Siebel RTD

Getting Started with Siebel RTD

Version 2.1   Document Revised July, 2005

Copyright

Restricted Rights Legend

Information in this document is subject to change without notice and does not represent a commitment on the part of Sigma Dynamics. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, Sigma Dynamics DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Getting Started with Siebel RTD

Preface

About this document

This document contains information and examples to help you get started in using Siebel RTD. This platform allows you to develop enterprise software solutions that analyze business process behavior and make recommendations *in real-time,* allowing you to identify and address problems and opportunities as soon as they emerge.

Intended Audience

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

To facilitate the transfer of knowledge this guide assumes the reader has installed Siebel RTD in a Windows system, and is using the Microsoft SQL Server database.

How to use this guide

This document is divided into the following sections: **Section 1: About Siebel RTD** provides background on Siebel RTD and provides hands-on information on creating an Observer Inline Service; **Section 2: Simulating load for Inline Services** introduces the user to the load generation toll included with Siebel RTD; **Section 3: Enhancing the Call Center Inline Service** expands the functionality of the Inline Service; **Section 4: 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, including labels, tabs, menus, etc.
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 Siebel RTD

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

Siebel 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). The Siebel RTD also includes other helpful load testing and debugging tools.

1.1 Terminology

Siebel RTD consists of 5 components:

- Siebel Decision Studio
- Siebel Real-Time Decision Server
- Siebel Decision Center
- Administration (JMX)

Inline Service refers to the configured application that is deployed.

Inline Services are configured and deployed using Studio and analyzed and updated using Decision Center. Inline Services run on Siebel 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.
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 Siebel Decision Studio

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

Studio is fully integrated with Eclipse, an open source Java IDE produced by the Eclipse Foundation. 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 aren't familiar with Eclipse, it is completely transparent to using Studio. Eclipse and Studio both have online help available through the Help menu.

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.

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 template you have chosen.

1.2.2 Problems View

Problems View shows validation and compiler errors as you build your Inline Service. If you double click on a compilation error, Problem View opens the generated source code at the point of the error. If you double click on a validation error, Problem View opens the meta data at the point of the error.

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 Sheet View

The Cheat Sheets view provides step by step instructions for common tasks. On installation it is located on the right hand side of the window.

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 on the element you wish to edit.

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

1.2.6 Arranging views and re-sizing 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 which provides functionality.

You can drag the views and editors of a perspective to any space on the screen. Views and editors will re-size 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 on the editor tab.

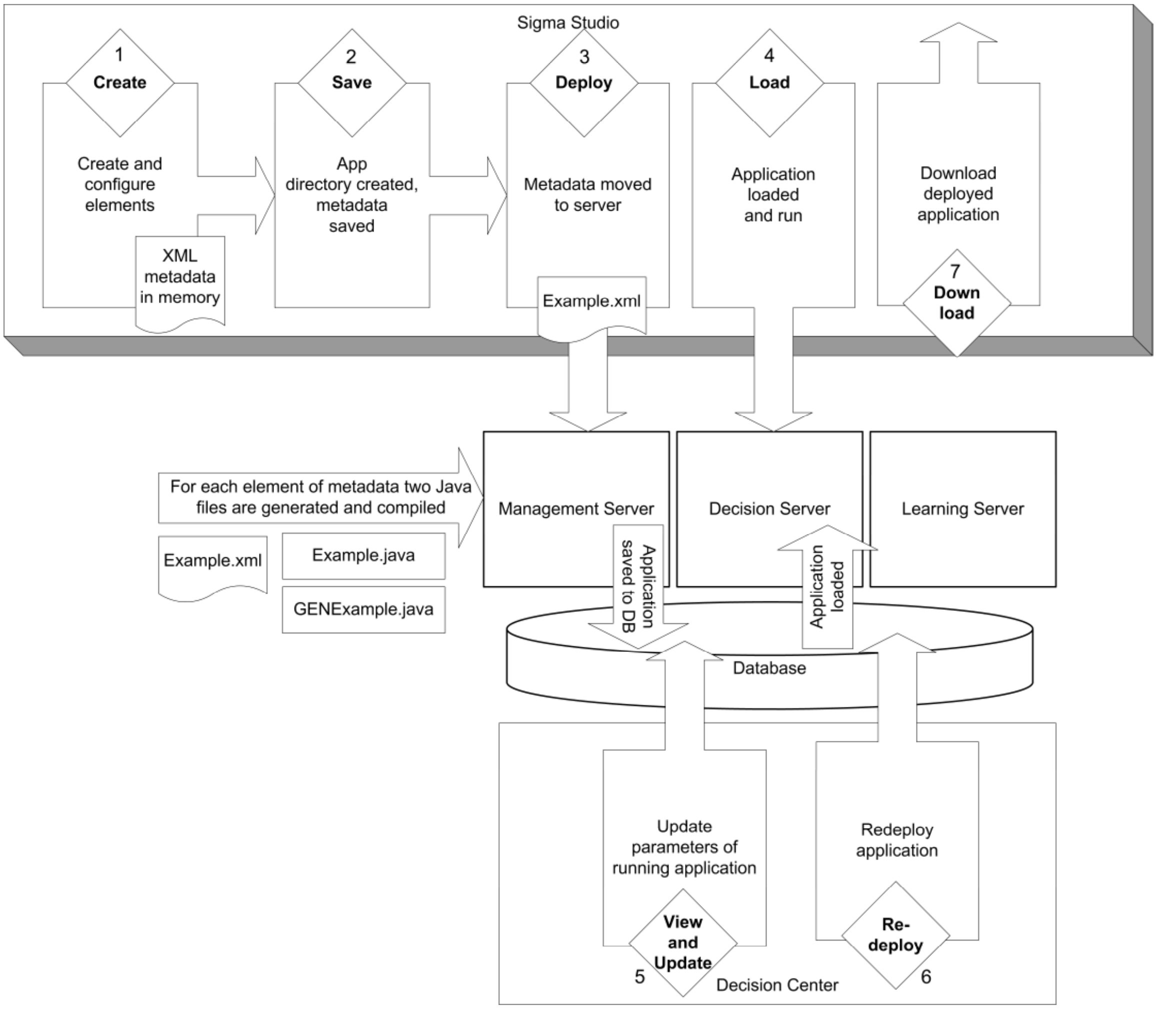
Both Editors and Views can be toggled between Maximize and Minimize by double-clicking on the tab or by using the right click menu item.

1.3 About Siebel Decision Center

Siebel 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.

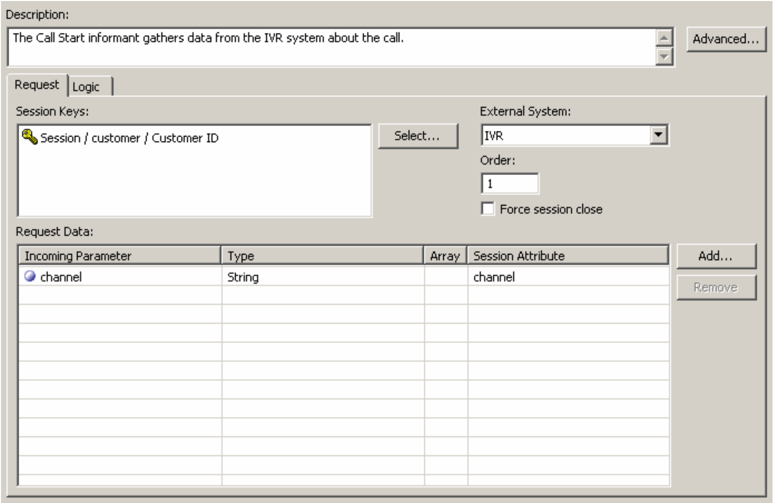
The Decision Center user interface displays Inline Services in two panes. The left pane shows the list of Inline Service elements while the right pane displays detailed information related to the selected element.

1.4 Overview of the Inline Service lifecycle



Inline Services are created using Siebel Decision Studio. The overall process by which Inline Services are created, deployed and loaded is outlined below:

1. **Create:** Using Studio, elements are created and configured. Configuration in many cases consists of checking attributes or settings that apply to your business situation. Some elements include Logic or Asynchronous Logic attributes that allow the introduction of Java scriptlets. For instance, an Informant element is represented below. This element is named CallStart.



Description: The Call Start informant gathers data from the IVR system about the call.

Session Keys: Session / customer / Customer ID

External System: IVR

Order: 1

□ Force session close

Request Data:

Incoming Parameter	Type	Array	Session Attribute
channel	String		channel

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

2. **Save:** By saving the Inline Service in Studio, the metadata is written to an Inline Service directory on the local file system. The CallStart metadata is shown here.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema:RTAPType
xmlns:schema="http://www.sigmadynamics.com/schema"
internalName="CallStart" name="CallStart"
forcesSessionClose="false" order="1.0">
  <schema:description />
  <schema:system ref="IVR" />
  <schema:sessionKey path="customer.customerId"
relativeTo="session" />
  <schema:request ref="BasicRequestData" />
  <schema:mapper ref="BasicRequestMapper" />
- <schema:body>
  <schema:java order="0">session().getCustomer().fill(); /*
Trigger data retrieval */</schema:java>
  </schema:body>
- <schema:postOutputBody>
  <schema:java order="0" />
  </schema:postOutputBody>
  </schema:RTAPType>
```

The attributes that were assigned in Studio, such as session key and external system, are represented here. Note that the Java scriptlet is inserted into the body of the schema.

3. **Deploy:** The Inline Service is deployed to the Management Service using Studio. Once it is resident on the server, two classes of Java code are generated. One set is the Inline Service; it is named the element name preceded by 'GEN'. For instance, the CallStart element will produce a class GENCallStart.java.

Another set is available for overriding the generated code, for instance, if you want to call your own Java code. This class is named the same as the element name. For instance, the CallStart element will produce a class CallStart.java. Note that CallStart simply extends GENCallStart.

When the Inline Service is compiled, the generated code is used, unless there is a change to the corresponding class in the override code. In that case, the override is used. The compiled Inline Service is saved to the database.

4. **Load:** The Management Service automatically loads the Inline Service to the Decision Service.
5. **View and Update:** Reports and learnings are viewed through the Decision Center interface. Parameters to your Inline Service can be updated from Decision Center.
6. **Re-deploy:** If updates are made to the Inline Service through Decision Center, it is re-deployed to the Decision Service.
7. **Download:** Using Studio, you can download a deployed Inline Service from the server. This is useful if you were not the original developer of the Inline Service, or if changes have been made and re-deployed through Siebel Decision Center. Downloaded Inline services can be updated and redeployed to the same or different 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 Siebel 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 Siebel Real-Time Decision Server, where it runs.
- Results from the Inline Service are viewed through Siebel 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 be quite varied, 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 end of call 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

Start the Siebel Real-Time Decision Server before you begin.

1 From the Start menu, select **Siebel Real-Time Decision Server** from the **Siebel Analytics→RTD** program group. A command window will open and the output from the server will be displayed in it. Startup time for the server varies between about 30 seconds and a few minutes, depending on the hardware, load and the Inline Services that are currently deployed.

2 You will know that the server is ready when you see a line that indicates the time it took to startup, as in the following:

```
14:12:11,837 INFO  [Server] JBoss (MX MicroKernel) [3.2.2 (build:
CVSTag=JBoss_3_2_2 date=200310182216)] Started in 2m:25s:58ms
```

2.2.2 How to configure the Application element

1 From the Start menu, select **Siebel Decision Studio** from the **Siebel Analytics→RTD** program group. Once 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 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**.

3 Expand the **Tutorial** and **Service Metadata** folders. Double-click on 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. Select the Siebel Real-Time Decision Server (by default localhost:8080). **Connect** appears. Login to the server using the username and password you created on installation.

5 Once you are logged on to the server, check the box **Show users** and use **Get Names** to display a list of users on the server. Your username should be listed. Choose your username 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.

2.2.3 How to configure the Performance Goal element

1 Double-click on the **Performance Goal** to open the editor. Use the **Add** button to add a Performance Metric. Name the metric 'Cost'. Click **OK**.

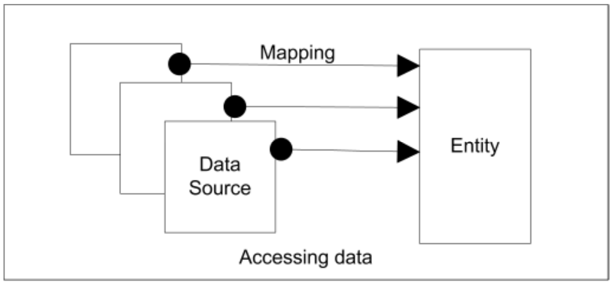
2 In **Optimization** choose 'Minimize' and make the metric **Required.**

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 "Minimize hold time" measured in seconds, the normalization factor would be how many minimized seconds are worth a dollar (revenue) to your organization.

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'. Click **OK**. The data source Editor appears.

2 Under **Description**, add a 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 will not be 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. Select and use **Remove** to remove unnecessary columns. Retain *only* the following columns.

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

3. Set the input for the table. The input is the column that you will be matching on to retrieve the data record. In this case we use the key, "Id". Use **Add** to add the Input to the data source. The data type is String.

4. Save your work using **File->Save All**. If there are errors in your work, you will receive notification.

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 a description 'Customer entity'.

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 the 'Customer' data source. Select the Customer data source from the list.

3 Under **Default Value** click to get an insertion point and add a default value for each attribute. String data types require quotes which are added automatically.

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, 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 uncheck the **Use for Analysis** checkbox.

The checkbox **Show in Decision Center** 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, and click **OK**.

> **Note:** The entity key attribute will always be of type String.

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 Siebel RTD framework. To associate the Entity, 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 Siebel RTD. The Session contains entities that represent the Caller and the Agent. For the duration of the session (i.e. 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'. Add a **Description** for 'customer'. 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:** Siebel 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

To associate the Customer entity with the Customer Data Source a mapping is created by Studio. The Mapping Element is automatically created by Studio when you associate the entity to a data source.

Our mapping of attributes was automatically done when the attributes were imported from the data source. We now map the key.

On the Customer Entity Editor, select the **Mapping** tab of the entity. Entities are identified by an 'E' icon

1 Since we used Import, the data attributes are already mapped to the entity attributes. If you had added additional attributes beyond the import, they are mapped by clicking ... under **Source** and locating the data source attribute, calculated value or constant to map to.

2 For the key value, we map the Input column to a Session attribute. Your key will appear under Data Source Input Values after you map all attributes. Click in the **Input Value** cell to bring up Edit Value.

3 Choose **Attribute or Variable**. Under Session expand the tree to Customer, the attribute of the Session. Choose 'customerId'. Click **OK**.

2.7 Creating an Informant

Informants are a type of Integration Point that can send a message to the Siebel 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, then deploy the Inline Service to the Siebel Real-Time Decision Server and 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'. Click **OK**.

2 In the Testing Editor add a description under **Description**.

3 Click **Advanced** next to Description. Uncheck **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 I Testing ✕.

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

> **Note:** When you configure an entity in 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. 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**.

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. Since Informants do not return value to their callers, so the results will be seen in the **Log** tab of **Test View**.

2.8.1 Deploying the Inline Service for testing

1 Using Studio, save the Inline Service using **File→Save All**. Use Deploy 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 a server to deploy to. Deploy to the location of your Siebel Real-Time Decision Server. This defaults to localhost, as does the default configuration of the Installation. Use the pull down menu to select a deployment state, 'QA'. Check **Terminate Active Sessions (used for testing)**. Click **Deploy.**

Deployment takes anywhere from about 30 seconds to a few minutes. A message 'Tutorial deployed successfully' will appear below the Inline Service Navigator when deployment is complete.

> **Note:** The reason we terminate active sessions is that we will be testing this Inline Service several times, and we do not want to leave a session resident on the server.

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 to see the results. Every printout coming from a `logInfo` command will be printed out with a timestamp.

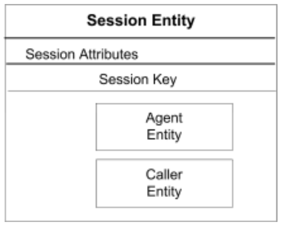
Your results should look similar to this:

```
11:53:54,102 Customer age = 80
```

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, etc. 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 Object Name** appears. Enter '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
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**.

2.9.2 Creating the end of call informant

Now we create an informant that will report on the call data. This informant is called by the CRM application at the end of the call. 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**.

2 Give the element a description.

3 In the Inline Service Explorer, select the group **Informants**. Right-click and use the menu to **New Informant. Object Name** appears. Name the system 'End of Call' and click **OK**.

4 Using the Informant Editor, enter a description.

5 To add a session key to the End of Call Informant, click **Select** adjacent to **Session Keys**. Expand **Customer** and choose **customerId**. Click **OK**.

6 To add the additional pieces of data to the informant, do the following *for each incoming item* 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 pull down. Click **OK.**
- Under **Session Attribute,** click on the ellipsis … to use **Assignment**. Follow the drop down menu to 'call' and then the call attribute that matches the incoming item.

Name	**Type**
agent	String
length	Integer
reason code	Integer

2.9.3 Testing the end of call informant

To test this Integration Point and the assignments we will add logic to log the incoming items. We will leave this code in for a while to get an indication in the **Test View** of events being received.

1 On the End of Call Informant Editor select the **Logic** tab. Enter the following code in the **Logic** pane.

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

2 Use **File→Save All.**

3 **Deploy to the server.** Remember to check **Terminate Active Sessions (used for testing)**.

4 Test the Integration Point by using the **Test** view at the bottom of the screen. Select the **End of Call** integration point and enter values for the Informant arguments.

- customerId - 7
- agent – 'John'
- length – '24'
- reason code – '17'

Click **Send** ▶ to send the request.

5 You should see a similar response to this on the **Log** tab.

```
09:32:33,177 Integration Point - End of Call
09:32:33,177 Reason Code: 17
09:32:33,177 Agent: John
09:32:33,177 Call Length: 24
```

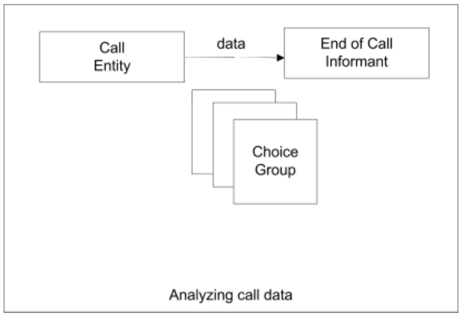
Tip: Troubleshooting

If there were errors in compilation a dialog in Studio will show these errors in the **Problem** view. Double clicking on the error will take you to the editor of the element that has the error.

Make sure that the server you are communicating with is “localhost.” Studio will remember in the dropdowns the values previously entered, and the default may not be 'localhost'.

2.9.4 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. First we will create a choice group for the call reasons. Then we will add an attribute for call reasons, the reason code.



2.9.5 Adding a choice group

1. In the Inline Service Explorer, select the group **Choices**. Right-click and use the menu to **New Choice Group.** Name the group ‘Call Reason’ and click **OK**. Add a description.
2. On the Choice Editor, on the **Attributes** tab, click **Add** next to **Choice Attributes**. Name it ‘code’. Select data type 'Integer’. 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 on **Call Reason** in the Inline Service Explorer under Choice 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 **Choice Groups**, 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 **Attribute Value** add the value for the code attribute from the following table:

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

2.9.6 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.

2.9.7 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 Uncheck **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.**

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

5 Use **Select** to choose ‘End of Call’ from the list.

6 Save the configuration.

2.9.8 Adding logic for selecting choices

When the End of Call 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 **End of Call** 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 - End of Call");
logInfo ("reason Code: " + session().getCall().getReasonCode());
logInfo ("Agent: " + session().getCall().getAgent());
logInfo ("Length: " + session().getCall().getLength());


int code = session().getCall().getReasonCode();
if (code == 17){
```

```
ReasonAnalysis.addToChoice("CheckBalance");
logInfo ("CheckBalance was added to the model");
}

else if (code == 18) {
ReasonAnalysis.addToChoice("MakePayment");
logInfo ("MakePayment was added to the model");
}

else if (code == 19) {
ReasonAnalysis.addToChoice("RateInquiry");
logInfo ("RateInquiry was added to the model");
}

else {
ReasonAnalysis.addToChoice("Other");
logInfo ("Other was added to the model");
}
```

2.9.9 Testing it all together

1. Save the configuration using **File→Save All**. Deploy the configuration to the server. Make sure there are no errors in deployment or compilation.
2. Using the **Test** view test the Integration Point.
3. Select **End of Call** and set values for the different arguments.
   - customerId - 7
   - agent – 'John'
   - length – 35
   - reasonCode – 17

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

```
14:18:58,227 Integration Point - End of Call
14:18:58,237 reason Code: 17
14:18:58,237 Agent: John
14:18:58,237 Length: 35
14:18:58,237 CheckBalance was added to the model
```

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

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 runtime 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 all the 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 point defined in the Inline Service, namely End of Call.

This informant expects four parameters:

- the customer Id ('customerId')
- the reason code for the call ('reasonCode')
- the agent name or identification ('agent')
- the length of the call ('length')

Note: Load Generator needs the object IDs as parameters. To view the object IDs in Studio, use the object ID toggle icon on the Inline Service Explorer taskbar.

3.1.1 Creating the performance under load script

1 Using the **Start** menu, open **Load Generator** by selecting it from the **Siebel Analytics→RTD** program group. Click on **Creates 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 Configuration File	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 not suitable for our needs.	$INSTALLDIR\client\clientHttpEndPoints.properties
Graph refresh Interval in Milliseconds	This parameter only affects the UI. It determines the refresh rate for the UI graph and counters. The default is to refresh every 2000 milliseconds, that is, every 2 seconds.	2000
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

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

4 Save the configuration. It is customary to save Load Generator script configurations in a folder named 'etc' together with the other metadata.

5 To define the values for the parameters to the Integration Point, Click on the **Variables** tab.

> **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 on **Script** and select **Add Variable**. Name it '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 customerId 1 and the last one will be 2000.

7 Right-click on **Script** and select **Add Variable** to add each of the following variables for the parameters to the Informant:

Parameter	**Variable**
reasonCode	Integer range between 15 and 21, randomly selected.
agent	A string array. To add a string to the array, right-click on the table area and select **Add Item**. Then select the newly created row to get a cursor and type the name to be used. Add a few sample values of agent names.
length	An integer between 75 and 567, randomly selected. This will be used as the length of the call in seconds.

8 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 is 'EndOfCall'.

9 In **Input Fields,** right-click and chose **Add item** to add an input field for each of the variables. Click in the space under **Name** and add the following input items.

- customerId
- reasonCode
- agent
- length

10 Click on **Variable** for each input field and use the pull down to choose the matching variable.

11 Set **Integration Point** to **EndOfCall.**

12 Mark customerId as a key by checking the box under **Session Key**.

13 Once again, save the load generator configuration script.

14 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 Number of Errors' and if the number is above 0 then 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 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 then there may be a mistake in the definition of the 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, as mentioned above.

3.1.2 Viewing performance 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 Using the **Start** menu, open **Decision Center** by selecting it from the **Siebel Analytics→RTD** program group. Logon using the default administrator ID you created on installation.

2 Expand **Call Reason** and select one of the Choices, such as **Make Payment.** In the right pane navigate to the **Analysis** tab. This report summarizes the number of times each reason was seen and correlations between the attributes and reasons for calls.

3 You will see something interesting. The **call reason code** has an expectedly strong correlation to the Call Reason.

Since we generated the data randomly, we would not expect to have any significant correlations. Of course, in this case what happened was that the call reason code absolutely determines the reason for call. To solve this we should exclude this attribute from the model.

3.1.3 Excluding the attribute

1 Go to **Studio**. Open the **Tutorial** project.

2 Expand **Models** and select **Reason Analysis** from the Inline Service Explorer.

3 Go to the **Attributes** tab. In the lower table, titled **Excluded Attributes** expand the 'call' Entity and select 'reason code'.

4 Save all and redeploy to the localhost server.

5 You can now rerun 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 run the load generator.

3.2 Resetting the Model learnings

To reset the learning we use the JMX console. You can use the JMX console available with your application server, or use the console provided with Siebel RTD. We explain here how this is done from the included console.

1. From the **Start** menu, open the JMX management console by selecting **Administration (JMX)** from the **Siebel Analytics→RTD** program group.
2. A browser window will open with an entry point to the JMX Management Beans. Click to open. A table for with links to manage the platform appears.
3. In the row **Local Server,** locate your server and click to open. Locate the **LearningService** object and click on **View MBean**.
4. Now locate the **DeleteStudy** operation and invoke it with 'Tutorial' as a parameter. This will delete the learning from all the models in the 'Tutorial' study.

In order to see the new results in Decision Center you will have to run the Load Generator script again.

3.2.1 Summary of the Inline Service

During this exercise, we created a 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 an Integration Point and a model to do the analysis. Logic was added to determine which reason the call was made. We then generated data for the Inline Service to see results in Decision Center.

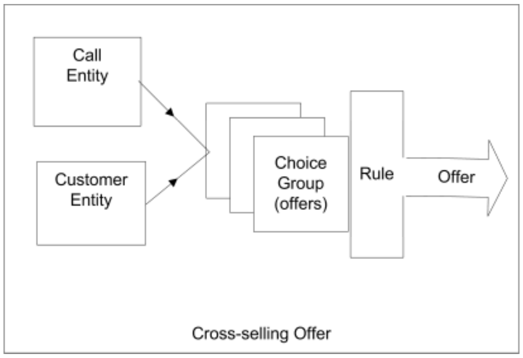
Section 4: Enhancing the Call Center Inline Service

In Section 1 we created an Inline Service that tracks and analyzes incoming data regarding the reason of calls to a credit card call center. In Section 2 we used the Siebel Load Generator to test the performance of our Inline Service.

In this section we will enhance the Call Center Inline Service to provide cross selling advice to a front end application. The Siebel Real-Time Decision Server will be used to select the best offer for each customer based first only on rules and then on self learning.

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 the newly created group. Add a description.

3 On the **Attributes** tab click **Add** next to **Choice Attributes.** Add the following attributes, making sure to check **Send to client** and **Overridable**:

Attribute	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 there to present the offer.

They are overridable because they need to be different for each actual offer. Offers will be represented by choices in this 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 maybe adds 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 the choices from the table below.

For each of the choices:

- Right-click on **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 **Choice Groups**, expand the **Cross Selling Offer** Group to show the choices Credit Card, Savings Account, Life Insurance, Roth IRA and Brokerage Account.
- For *each of the five choices*:

  1. Select the Choice in the Inline Service Explorer. In the Editor for that choice add a description for each.
  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	**Agent Script**	**Offer description**	**URL**
Brokerage Account	"Would you like to try our new brokerage account?"	Brokerage Account offer	http://www.offer.com
Credit Card	"Would you like to try our new credit card?"	Credit Card offer	http://www.offer.com
Life Insurance	"Would you like to try our new life insurance?"	Life Insurance offer	http://www.offer.com
Roth IRA	"Would you like to try our new Roth IRA?"	Roth IRA offer	http://www.offer.com
Savings Account	"Would you like to try our new savings account?"	Savings Account offer	http://www.offer.com

4.1.2 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 Open the Choice editor.

2 On the **Scores** tab of each product, do the following:

- Use **Select Metrics** to add the 'Cost' performance metric. Using **Score** assign the following constant costs to each product.

Choice	**Cost**
Brokerage Account	800
Credit Card	200
Life Insurance	800
Roth IRA	765
Savings Account	150

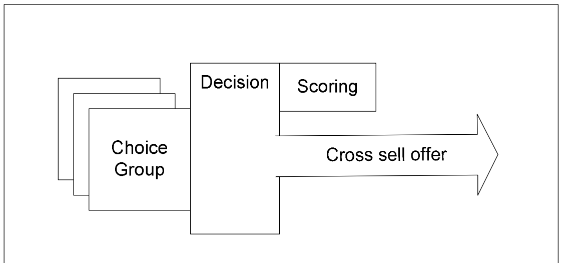
Since our Performance Goal is to minimize costs, it is obvious that the Savings Account offer will be chosen each time. In another step we will add another Performance Goal 'Maximize Revenue' to see how these two competing performance metrics are optimized by the platform.

4.1.3 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 by Siebel RTD.



Structure of Get Cross Sell Offer Advisor

4.1.4 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 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 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 Siebel RTD was not installed. For example, in a cross-selling application for a call center, if agents randomly selected an offer before Siebel RTD was introduced, then the Control Group Decision should return a random selection.

4.1.5 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.** Under **Order** enter 2.

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 Inline Service Explorer, select **Informant** and then the **End of Call** informant. On the **Request** tab, under **External System** select **CRM system.** Under **Order** enter 1.

As part of this more complete process we will first call the End of Call informant and then we decide on a Cross Selling Offer. This reflects the business process where the Agent finished processing the call in the normal way (End of Call) and then when wrapping up the call takes the opportunity to present an offer to the customer.

7 Save the Inline Service. Uncheck **Terminate Active Sessions (used for testing)** before deploying, because we now want to have the session stay active. Check **Release Inline Service Lock** so that the Inline Service can be edited through Decision Center. Deploy.

4.1.6 Viewing the integration map

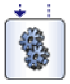
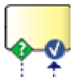
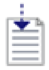
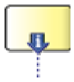
1 Using the **Start** menu, open the **Decision Center** from the **Siebel Analytics→RTD** program group**.** The Siebel Real-Time Decision Server must be started for Decision Center to run.

2 Login then click on 'Open an Inline Service'.

3 Choose the Tutorial Inline Service.

4 On the left hand tree, click on Tutorial. In the right pane, on the **Definition** tab, bok at the **Integration Map** subtab.

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

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

4.1.7 Testing the advisor

1 In Studio use the **Test** view to send a request integration point. Select **End of Call** informant and fill in some values for the parameters.

2 Click **Send** ▶ and confirm in the **Log** subtab that the message was sent.

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 offer selected and its attributes are returned and displayed in the **Response** pane. Notice that the same offer is returned. Since we have optimized only on the 'Cost' performance metric, the lowest cost offer is always chosen at this point.

4.1.8 Updating the load generator script

1. Using the **Start** menu, open **Load Generator** by selecting it from the **Siebel Analytics→RTD** program group. Open the previous script.
2. 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 'GetCrossSell Offer'.
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 on **Variable** for the input field and use the pull down to choose the matching variable.
5. Mark **customerId** as a key by checking the box under **Session Key**.

   This defines a session as a list of two Integration Points, first the EndofCall informant and then the GetCrossSellOffer advisor. You can now run the script to see that things are working.
6. Once again, save the load generator configuration script.
7. Go to the **Run** tab and press the **Play** ▶ button.

4.2 Redeploying from Decision Center

You can make certain Inline Service changes from Decision Center and re-deploy to the Siebel Real-Time Decision Server. These changes include:

1. Adjust Filtering and Scoring Rules.
2. Adjust weighting for Performance Metrics on Decisions.
3. Add and remove segments that are targeted.
4. Create alerts that go to a user's email.
5. Modify the eligibility of choices and choice groups.
6. Change the state of an Inline Service on re-deployment.

We will use Decision Center to add a new Rule on a Choice for Reason Analysis.

4.2.1 Making changes from Decision Center

Business users can make changes from Decision Center. In this example, we will add a rule on one of our choices, Credit Card. This rule will make the Credit Card offer eligible only for customers that are 18 or older that have called to Check Balance or Make Payment.

1. Login to Decision Center. You will be in the default perspective, Explore.
2. Use the **Perspective** button on the taskbar to change to the 'Design' perspective. For more about Perspectives, see *Decision Center User's Guide.*
3. Go to the **Decision Process** and expand the **Cross Selling Offer** Choice Group. The report for the Choice Group appears in the right pane of Decision Center.
4. Choose the **Credit Card** choice. From the **Choice Eligibility** subtab click on the **Add Rule** button. A two-side rule will be added. Click on the left side and then on the ellipsis .... **Edit Value** appears.
5. Select the **Attribute** and then choose Session →customer→age. Choose data type Integer. Click **OK**.

6 Click on the operator = and click on the lower right corner and select **>=**. In the right side enter the number 18.

7 Click the **Add Ruleset** button. A new group of rules appears in the expression. By default it is an **All of the following** expression. Click on **All of the following** at the right corner and change it to **Any of the following**.

8 Click on the left side and then on the ellipsis .... **Edit Value** appears.

9 Select the **Attribute** option and then choose **Session → call → reason code**. Choose data type Integer. Click **OK**. On the right hand side enter 17.

10 Select **Ruleset 2** and click on the **Add Rule** button. A two-side rule will be added to the ruleset. Click on the left side and then on the ellipsis .... **Edit Value** appears.

11 Select the **Attribute** option and then choose **Session → call → reason code**. Choose data type Integer. Click **OK**. On the right hand side enter 18.

12 Save the Rule using the Disk icon.

13 From the toolbar, use the Redeploy icon to deploy to the Siebel Real-Time Decision Server. Check the **Continue editing** box and redeploy to the **QA** state.

Section 5: Closing the feedback loop

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 Siebel 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 there is 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, Siebel 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'.
3. For each event set the **Statistic Collector** to 'Choice Event Statistic Collector' using the pull down menu. 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.

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**, right click and use the menu to **New Choice Event Model.** Call the new model 'Offer Acceptance Predictor' Click **OK.**

2 In the Editor, uncheck the box **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'.

This is the event from which we want to measure the success. We want to track whether an offer was accepted or not after it is 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 3 possible values and another with 4 possible values will use 12 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 Observer model that having the code for the call reason as input was the reason for call created a completely correlated relationship in our model.

It should be noted that the reason code is an important factor for other models, like the offer acceptance model therefore it should not be ignored altogether from all models.

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 offer 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** and select it.
2. Use **Add Attribute** to add an attribute named 'Offer Extended'.
3. Enter a description. Un-check the **Show in Decision Center** and **Use for Analysis** checkboxes. 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 **Select Offer** under **Decisions.** Go to the Editor.
5. On the **Pre/Post Selection** tab **under Post Selection Logic** enter the following code:

```
if (choices.size() > 0) {
  Choice ch = choices.get(0);
  ch.recordEvent("presented");
  session().setOfferExtended(ch.getSDOId());
}
```

This will assign the SDOId of the selected choice to the OfferExtended attribute of the session. The SDOId is a unique identifier. Every object in a Siebel RTD configuration has a unique SDOId. It will also record the Presented event for the selected offer.

5.1.8 Creating the feedback informant

This informant provides Siebel 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 3.
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.

```
if ("yes".equals(request.getPositive())) {
  String extendedID = session().getOfferExtended();
  if (extendedID != null) {
    Choice offer = CrossSellingOffer.getChoice(extendedID);
    if (offer != null)
      offer.recordEvent("accepted");
  }
}
```

6 Save all and re-deploy the Inline Service. On the **Deploy** dialog, check **Terminate Active Sessions (used for testing).**

5.1.9 Testing the feedback informant

You can send integration point requests from Studio and simulate a session and check that the logic works correctly. You can also use `logInfo` calls to follow the execution.

5.1.10 Modifying the load generator script for feedback

Change the load generator script to include a call to the feedback informant. Since we can only send random data, we will not be able to create any interesting correlations between offer acceptance and the characteristics of the customers or of the calls.

1 Using the **Start** menu, open **Load Generator** from the **Siebel Analytics→RTD** program group.

2 Open the saved load generator script. In the **Variables** section select **Message** and right-click and use **Add variable** to add a new variable. Call it 'isPositive'. In Contents, select 'String Array' from the menu.

3 Right-click in **String** and use **Add item** to add two values, "no" and "yes".

4 Verify that all variables are set correctly. Save the script.

5 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**.

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

7 Click on **Variable** for the input field and use the pull down to choose the matching variable.

8 Mark **customerId** as a key by checking the box under **Session Key**.

9 In **Input Fields,** right-click and chose **Add item** to add an input field. Click in the space under **Name** and add 'positive'**.**

10 Click on **Variable** for the input field and use the pull down to choose the newly created variable, 'isPositive'.

11 Save your changes.

Run the script. After a while look at the results in Decision Center. You should see that the count for accepted is roughly half of the presented, as we have a 50% of feedback with "yes".

When running this Inline Service with real data, the Decision Center reports would be a valuable tool to explore the relationship between the characteristics of the customers and calls and the acceptance of offers. In the next section we will explore using these models to automatically improve the targeting of offers.

5.2 Using the predictive power of models

The model we have created learns the correlations between the characteristics of the customers and the calls and the cross selling results. This model can be used in a predictive fashion, to predict the likelihood an offer will be accepted. To do so we will replace the random selection function with one based on the predicted likelihoods.

Once again we will write the selection function from scratch even though it is found in most of the templates.

5.2.1 Computing likelihood for choices

Choice Event Models can be directly used to predict the likelihood of an event for each specific choice in a given situation. The easiest way to achieve this is to define a choice attribute that is computed from the prediction in the model.

1 In the Inline Service Explorer, select the **Cross Selling Offer** choice group. In the Editor, use **Add** adjacent to Choice Attributes to add a choice attribute. Name it 'likelihood of acceptance'.

2 Make it of type 'Double'. Click **OK.**

3 Click on the ellipsis ... in the **Value** column for this attribute and select the value as **Model Prediction**.

4 Under **Value is a likelihood predicted** choose 'Offer Acceptance Predictor' for **By Model** and 'Accepted' for **By Event**.

5 Click **OK**.

5.2.2 Adding a choice attribute

Since the performance metric 'Revenue' is based on profits, we need a profit margin attribute on each of the cross sell offers.

1 Choose the 'Cross Selling Offers' choice group from the Inline Service Explorer. Use **Add** next to **Choice Attributes** to add an attribute. Name it 'Profit Margin' make it of type double.

2 Set a default value for 'Profit Margin' at the group level. Click under **Value** to add a value 0.5. This will be inherited and overridden at the Choice level, but can be used in case no value is available.

3 For each of the choices, add a Profit Margin value at the Choice level.

Choice	Profit Margin
Credit Card	0.33
Savings Account	0.25
Life Insurance	0.65
Roth IRA	0.45
Brokerage Account	0.56

5.2.3 Checking the likelihood

To explore the kind of values we get for the likelihood, add a print statement in the Advisor.

1 In the Inline Service Explorer, expand the **Integration Points** and select the **Get Cross Sell Offer** Advisor. In the Editor and add two lines to the Asynchronous Logic code:

```
if (choices.size() > 0) {
  Choice ch = choices.get(0);
  ch.recordEvent("presented");
  session().setOfferExtended(ch.getSDOId());
  CrossSellingOfferChoice cso = (CrossSellingOfferChoice) ch;
  logInfo("Likelihood = " + cso.getLikelihoodOfAcceptance());
}
```

2 Save and deploy the Inline Service.

3 Send the events from Studio and make sure that the output in the server's screen reflects the likelihood.

Make sure you send the EndofCall informant first and then the Get Cross Sell Offer Advisor. The output should have a likelihood value in the server command window. If the model does not have enough data to compute the likelihood it will return a numerical NaN – meaning 'Not a Number'.

5.2.4 Adding a second performance goal

We will use the likelihood of acceptance and the profit margin of the choice to achieve scoring that will support the maximization of the revenue performance metric. The formula for this is: (profit margin) * (likelihood of acceptance) = potential revenue score.

1 Double-click on the **Performance Goal** to open the editor. Use the **Add** button to add a Performance Metric. Name the metric 'Revenue'. 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 In **Inline Service Explorer,** select the 'Select Offer' Decision.

4 Click **Goals** to add the 'Revenue' performance metric. Click **OK.** The default is evenly split weighting between all performance metrics. If you wanted the Revenue performance goal to take precedence over Cost, you could adjust the percentages so that it had more weight.

5.2.5 Scoring the revenue performance metric

1 On the Cross Selling Offer choice group, choose the score tab. Use **Select Metrics** to add the 'Revenue' performance goal.

2 'Revenue' for all offers is computed the same way, by multiplying Likelihood of Acceptance by the Profit Margin for the product. Click in the **Score** column for 'Revenue' and then on the ellipsis ... to bring up the **Edit** window.

3 Choose **Function Call.** Under **Function to Call** choose the function 'Multiply'. In the parameters to multiply, click in the **Value** cell. Click the ellipsis ... and choose **Attribute or variable** then **Choice** then **Profit Margin.** In the second argument click in the **Value** cell. Click the ellipsis ... and choose **Attribute or variable** then **Choice** then **likelihood of acceptance.**

4 Save all and then redeploy the Inline Service.

5.2.6 Modifying Load Generator to include predictive offer selection

Start the load generator script. After about 500 sessions, pause the script using the pause button. Look at the server's output. You will see that the likelihood is NaN for all choices.

Now continue running the script. After a few thousand sessions pause again and look at the likelihoods. You will see that they now are real numbers, varying around 50%. Remember that we set the feedback informant with 50% of the cases returning "yes".

5.2.7 About randomized likelihood

You may be asking yourself how come the system did not get into local maxima where one offer had the best likelihood. The answer is in one of the checkboxes of the definition of the model.

In Studio go back to the **Offer Acceptance Predictor** model and see that the checkbox labeled **Randomize likelihood** is checked. This will introduce a certain degree of randomness to the computed likelihoods. This introduces enough noise into the system to disturb it and move it from local maxima.

5.2.8 Studying the results

Use Decision Center to explore the results. You will see an imbalance in the number of time the Credit Card offer was extended. This can be easily explained by the fact that this offer is restricted by the eligibility rule.