also use it, as card usage is what brings revenue to the company.

Usually, it is easier to track events that are closer to the presentation of the offer. For example, if an offer is presented in the call center by an agent, the agent can gauge the degree of interest shown by the customer. For an offer presented in a Web site, a click-through may be the indicator of interest.

Events further down the life of an offer may be much more difficult to track and decide on the right offer. Therefore, it is not unusual to begin a project having only the immediate feedback loop closed, and adding events further down the road as the system matures. Nevertheless, even with only immediate feedback, Oracle RTD can provide significant lift in marketing decisions.

5.1.1 About defining events in choice groups

Events are defined at the Choice Group level. While they can be defined at any level in the hierarchy, they are usually found at the highest level, close to the root.

We will define two events, one to represent the fact that an offer was presented to the customer, and the other to represent the fact that the offer was accepted. For the tutorial, we will assume that every offer selected as a result of the Advisor will be presented, and that the acceptance of offers is known immediately.

5.1.2 Defining events in a choice group

1. In the Inline Service Explorer, select the Choice Group Cross Selling Offer.
2. Select the Choice Events tab. Use Add to add two events, one named `Presented` and the second named `Accepted`. Note that these event names are simply labels and do not correspond to any internal state of the offer. These events will be used in a Choice Event Model (described in the next section), where these event names will take on meaning.
3. For each event, set the Statistic Collector to `Choice Event Statistic Collector` using the drop-down list. This is the default statistics collector. This will provide for statistics gathering regarding each of the events.

4 Make sure that the Event History (days) is set to `Session Duration`.

This setting indicates that the system will remember the events only for the duration of the session. If we were interested in remembering offer events for a few days or weeks, we would set it up here.

5 Leave the Value Attribute empty.

This is used for the automatic computation of the event. In this tutorial, we will be causing the events to be recorded from the logic of the feedback Informant.

6 Save All.

5.1.3 About the choice event model

Events are defined and are ready to have statistics tracked. In addition to tracking statistics, we are interested in having a self-learning-model learn about the correlations between the characteristics of the customers, calls and agents, and the success or failure of offers. This knowledge is useful in two ways:

- It is useful for providing insight and understanding to the marketing and operations people.
- It is useful to provide automatic predictions of the best offer to present in each situation.

In this tutorial we will show both usages.

5.1.4 Defining a choice event model

1 In the Inline Service Explorer, select Models, then right-click and choose New Choice Event Model. Call the new model `Offer Acceptance Predictor` and click OK.

2 In the Editor, deselect Default time window and set it to a week.

3 Under Choice Group, choose `Cross Selling Offer`.

This is the group at the top of the choice hierarchy for which we will track offer acceptance using this model.

4 Under Base Event, choose `Presented`. Recall that you had defined these event names in the choice group in section 5.1.2.

This is the event from which we want to measure the success. We want to track whether an offer was *accepted* after it was *presented*.

5 In Positive Outcome Events, use Select to choose Accepted from the list and click OK. For the tutorial, this is the only positive outcome. If more events were being tracked, we would add them here also.

6 Optionally, you may change the labels to be more offer-centric.

5.1.5 Additional model settings

There are other settings that are useful for Choice Event Models. Using the Attributes tab, you see there are two main settings: partitioning attributes and excluded attributes.

5.1.5.1 Partitioning attributes

Partitioning attributes are used to divide the model along strong lines that make a big difference. For example, the same offer is likely to have quite different acceptance profiles when presented in the Web or the call center, thus the presentation channel can be set as a partitioning attribute.

You can have more than one partitioning attribute, but you should be aware that there may be memory usage implications. Each partitioning attribute multiplies the number of models by the number of values it has. For example, a model having one partitioning attribute with three possible values and another with four possible values will use twelve times the memory used by a non-partitioned model. Nevertheless, do use partitioning attributes when it makes sense to do so, as it can significantly improve the predictive and descriptive capabilities of the model.

5.1.5.2 Excluded Attributes

Sometimes, it does not make sense to have an attribute be an input to a model. For example, we saw in the Reason Analysis model (section 3.1.2) that having the reason code as an input created a correlation between reason code and the call reason choices. This relationship was entirely expected due to the logic we had written in section 2.10.5. Since this correlation was artificial and did not offer insight, we excluded reason code from the model.

It should be noted that the reason code could be an important factor for other models and should not be excluded. For example, in the Offer Acceptance Predictor model, we would be very interested to see if offer acceptance was correlated with the reason code.

5.1.5.3 Learn Location

The Learn Location tab has the settings for the location in the process where model learning happens. The default, On session close, is a good one for most cases. Learning on specific Integration Points may be useful when it is desired to learn from more than one state in a session.

5.1.6 About closing the loop

The choice event model is complete and it is ready to be used. In order to feed it with the right information, we need to complete the logic for closing the loop.

In order to have available which offer was extended, we will remember the offer ID in the session. This is not absolutely necessary, as the front-end client could remember that, but here we do not want to make any assumptions about the capabilities of the front end. We will just use a simple String attribute to remember the offer; in more complex cases we would use an array to remember many choices.

5.1.7 Remembering the extended offer

1. In the Inline Service Explorer, select the Session entity under Entities.
2. Use Add Attribute to add an attribute named `Offer Extended`.
3. Enter a description. Deselect Show in Decision Center and Use for Analysis. Click OK.

   We do so because for now, we will treat this as an internal variable, not to be seen by the business users.
4. In the Inline Service Explorer, select Get Cross Sell Offer under Integration Points: Advisors. Go to the Editor.
5. In the Asynchronous Logic tab, update the existing code by adding several lines to record the presented event and to set the OfferExtended session attribute with the value of the choice id. The completed code should be as follows:

```
logInfo("Integration Point - Get Cross Sell Offer");
logInfo("  Customer age = " + session().getCustomer().getAge() );
// 'choices' is array returned by the 'Select Offer' decision
if (choices.size() > 0) {
  //Get the first offer from array
  Choice offer = choices.get(0);
  //For the selected offer, record that it has been 'presented'
  offer.recordEvent("presented");
  //Set the session attribute 'OfferExtended' with the offer's ID.
  session().setOfferExtended(offer.getSDOId());
  logInfo("  Offer presented: '" + offer.getSDOLabel() + "'");
}
```

This will assign the SDOId of the selected choice to the 'OfferExtended' attribute of the session entity. The SDOId is a unique identifier. Every object in an Oracle RTD configuration has a unique SDOId. It will also record the Presented event for the selected offer. Note the event name is in lowercase and

corresponds to the choice event id for 'Presented'. To see the id, go to Inline Service Explorer, expand Choices, double-click on Cross Selling Offer, click on the Choice Events tab, and click on the label/id toggle button:  .

At this point of the decision, the session knows which offer has been chosen to be presented to the customer by the call center agent (through the Get Cross Sell Offer Advisor). We do not yet know the response from the customer. The response will be sent through a feedback Informant described in the next section.

5.1.8 Creating the feedback Informant

This Informant provides Oracle RTD with the information needed to determine the result of the offer selection decision.

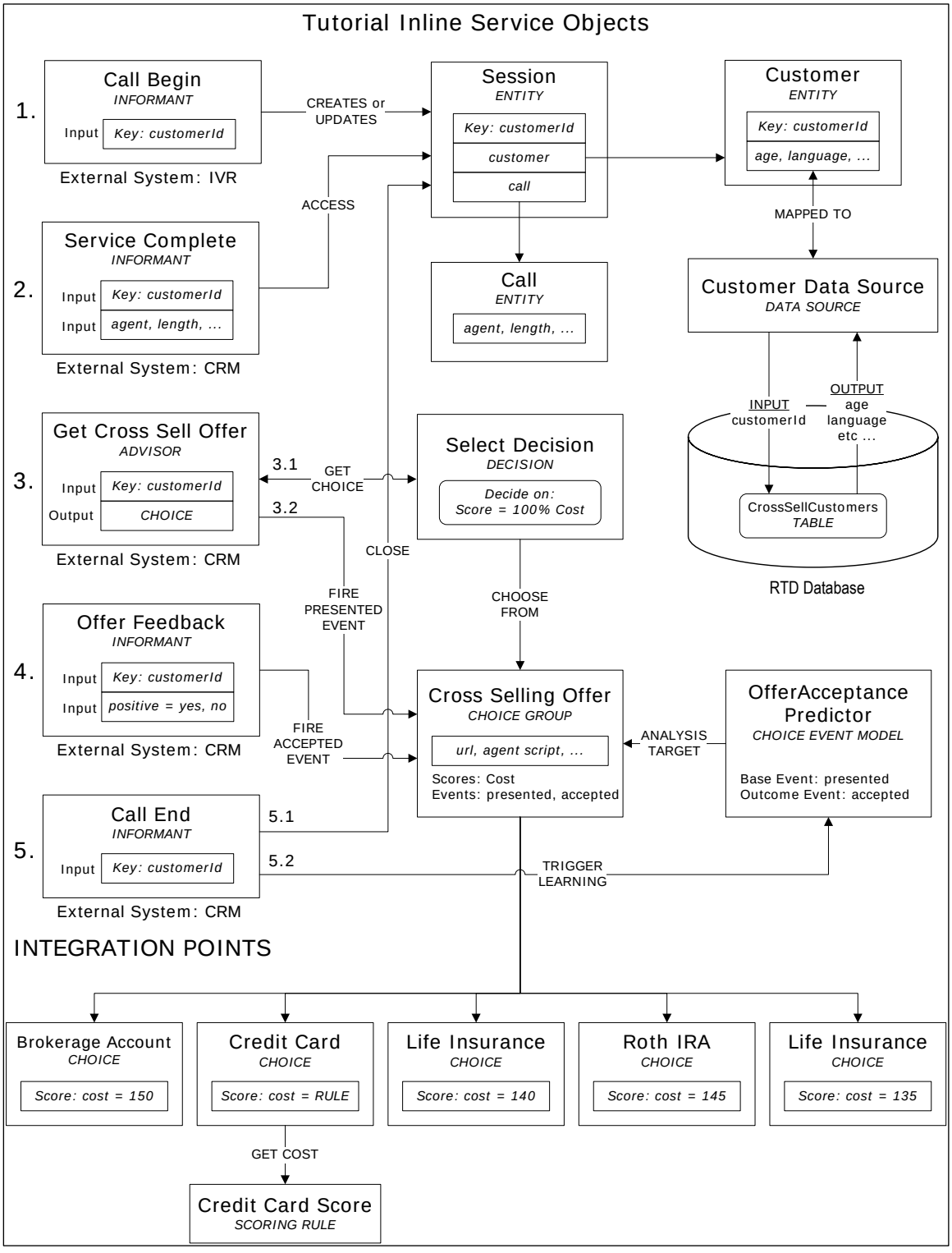
1. In the Inline Service Explorer, expand Integration Points and select Informants. Right click and use the menu and select New Informant. Call the Informant `Offer Feedback`.
2. In the Editor, type a description. Under External System, select CRM. Under Order, enter 4.
3. To add a session key to the Offer Feedback Informant, use Select near Session Keys to choose customerId from Customer. Click OK.
4. Use Add to add an incoming parameter. Call it `Positive`.
5. Select the data type String if is not already selected.

   Leave it unmapped. We do not need to map it to any session attribute because we will use this argument immediately to determine whether the offer was accepted or not. A yes value will be used to indicate offer acceptance.

   Using the Logic tab, enter the following under Logic to record the acceptance event when appropriate.

```
logInfo("Integration Point - Offer Feedback");

//"yes" or "no" to accept offer.
String      positive = request.getPositive();
positive = positive.toLowerCase();


//Get the offer id from session attribute 'OfferExtended'
String extendedOfferID = session().getOfferExtended();

if (extendedOfferID != null) {
   //Get the offer from choice group 'Cross Selling Offer'
   Choice offer = CrossSellingOffer.getChoice(extendedOfferID);

   if (offer != null){
        String offerId = offer.getSDOId();

        //If response is "yes", then record the offer as accepted.
        if (positive.equals("yes")) {
            offer.recordEvent ("accepted");
            logInfo("  Offer '" + offer.getSDOLabel() + "' accepted");
        }

   }
}
```

6. Save all and redeploy the Inline Service. On the Deploy dialog, check Terminate Active Sessions (used for testing).
7. The following diagram shows how the Get Cross Sell Offer Advisor retrieves and presents an offer, and then the Offer Feedback Informant accepts or rejects the offer. When the Call End Informant closes the session, the Offer Acceptance Predictor model is updated with the offer Presented/Accepted events.



5.1.9 Testing the feedback Informant

In order to test the Offer Feedback Informant, we need to first call the Get Cross Sell Offer to retrieve and present an offer.

1 In Test View, select the Integration Point Get Cross Sell Offer. Enter a value for the customerId, such as 10.

2 Click Send  and confirm in the Response subtab that an offer was retrieved. In the Log subtab, you should see something similar to:

```
00:45:28,466 Integration Point - Get Cross Sell Offer
00:45:28,466   Customer age = 38
00:45:28,466   Offer presented: 'Credit Card'
```

Note that even if you tried different values for customerId, the offer presented is always Savings Account or Credit Card. This is because we have only one performance goal at this point – to minimize cost, and Savings Account or Credit Card is the lowest cost, depending on the age of the customer.

3 Now select the Offer Feedback Informant from the Integration Point drop-down list. Leave the customerId as it is, as we want to continue with the same session. Enter a value for input Positive, such as `yes`.

4 Click Send and confirm in the Log subtab that the offer retrieved by the Get Cross Sell Offer Advisor is accepted. You should see something similar to:

```
00:46:01,418 Integration Point - Offer Feedback
00:46:01,418   Offer 'Credit Card' accepted
```

5 Change the input Positive value to `no` and re-Send the Offer Feedback Informant. The Log subtab will look something similar to:

```
00:47:31,494 Integration Point - Offer Feedback
```

5.1.10 Updating the Load Generator script

We will now update the Load Generator script to include calls to the GetCrossSellOffer Advisor and the OfferFeedback Informant. Note that these integration point calls should take place after the ServiceComplete Informant but before the CallEnd Informant, which closes the session. The logic is: 1) call begins, 2) regular service is complete – we record and analyze call reasons using the ReasonAnalysis model, 3) agent presents a cross sell offer to customer, based on lowest Cost goal, 4) we record if customer has accepted offer, 5) call/session ends, OfferAcceptancePredictor model learns on offer presented/accepted.

Add the GetCrossSellOffer Advisor to the Load Generator script:

1 Open Load Generator by running *RTD_HOME*`\scripts\loadgen.cmd`. Then, open the previous script.

2 Select the Edit Script tab, then right-click the left area and select Add Action. The action is of type Message and the Integration Point name should be `GetCrossSellOffer`.

3 In Input Fields, right-click and chose Add item to add an input field. Click in the space under Name and add `customerId`.

4 Click Variable for the input field and use the drop-down list to choose the matching variable, `var_customerId` (see section 3.1.1). Mark customerId as a session key by selecting Session Key.

5 After we add this action to the script, it is placed at the bottom of the actions list. We need to adjust the order so that GetCrossSellOffer is called after ServiceComplete. In the left side of the Edit Script tab, right-click GetCrossSellOffer and select Move Up or Move Down so that the order is CallBegin, ServiceComplete, GetCrossSellOffer, and CallEnd.

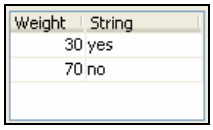
6 Save the Load Generator script.

Add the OfferFeedback Informant to the Load Generator script:

1 Before we add the call to OfferFeedback in the Edit Script tab, we need to create a new variable in the Variables tab. Recall in the definition of the OfferFeedback Informant, the

parameter positive is used to indicate offer acceptance. In Load Generator, we will set the value of this parameter to randomly be yes 30% of the time and no 70% of the time. We do this by using a weighted string array.

2 In the Variables tab, in the left side, right-click on the folder Script and select Add Variable. Enter `var_positive` for Variable name, then set the Contents type to Weighted String Array. Add two items to the array (right-click in the space below the content type and select Add Item). For the first item, double-click in the Weight cell to make it editable and type in the value `30`, and in the corresponding String cell, type in the value `yes`. The second item should have the weight value of `70` and string value of `no`. Note that the weights did not have to add up to 100, they are normalized automatically. Weight values of 6 and 14 would have had the same desired effect.

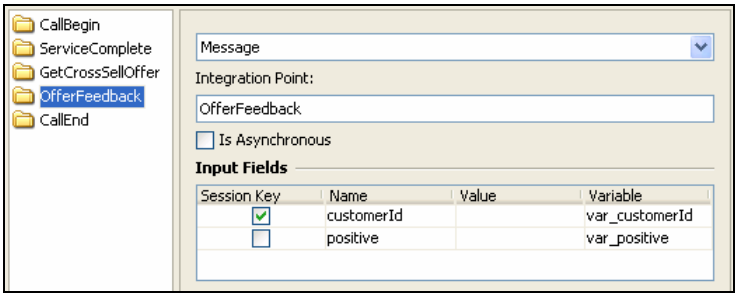


3 Select the Edit Script tab and right-click on the left area and select Add Action. The action is of type Message and the Integration Point name should be `OfferFeedback`.

4 In Input Fields, right-click and chose Add item to add an input field. Click in the space under Name and add `customerId`. In the Variable column, select the matching variable, `var_customerId` (see section 3.1.1). Mark customerId as a session key by selecting Session Key.

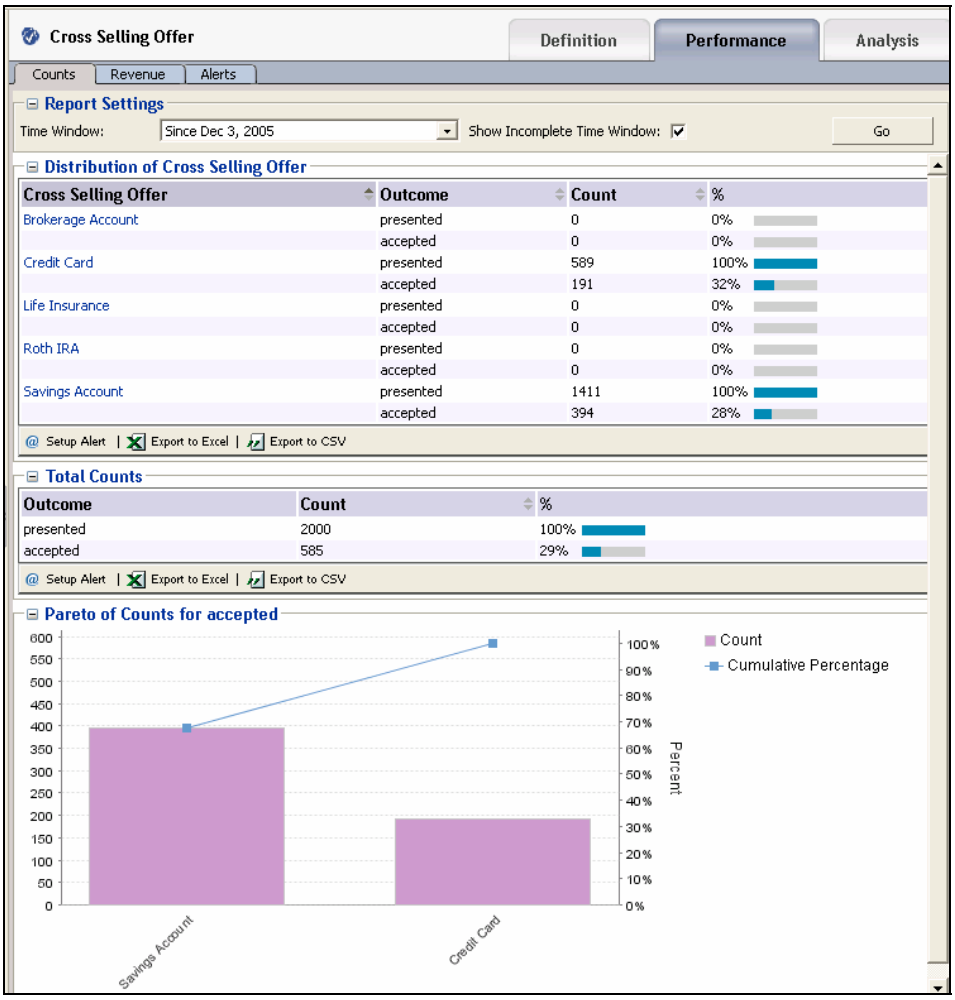
5 Again in Input Fields, right-click and chose Add item to add an input field. Click in the space under Name and add `positive`. In the Variable column, select the matching variable, `var_positive`.

6 After we add this action to the script, it is placed at the bottom of the actions list. We need to adjust the order so that OfferFeedback is called after GetCrossSellOffer. In the left side of the Edit Script tab, right-click OfferFeedback and select Move Up or Move Down so that the order is CallBegin, ServiceComplete, GetCrossSellOffer, OfferFeedback, and CallEnd.



7 Save the Load Generator script.

You can run the Load Generator script at this point. Again, it is recommended that you remove existing data before running the script so the results are not mixed with older data – see section 3.2 for information about how to do this. If you do run the Load Generator script, you can view the results in Decision Center. Log in to Decision Center and click on the choice group Cross Selling Offer to show the results of the Offer Acceptance Predictor model. Click the Performance tab and then the Counts subtab. The distribution of offers and the Pareto graph should look like the following:



Notice that only two offers were presented – Credit Card and Savings Account, and each one had an acceptance rate of about 30%. This is entirely expected due to the logic we have set up so far: 1) Only one performance goal – minimizing Cost was to be met and the Cost is lowest for Savings Account or Credit Card, depending on the age of the customer (see section 4.1.3). 2) In the Load Generator script, we specified 30% of the time, a positive response to an offer is registered through the OfferFeedback Informant. If we drill down into the analysis reports of individual offers, we will not see much correlation between the acceptance of an offer and session attributes. This is because we are using random customer profile data and forcing the acceptance rate to be 30%, regardless of customer or other attributes (such as call length, call agent name, call reason, and so on).

We have now demonstrated how to use performance goal to decide which offer to present and how to use a choice event model to record how often presented offers are accepted. We have only used the model for analysis so far. In the next section, we will add a second performance goal (Maximize Revenue) and use what the model has learned in order to influence which offer is to be presented. We will also introduce an artificial bias that increases the likelihood of customers who have two or more children to accept the Life Insurance offer if it is presented. We will then be able to see how the bias affects the model results.

5.2 Using the predictive power of models

The model we have created learns the correlations between the characteristics of the customers, the call characteristics, and the cross selling results. This model can be used in a predictive fashion, to predict the likelihood an offer will be accepted. We can use the likelihood information to adjust the values of offers when deciding which offer to present. For example, if offer A is twice as likely to be accepted as offer B, it is reasonable to favor offer A when an

offer is picked to be presented. In this section, we will introduce a second performance goal – Maximize Revenue – whose value/score is calculated as the product of the likelihood of acceptance and the base Revenue.

For example, if the base Revenue for the Brokerage Account offer is $300, and the likelihood of acceptance is 30% (0.3), then the Maximize Revenue score is $300 x 0.3 = $90. If the base Revenue for the offer Life Insurance is $185, but the likelihood of acceptance is 60% (0.6), then the Maximize Revenue score is $185 x 0.6 = $111. Even though Brokerage Account had a higher base Revenue value, the Life Insurance offer would be favored because its Maximize Revenue score is higher.

Note that we will be choosing the offer to present based on both the Cost and Maximize Revenue performance goals, so in the previous example, Brokerage Account may still win if the weighted total of its Cost and Maximize Revenue is higher than the total for Life Insurance.

We will begin this section by adding a base Revenue, then adding the second performance goal Maximize Revenue. Then we will set the score for the Maximize Revenue goal to Revenue multiplied by the likelihood of acceptance. Afterwards, we will update the Select Offer decision so that both Cost and Maximize Revenue goals are considered when choosing an offer to present. Finally, in the Offer Feedback, we will add logic to introduce offer acceptance bias for customers with a certain profile who are presented the Life Insurance offer.

5.2.1 Adding a base Revenue choice attribute

1. In the Inline Service Explorer, select the Choice Group Cross Selling Offer. In the Choice Attributes tab, click the Add button.
2. Set the name of this attribute to `Revenue` of data type `Integer`. Make sure the Overridable option is selected, as we will assign a different value for each of the offers.
3. For each choice under choice group Cross Selling Offer, set the value of the Revenue attribute as shown in the following table:

Choice	Revenue Value
Brokerage Account	300
Credit Card	205
Life Insurance	185
Roth IRA	190
Savings Account	175

5.2.2 Adding a second performance goal – Maximize Revenue

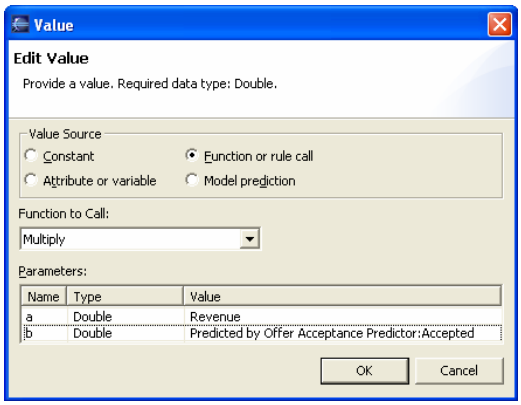
Earlier in this tutorial, we defined a Cost performance goal. Now we will add a second performance goal – Maximize Revenue. We will use the likelihood of acceptance and the base Revenue of the choice in calculating the score for this new performance metric. The formula for this is: (Revenue) * (likelihood of acceptance) = potential revenue score.

1. In the Inline Service Explorer, double-click Performance Goals to open the editor. Use the Add button to add a Performance Metric. Name the metric `Maximize Revenue`, then click OK.
2. In Optimization, choose Maximize and make the metric Required. Since $1 of cost equals $1 of revenue, the Normalization Factor does not need to be adjusted.
3. We need to next add this metric to the Cross Selling Offer choice group. In Inline Service Explorer, select the Choice Group Cross Selling Offer. In the Scores tab, click Add. In the Select dialog, select Maximize Revenue and click OK.

5.2.3 Calculating score value for performance goal Maximize Revenue

To calculate the score value for the Maximize Revenue goal, we need the base Revenue and the likelihood of acceptance value as determined by the Offer Acceptance Predictor choice event model. This can be retrieved easily using the edit value dialog by changing the value source to **Model Prediction**.

1. In **Inline Service Explorer**, select the Choice Group **Cross Selling Offer**. In the **Scores** tab, click in the **Score** column for the Maximize Revenue metric, then click the ellipsis to bring up the **edit value** dialog.
2. For the **Value Source**, select **Function or rule Call**. Under **Function to Call**, choose the function **Multiply**. In the parameters to multiply, click in the **Value** cell for parameter a. Click the ellipsis and choose **Attribute or variable**, then expand the **Choice** folder, select **Revenue**, and click **OK**. In the parameters to multiply, click in the **Value** cell for parameter b. Click the ellipsis and choose **Model Prediction**. Choose the likelihood predicted by the Offer Acceptance Predictor model and the Accepted event, then click **OK**.



   The actual value of the likelihood is from 0 to 1, 1 being 100% likely to accept. It is also possible for the value to be NaN (Not a number), which means the model did not have enough data to compute a likelihood value. In such situations, the Maximize Revenue score cannot be computed and the offer selection by the Select Offer decision will be based on built-in score comparison logic, which depends on whether the score is or is not required.

3. By defining the score for Maximize Revenue on the choice group level, all of the choices within this group will inherit the definition and apply choice-specific values for Revenue and likelihood of acceptance during runtime.

5.2.4 Updating the Select Offer Decision to include second performance goal

We have so far defined a new performance metric and how to calculate its value. We will now update the Select Offer decision to consider both performance metrics when choosing an offer to present.

1. In **Inline Service** Explorer, open the **Decisions** folder and open **Select Offer**.
2. In the **Selection Criteria** tab, you should see in the Priorities for "Default" Segment table only one **Performance Goal**, Cost, with a **Weight** value of 100%. Click **Goals**, then select the goal **Maximize Revenue** and click **OK**.
3. The priorities table now shows two performance goals, each with a **Weight** of 50%. The default is to evenly split weighting between all selected metrics. If you wanted the Maximize Revenue performance goal to take precedence over Cost, you could adjust the percentages so that it had more weight. We will use the default **Weight** of 50% in this tutorial.
4. The following table represents an example of how the Select Offer decision calculates a total score for a particular offer, assuming the offer's Cost score is 150 and its Maximize Revenue score is 215:

Performance goal	Score	Weight	Max/Min	Norm.	Weighted Score
Cost	150	50%	Min	1	-75
Maximize Revenue	215	50%	Max	1	107.5
				Total score	**32.5**

The total weighted score of the offer is 32.5. The weighted Cost score is negative because the optimization is Minimize. The total score of the offer is the sum of the two weighted scores. The total score is calculated for each offer, and the offer with the highest value will be selected.

5.2.5 Adding a choice attribute to view likelihood of acceptance

To view the value of the likelihood of acceptance, we can add a choice attribute and display it through logInfo or in the Response tab of Test view.

1. Choose the Cross Selling Offers choice group from the Inline Service Explorer. In the Choice Attributes tab, click Add to add an attribute. In the properties dialog, set the Display Label to `Likelihood Of Acceptance`. Set the Data Type to `Double`.
2. Deselect the option Overridable, as all choices in this choice group will use the same definition for this attribute. Then, select the option Send to client and click OK.
3. In the Value column for the Likelihood Of Acceptance attribute, click the ellipses  to set its value. In the Edit Value dialog, set the Value Source to Model prediction. Choose the Offer Acceptance Predictor model and the Accepted event, then click OK.
4. Save all changes to the inline service.

5.2.6 Checking the likelihood value

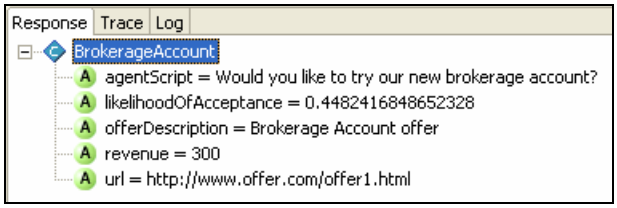
To view values of the likelihood, add a logInfo statement in the Get Cross Sell Offer Advisor.

1. In the Inline Service Explorer, select Get Cross Sell Offer under Integration Points: Advisors. Go to the Editor.
2. In the Asynchronous Logic tab, update the existing code by adding several lines to print the value of the Likelihood Of Acceptance. The completed code should be as follows:

```
logInfo("Integration Point - Get Cross Sell Offer");

logInfo("  Customer age = " + session().getCustomer().getAge() );

// 'choices' is array returned by the 'Select Offer' decision.  The
// name 'choices' was set (and can be changed) in the 'Choice Array'
// text box in the 'Select Offer' decision's 'Pre/Post Selection
// Logic' tab.
if (choices.size() > 0) {

  //Get the first offer from array
  Choice offer = choices.get(0);

  //For the selected offer, record that it has been 'presented'
  offer.recordEvent("presented");

  //Set the session attribute 'OfferExtended' with the offer's ID.
  session().setOfferExtended(offer.getSDOId());

  logInfo("  Offer presented: '" + offer.getSDOLabel() + "'");

  //Cast selected offer to type CrossSellingOfficeChoice -
  //the base Choice type of choice group 'Cross Selling Offer'
  CrossSellingOfferChoice cso = (CrossSellingOfferChoice) offer;
  logInfo("  Likelihood of Acceptance = " + cso.getLikelihoodOfAcceptance());

}
```

3 To see the effect of the changes to the Advisor, save all and deploy the Inline Service.

4 In Test view, select the Get Cross Sell Offer integration point and input a value for customerId, such as 8. Click Send. In the Response subtab in Test view, you should see something similar to:



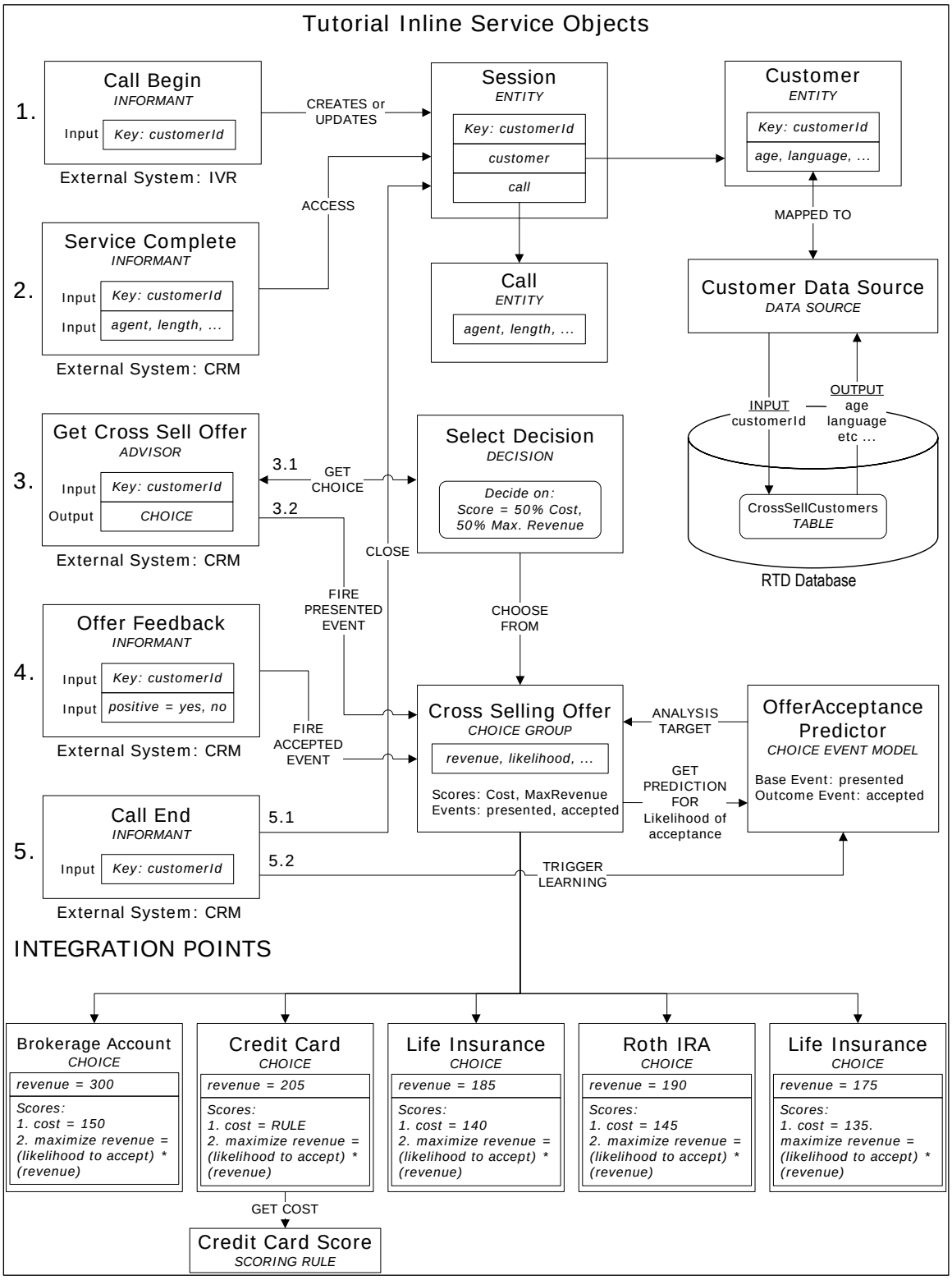
In the Log subtab, you should see something similar to:

```
14:07:37,908 Integration Point - Get Cross Sell Offer
14:07:37,908   Customer age = 57
14:07:37,908   Offer presented: 'Brokerage Account'
14:07:37,908   Likelihood of Acceptance = 0.4482416848652328
```

If you are getting a value of NaN (Not A Number) for Likelihood Of Acceptance, this means the model did not have enough data to compute the likelihood value for this offer. The number of iterations necessary to reach model convergence (likelihood numbers no longer NaN) depends on the application and quality of the data.

In our case, we had imposed a definite offer acceptance rate of about 30% (see section 5.1.10), and since we are using random customer profile data, the Offer Acceptance Predictor model should converge quickly and be able to compute likelihood of acceptance values within just a few hundred iterations. Before the model has reached convergence, the offer selection process is based on built-in score comparison logic, which depends on whether the score is required.

5 The following diagram shows the Get Cross Sell Offer Advisor retrieving an offer from the Cross Selling Offer choice group, where the total score of each offer is a weighted sum of two scores – Cost and Maximize Revenue.



5.2.7 Introducing offer acceptance bias for selected customers

Earlier in the Offer Feedback Informant, we specified whether to accept a presented offer through the Positive Informant parameter. We then updated the Load Generator script so that when this Informant is called, we pass the value yes to the parameter Positive 30% of the time (see section 5.1.10). This percentage did not depend on any customer profile data – any presented offer had a 30% chance of being accepted by any customer.

If we run the Load Generator Script at this point, the models would not show any strong correlation between customer attribute to the acceptance of the offer. We will introduce an artificial bias in the Offer Feedback Informant logic which will always record positive offer acceptances for customers who have two or more children and who were presented the Life Insurance offer. This logic is in addition to the default acceptance rate (as defined in the Load Generator Script) and will skew the acceptance rate for the Life Insurance offer to more than 30%. In Decision Center, we will be able to see clear correlations between the number of children and the acceptance rate of this offer.

1 In the Inline Service Explorer, select Offer Feedback under Integration Points: Informants. Go to the Editor.

2 In the Logic tab, update the existing code by adding several lines to add offer acceptance bias for customers who have two or more children and who were presented the Life Insurance offer. The completed code should be as follows:

```
logInfo("Integration Point - Offer Feedback");

//"yes" or "no" to accept offer.
String      positive = request.getPositive();
positive = positive.toLowerCase();


//Get the offer id from session attribute 'OfferExtended'
String extendedOfferID = session().getOfferExtended();

if (extendedOfferID != null) {
   //Get the offer from choice group 'Cross Selling Offer'
   Choice offer = CrossSellingOffer.getChoice(extendedOfferID);

   if (offer != null){
        String offerId = offer.getSDOId();

        //Introduce artificial bias for customers with 2 or more
        //children to always accept "LifeInsurance" if it was
        //selected after scoring.
        //If data source is Oracle, change the following method from
        //getNumberOfChildren() to getNumberofchildren()
        int numOfChildren = session().getCustomer().getNumberOfChildren();
        if ( numOfChildren >= 2 && offerId.equals("LifeInsurance")) {
            positive="yes";
        }

        //If response is "yes", then record the offer as accepted.
        if (positive.equals("yes")) {
            offer.recordEvent ("accepted");
            logInfo("  Offer '" + offer.getSDOLabel() + "' accepted");
        }

   }
}
```

3 Save all changes and deploy the inline service.

5.2.8 Running Load Generator script

In section 5.1.10, we updated the Load Generator Script to include the GetCrossSellOffer Advisor and the OfferFeedback Informant. At that point, the offer selection process was based on only one performance goal – to minimize Cost. We then added a second performance goal, Maximize Revenue, which uses predicted values of acceptance likelihoods as computed by the Offer Acceptance Predictor model. The offer selection process now depends on both performance goals. We have also introduced an artificial acceptance bias for customers who fit a certain profile, and who were presented the Life Insurance offer. We will now run the Load Generator script again to see the results.

1 If you are using OC4J or WebLogic, open JConsole by running *JAVA_HOME*`\bin\jconsole.exe`. If you are using WebSphere, run the batch script

you created during JConsole configuration. See *Installation and Administration of Oracle RTD* for more information about using JConsole.

2 Click the Remote tab. Then, enter the appropriate port number (typically 12345) and the administrator credentials you created during installation and click Connect.

3 Click the MBean tab, then go to the OracleRTD > InlineServiceManager > Tutorial > Development > Loadable MBean.

4 Click the Operations tab, then use the deleteAllOperationalData() operation to remove all operational data, including the study, for this Inline Service.

5 Start Load Generator and open the Load Generator script previously defined. There should be no changes necessary.

6 Start the Load Generator script. After about 200 total finished scripts, click the pause button  to temporarily stop sending requests to the server. View the server's output in the server.log file. You will see that the printed Likelihood Of Acceptance values are NaN for all sessions. This is an indication that the model has not yet learned enough data to be able to compute the likelihood of acceptance. Note that offers are still being presented despite the lack of likelihood values. Offers are being selected using built-in scores comparison logic.

7 Un-pause the Load Generator script and let it finish running for 2000 total finished scripts. In the server output, you should now see actual values for Likelihood Of Acceptance, varying around 0.3 for all offers except Life Insurance, which has higher values because of the bias introduced.

8 It is important to note that the model-predicted Likelihood Of Acceptance values for a given offer will differ for different customer profiles. For example, suppose we have two customers John and Tom, who only differ in the number of children they have. If we printed the Likelihood Of Acceptance values for the Life Insurance offer for these two customers (at a snapshot in time), we will see a higher value for Tom. This is because Tom has three children, and is therefore more likely to accept the Life Insurance offer, if it is presented to him.

Customer	# of children	Likelihood Of Acceptance for offer Life Insurance
John Doe	0	0.32
Tom Smith	3	0.89

Since we determine which offer to present to the customer based on the combination of Cost and Maximize Revenue scores, and because Maximize Revenue depends on the model's predicted Likelihood Of Acceptance value for each offer, the Life Insurance offer will have a high Maximize Revenue value for customers with two or more children, and therefore for such customers, Life Insurance will be presented (and then accepted) far more frequently than other offers!

5.2.9 Studying the results

To view the results of the Load Generator run, log into Decision Center. Click the choice group Cross Selling Offer in the left navigation box. This will show the results of the Offer Acceptance Predictor model. Click the Performance tab and then the Counts subtab. You should see a table similar to the following:

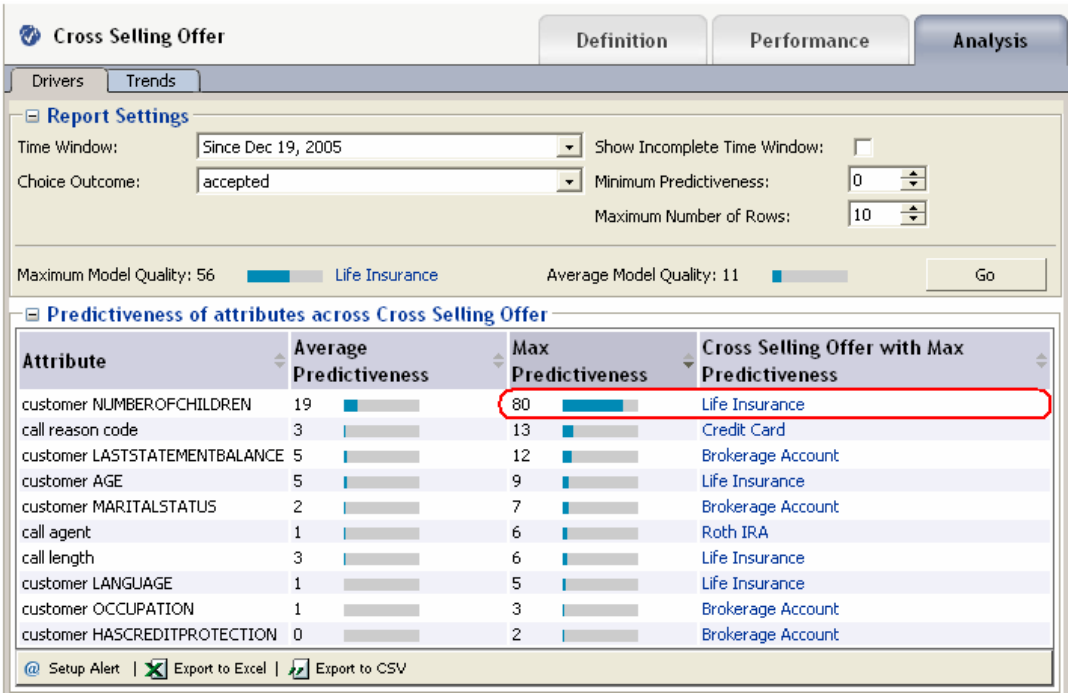
Distribution of Cross Selling Offer

Cross Selling Offer	Outcome	Count	%
Brokerage Account	presented	735	100%
	accepted	260	35%
Credit Card	presented	164	100%
	accepted	50	30%
Life Insurance	presented	660	100%
	accepted	334	51%
Roth IRA	presented	312	100%
	accepted	96	31%
Savings Account	presented	129	100%
	accepted	43	33%

Setup Alert | Export to Excel | Export to CSV

The above table shows the distribution of the offers – how many were presented and how many were accepted for each offer. Except for Life Insurance, all of the other offers had acceptance rate of about 30%. This is expected because of how we set up the load generator script (see section 5.1.10). The acceptance rate for Life Insurance is higher than 30% because of the artificial bias we introduced in section 5.2.7. The bias dictated that in addition to 30% of the customers accepting any offer, customers who had two or more children and were offered Life Insurance will always accept the offer.

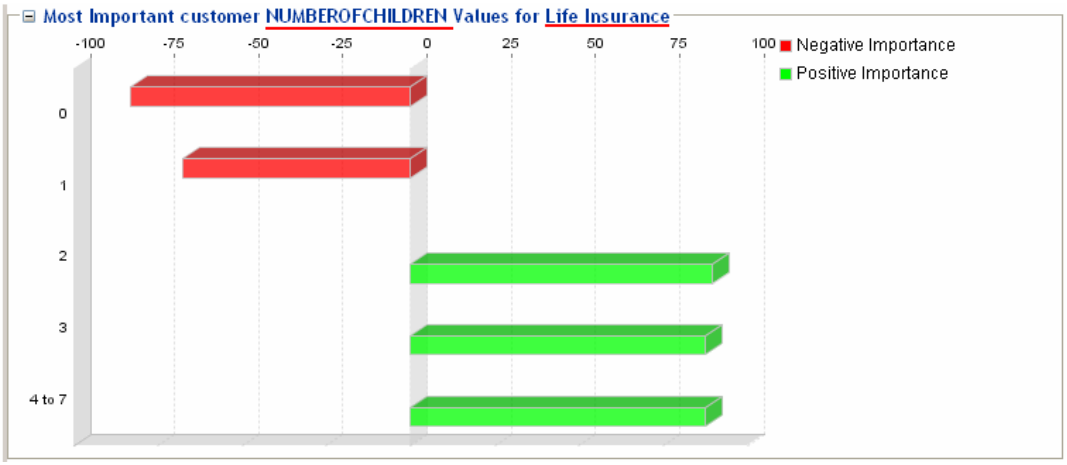
Given the artificial bias, the model results should show that for the Life Insurance offer, the NumberOfChildren attribute will be an excellent predictor for whether or not the offer will be accepted. This is exactly what we see in the Decision Center reports: click the choice group Cross Selling Offer and click on the Analysis tab, then the Drivers subtab. In the Report Settings section, change the Minimum Predictiveness value to 0 and then click Go. You will see a list of attributes, ordered by the maximum predictiveness value. The highest value for Max Predictiveness should be for the NumberOfChildren attribute, since it is the only artificial bias we added. The corresponding offer should be Life Insurance, similar to the following figure:



We can further analyze the importance of the NumberOfChildren attribute for the Life Insurance offer by viewing reports specific to this offer. In the navigation box in Decision Center, expand the choice group Cross Selling Offer and click on the choice Life Insurance, then click on the Analysis tab and finally the Drivers tab. This report shows the important drivers for acceptance of this particular offer (Life Insurance).

In the Report Settings section, change the Minimum Predictiveness value to 0 and then click Go. You will see a list of attributes, ordered by the Predictiveness value. The NumberOfChildren attribute should have the highest

predictiveness value. Click the attribute name to display more detailed reports, the first of which should look similar to the following figure:



This graph shows that for NumberOfChildren values of 2 and above, there is a strong positive correlation for offer acceptance. This means that the number of acceptances of this offer for these attribute values (2 or more) is much higher than expected. Similarly, for values of 0 or 1, the correlation is also very strong, but is negative, meaning that customers with 0 children or 1 child did not accept Life Insurance as much as expected.