Progress. | Corticon.

PROGRESS®
CORTICON®

Corticon Studio Tutorial: Advanced Rule Modeling

PROGRESS
software

Notices

Copyright agreement

See Table of Contents for location of Third party acknowledgements within this documentation.

Table of Contents

Chapter 5: Modeling and testing the 'coupons' Rulesheet......................47

Chapter 6: Modeling and testing the 'use_cashBack' Rulesheet............69

Preface

For details, see the following topics:

- Progress Corticon documentation
- Overview of Progress Corticon

Progress Corticon documentation

The following documentation, as well as a *What's New in Corticon* document, is included with this Progress Corticon release:

Corticon Tutorials	
Corticon Studio Tutorial: Basic Rule Modeling	Introduces modeling, analyzing, and testing rules and decisions in Corticon Studio. Recommended for evaluators and users getting started. *See also the PowerPoint-as-PDF version of this document that is accessed from the Studio for Analysts'* ***Help*** *menu.*
Corticon Studio Tutorial: Advanced Rule Modeling	Provides a deeper look into Corticon Studio's capabilities by defining and testing vocabularies, scope, collections, messages, filters, conditions, transient data, and calculations in multiple rulesheets that are assembled into a Ruleflow. *See also the PowerPoint-as-PDF version of this document that is accessed from the Studio for Analysts'* ***Help*** *menu.*
Corticon Tutorial: Using Enterprise Data Connector (EDC)	Introduces Corticon's direct database access with a detailed walkthrough from development in Studio to deployment on Server. Uses Microsoft SQL Server to demonstrate database read-only and read-update functions.

Corticon Studio Documentation: Defining and Modeling Business Rules	
Corticon Studio: Installation Guide	Step-by-step procedures for installing Corticon Studio and Corticon Studio for Analysts on computers running Microsoft Windows. Also shows how use other supported Eclipse installations for integrated development. Shows how to enable internationalization on Windows.
Corticon Studio: Rule Modeling Guide	Presents the concepts and purposes the Corticon Vocabulary, then shows how to work with it in Rulesheets by using scope, filters, conditions, collections, and calculations. Discusses chaining, looping, dependencies, filters and preconditions in rules. Presents the Enterprise Data Connector from a rules viewpoint, and then shows how database queries work. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting and many *Test Yourself* exercises.
Corticon Studio: Quick Reference Guide	Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.
Corticon Studio: Rule Language Guide	Reference information for all operators available in the Corticon Studio Vocabulary. A Rulesheet example is provided for many of the operators. Includes special syntax issues, handling arithmetic and character precedence issues.
Corticon Studio: Extensions Guide	Detailed technical information about the Corticon extension framework for extended operators and service call-outs. Describes several types of operator extensions, and how to create a custom extension plug-in.
Corticon Server Documentation: Deploying Rules as Decision Services	
Corticon Server: Deploying Web Services with Java	Details installing the Corticon Server as a Web Services Server, and then and deploying and exposing Decision Services as Web Services on Tomcat and other Java-based servers. Presents the features and functions of the browser-based Server Console.
Corticon Server: Deploying Web Services with .NET	Details installing the Corticon Server as a Web Services Server and deploying and exposing decisions as Web Services with .NET. Provides installation and configuration information for the .NET Framework and Internet Information Services (IIS) on various supported Windows platforms.
Corticon Server: Integration & Deployment Guide	An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the Deployment Console tool. Discusses relational database concepts and implementation of the Enterprise Data Connector. Goes deep into the server to discuss state, persistence, and invocations by version or effective date. Includes samples, server monitoring techniques, and recommendations for performance tuning.

Overview of Progress Corticon

Progress® Corticon® is the Business Rules Management System with the patented "no-coding" rules engine that automates sophisticated decision processes.

Progress Corticon products

Progress Corticon distinguishes its development toolsets from its server deployment environments.

- **Corticon Studios** are the Windows-based development environment for creating and testing business rules:
  - **Corticon Studio for Analysts.** is a standalone application, a lightweight installation that focuses exclusively on Corticon.
  - **Corticon Studio** is the *Corticon Designer* perspective in the **Progress Developer Studio** (PDS), an industry-standard Eclipse 3.7.1 and Java 7 development environment. The PDS enables integrated applications with other products such as Progress OpenEdge and Progress Apama.

  The functionality of the two Studios is virtually identical, and the documentation is appropriate to either product. Documentation of features that are only in the *Corticon Designer* (such as on integrated application development and Java compilation) will note that requirement. Refer to the *Corticon Studio: Installation Guide* to access, prepare, and install each of the Corticon Studio packages.

  Studio Licensing - Corticon embeds a time-delimited evaluation license that enables development of both rule modeling and Enterprise Data Connector (EDC) projects, as well as testing of the projects in an embedded Axis test server. You must obtain studio development licenses from your Progress representative.

- **Corticon Servers** implement web services for business rules defined in Corticon Studios:
  - **Corticon Server for deploying web services with Java** is supported on various application servers, and client web browsers. After installation on a supported Windows platform, that server installation's deployment artifacts can be redeployed on various UNIX and Linux web service platforms as Corticon Decision Services. The guide *Corticon Server: Deploying web services with Java* provides details on the full set of platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.
  - **Corticon Server for deploying web services with .NET** facilitates deployment of Corticon Decision Services on Windows .NET Framework 4.0 and Microsoft Internet Information Services (IIS). The guide *Corticon Server: Deploying web services with .NET* provides details on the platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.

  Server Licensing - Corticon embeds a time-delimited evaluation license that enables evaluation and testing of rule modeling projects on supported platform configurations. You must obtain server deployment licenses and server licenses that enable the Enterprise Data Connector (EDC) from your Progress representative.

1

Introduction

This is the Corticon *Advanced Rule Modeling Tutorial*. You can start to learn about Corticon Studio in this guide, but you might want to start with the *Basic Rule Modeling Tutorial*, an introduction to the Corticon Studio. The basic tutorial walks through how to capture rules from business specifications, model the rules, analyze them for logical errors, and then test the execution of your rules.

This guide presents the concepts underlying some of Corticon Studio's more complex and powerful functions, including:

- Using **Scope** and **Defining Aliases** in rules
- Understanding **Collections**
- Using **String**, **DateTime**, and **Collection** operators
- Modeling formulas and equations in rules
- Using **Filters**
- **Sequencing Rulesheets** in a Ruleflow
- Testing at rule, Rulesheet, and Ruleflow levels.

The Ruleflows that you build using Corticon Studio will be deployable as executable, standards-based Decision Services that can be used by other software applications via Java Messaging or XML Web Services.

See the **Tutorials for Corticon Server**, *Deploying Web Services with Java* and *Deploying Web Services with .NET*, for straightforward instructions for installing and configuring the server, and for deploying your Ruleflows as Decision Services.

In this tutorial, you start with the Corticon tool set to build a Vocabulary, create three Rulesheets that are tested in Ruletests, and then assemble them into a Ruleflow.

Note: See the *Corticon Studio: Installation Guide* for instructions on downloading and installing Corticon Studios.

2

Building the Vocabulary

This chapter shows how to build the Vocabulary for the grocery store tutorial.

For details, see the following topics:

A scenario's business requirements

The Business Problem

Describing a business problem in natural language might proceed as follows.

Scenario

The management of a chain of grocery stores intends to create and deploy a system of business rule-based "smart" cash registers in each of its branches. Some branches are large supermarkets, and some are small convenience stores, which sell gasoline and other essentials.

In addition to minimum cash register functionality (adding up the price of items in a customer's shopping cart, for example) the new system will also include the ability to apply promotional rules - rules that determine coupon generation, loyalty program rules, and special warning rules to alert the cashier to take certain actions. Because every item in every store has a bar-coded label, the system's scanner will be able to determine complete information about each item, such as which department the item comes from.

To foster customer loyalty and drive additional sales, a "Preferred Shopper" program will be launched in conjunction with the installation of the new business rule-based cash registers. Shoppers who enroll in the program will be issued Preferred Shopper membership cards (one card per household) to present to the cashier at check-out time. Benefits of the Preferred Shopper program include:

- A Preferred Shopper earns 2% cash back on all purchases at any branch
- The Preferred Shopper account will track the accumulated cash back and allow the shopper to apply it to any visit's total amount. The cashier will ask a Preferred Shopper if they would like to apply a cash back balance to the current purchase
- Once a Preferred Shopper chooses to apply his cash back balance, the cumulative cashback amount maintained by the system is reset to zero, and the accumulation of cashback begins with the customer's next purchase.
- A Preferred Shopper will be eligible for special promotions and coupons as follows:
    1. For every item purchased from the floral department, a never-expiring coupon for a free balloon.
    2. When three or more soda or juice items are purchased in a shopping cart, a coupon for $2 off a future purchase within the next year.
    3. When the shopping cart total is at least $75, a coupon for 10% off a gasoline purchase within three months at any of their stores.

To comply with government liquor laws, the chain needs to confirm that a shopper with any liquor items at check-out is of legal age. Alerting the cashier of a liquor item will call for checking the shopper's identification.

Identifying the terms

Identifying the terms

The scenario used terms and relationships of terms that define the business rules.

Highlighting the terms in the scenario

The management of a chain of grocery stores intends to create and deploy a system of business rule-based "smart" cash registers in each of its branches. Some branches are large supermarkets, and some are small convenience stores, which sell gasoline and other essentials.

In addition to minimum cash register functionality (adding up the **price of items** in a **customer**'s **shopping cart**, for example) the new system will also include the ability to apply promotional rules - rules that determine coupon generation, loyalty program rules, and special warning rules to alert the cashier to take certain actions. Because every **item** in every store has a **bar-coded label**, the system's scanner will be able to determine complete information about each item, such as which **department** the item comes from.

To foster customer loyalty and drive additional sales, a "**Preferred Shopper**" program will be launched in conjunction with the installation of the new business rule-based cash registers. Shoppers who enroll in the program will be issued Preferred Shopper **membership card**s (one card per household) to present to the cashier at check-out time.

Benefits of the Preferred Shopper program include:

- A Preferred Shopper earns 2% **cash back** on all purchases at any branch
- The Preferred Shopper **account** will track the **accumulated cash back** and allow the shopper to apply it to any visit's **total amount**. The cashier will ask a Preferred Shopper if they would like to apply a **cash back balance** to the current purchase
- Once a Preferred Shopper chooses to apply his cash back balance, the cumulative cashback amount maintained by the system is reset to zero, and the accumulation of cashback begins with the customer's next purchase.
- A Preferred Shopper will be eligible for special promotions and **coupons** as follows:
    1. For every item purchased from the floral department, a never-**expiring** coupon for a free balloon.
    2. When three or more soda or juice items are purchased in a shopping cart, a coupon for $2 off a future purchase within the next year.
    3. When the shopping cart total is at least $75, a coupon for 10% off a gasoline purchase within three months at any of their stores.

To comply with government liquor laws, the chain needs to confirm that a shopper with any liquor items at check-out is of **legal age**. Alerting the cashier of a liquor item will call for **checking the shopper's identification**.

Grouping the terms

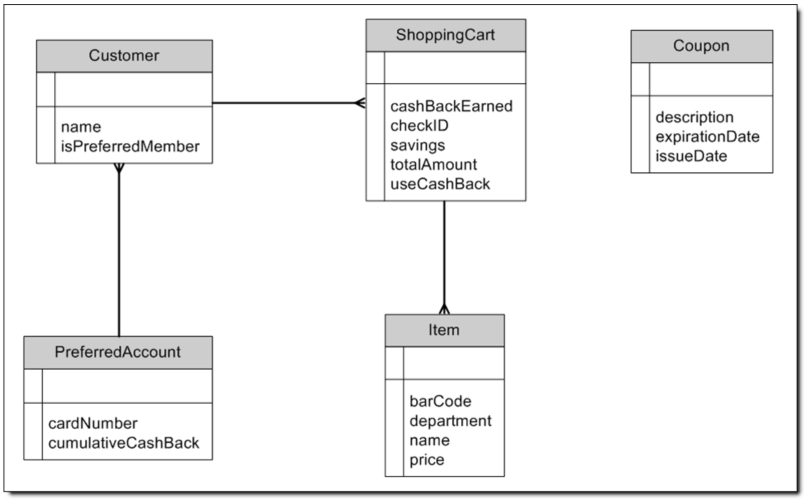
Compiling a list of terms based on our findings within the previous slide, the following assumptions can be made and can be used to build a **Fact Model** or an **ER Diagram**.

- A Customer has a Name
- A Customer may be a Preferred Shopper and have a Preferred Shopper account that is identified by showing their Preferred Card at checkout
- A Preferred Shopper account has a Card Number
- A Preferred Shopper account holds a Cash-Back Balance
- One Preferred Shopper account may be used by anyone in that family
- An Item has a Name
- An Item has a Price
- An Item has a Bar-coded Label

- An Item is located in a Department
- A Shopping Cart contains the Items a Customer purchases during each visit
- A Customer uses a Shopping Cart to carry items
- A Shopping Cart has a Total Amount
- A Cash-Back Bonus is calculated using the Shopping Cart's Total Amount and is deducted from the Total Amount upon Customer request
- Coupons are issued to shoppers
- A Coupon has a Description
- A Coupon has an Issue Date
- A Coupon has an Expiration Date

Using an Entity Relationship diagram

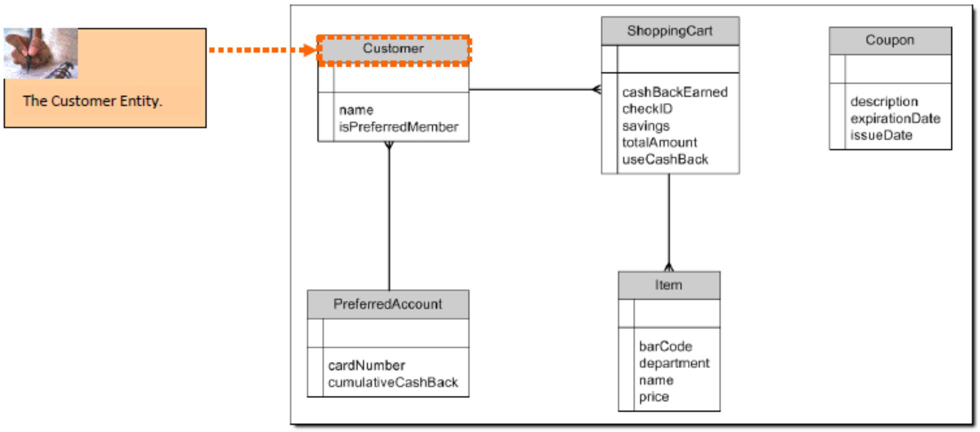
You can use an **Entity-Relationship (ER) Diagram** or a Fact Model (FM) to provide a graphical depiction of the entities and their respective attributes, as well as the associations (relationships) between entities. For very large and complex vocabularies, you might find an advantage in using ER or FM to define the model. This functionality is not a component of this product, yet it serves well to visualize the entities, attributes, and associations used in the vocabulary of this exercise.



Define entities

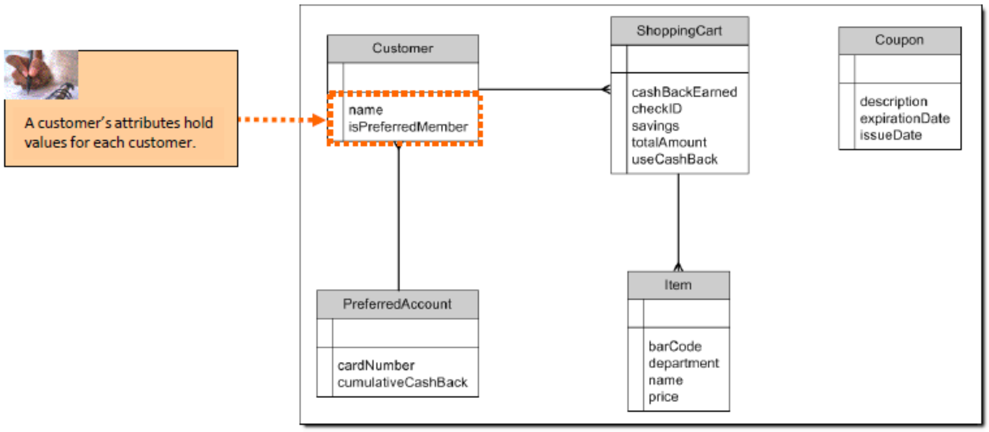
Entities

As the ER diagram illustrates, we need to build **Customer**, **Coupon**, **ShoppingCart**, **Item**, and **PreferredAccount** (our main business terms) into our Vocabulary.



Add attributes to the entities

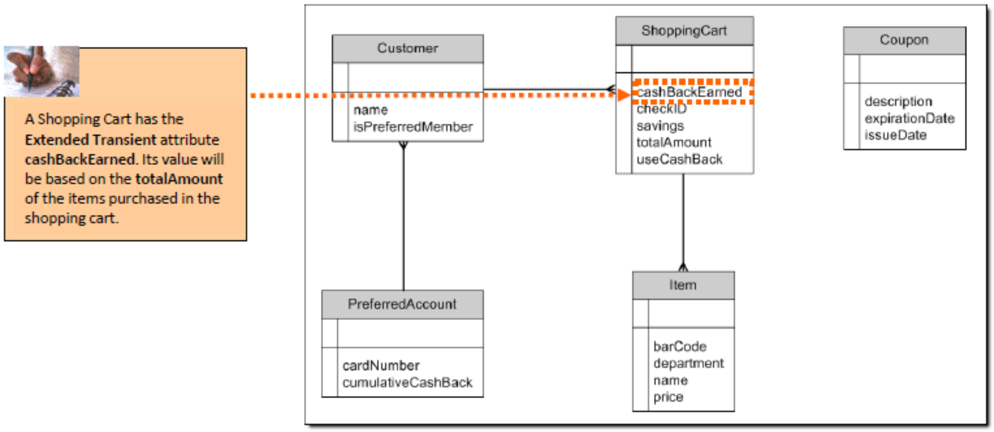
Entities have attributes - properties or characteristics that distinguish them from another entitiy, and whose values distinguish one instance of an entity from another instance of the same entity. As there will be *many* customer, we distinguish them by defining attributes that hold the values of each customer's identifier, such as their name.



Extended transient attributes

Some attributes are "intermediate" or "temporary" value holders. We do not need to return these values in a response, or save them in a database. In Corticon Studio, **Extended Transient** attributes fill this purpose. Because an extended transient is not part of the Decision Service's response message, its presence (or absence) in the Vocabulary or rules does not affect the technical integration with the Decision Service in runtime. Therefore, a Rule Modeler may add/remove extended transients to/from the Vocabulary without concern of upsetting the runtime integration.

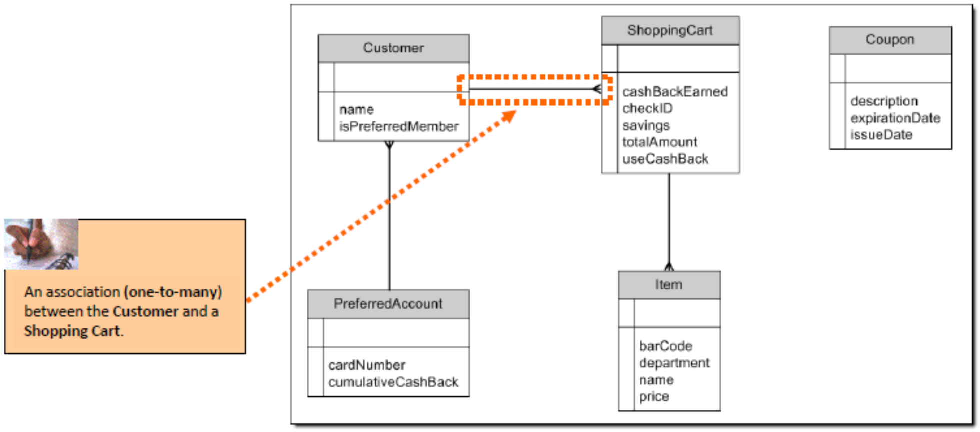
In our example, the **cashBackEarned** attribute will serve as an intermediate value, helping to calculate other attribute values that *will* be included in the Decision Service's response message (**Base** attributes).



A Shopping Cart has the **Extended Transient** attribute **cashBackEarned**. Its value will be based on the **totalAmount** of the items purchased in the shopping cart.

Define associations between entities

Associations between entities allow us to define relationships between them. In our example, each individual **Customer** will have his/her own shopping cart, most likely with different items in each cart. How do we distinguish between them? By associating a unique instance of a **shopping cart** with each **Customer** who visits our store. Over successive visits, a customer may have several shopping carts, each with many items.



An association **(one-to-many)** between the **Customer** and a **Shopping Cart**.

Organizing the terms

The terms that you would use to build a **Fact Model** or **Entity Relationship Diagram** translate to the terms that you will use in the **Vocabulary**:

Table 1: Entities and their attributes in the `groceryStore` vocabulary

Entity: Name	Attribute: Name	Data Type	Mandatory	Mode
Customer				
	name	String	No	Base

Entity: Name	Attribute: Name	Data Type	Mandatory	Mode
	isPreferredMember	Boolean	No	Extended Transient
Item				
	barCode	String	No	Base
	department	String	No	Base
	name	String	No	Base
	price	Decimal	No	Base
ShoppingCart				
	cashBackEarned	Decimal	No	Extended Transient
	checkID	Boolean	No	Base
	savings	Decimal	No	Base
	totalAmount	Decimal	No	Base
	useCashBack	Boolean	No	Base
PreferredAccount				
	cardNumber	String	No	Base
	cumulativeCashBack	Decimal	No	Base
Coupon				
	description	String	No	Base
	expirationDate	Date	No	Base
	issueDate	Date	No	Base

where:

- **Name** is the case-sensitive, no spaces, alphanumeric identifier of the attribute. It must be unique within its entity.
- **DataType** indicates how data is validated, manipulated and interpreted, **String** is text while other types are the true/false type (**Boolean**), math types (**integer** and **decimal**), and timespan types (**Time**, **Date**, and **DateTime**).
- **Mandatory** indicates whether the attribute is required to have a value. In our examples, none of the attributes are mandatory.

- **Mode** indicates how the attribute will be handled. Most attributes use **Base** mode because their values will be sent in to the rules or returned from the rules. In other words, base attributes are what carry values to and from the client application. **Extended Transient** mode is used when an attribute's value is assigned or derived by rules, but *not* sent in from the client application, or back to the client application. You will see examples of extended transient attributes in this tutorial.

Ready to implement the Vocabulary design

That completes the information needed to build a Vocabulary that incorporates the key facts and relationships for the tutorial.

You might want to learn more about building vocabularies in the *Progress Corticon Quick Reference Guide* and the *Progress Corticon Rule Modeling Guide*.

Note: If you have not yet installed Corticon Studio, see the *Corticon Studio: Installation Guide* for instructions on downloading and installing Corticon Studios.

Creating the Vocabulary for the tutorial

The following procedures provide a straightforward path to setting up the Vocabulary for the scenario:

1. Creating a project and a Vocabulary file
2. Adding entities to the Vocabulary
3. Adding each entity's attributes, including their data types and mode
4. Adding associations between entities

Creating the Vocabulary file

The first step is to create a project, and then its Vocabulary file, as follows:

1. Launch Corticon Studio.
2. In the **Corticon Designer** perspective, select the menu command `File > New > Rule Project`. Name the project `MyAdvancedTutorial`, and then click `Finish`.
3. Select the menu command **File > New > Rule Vocabulary**. Choose `MyAdvancedTutorial` as as the parent folder, name the Vocabulary file `groceryStore`, and then click **Finish**.

Adding entities to the Vocabulary

Add all the entities for the tutorial, as follows:

1. In the groceryStore.ecore panel, right-click on groceryStore, and then choose **Add Entity**.
2. Enter the Entity Name Customer.
3. Repeat these steps to add Entities named Item, ShoppingCart, PreferredAccount, and Coupon.

Adding attributes to the entities

Note: In this example, all the attributes accept the **Mandatory** property's value default **No**.

Add the attributes of each entity, as follows:

1. Right-click on Customer, choose Add Attribute.
   a) Enter the **Attribute Name** (case-sensitive).
   b) Select the **DataType** pulldown if you need to change it from the default value String.
   c) Select the **Mode** pulldown if you need to change it from the default value Base.
2. Repeat the previous step to create the next Customer Attribute, isPreferredMember, and its values.
3. Repeat these steps to add the attributes of each of the entities listed in Organizing the terms on page 20

Adding associations between entities

The tutorial focuses on the Customer root-level entity, and the associations between it and the preferredAccount, Shopping Cart and Item entities. You will see why this particular perspective or view of our Vocabulary is appropriate.

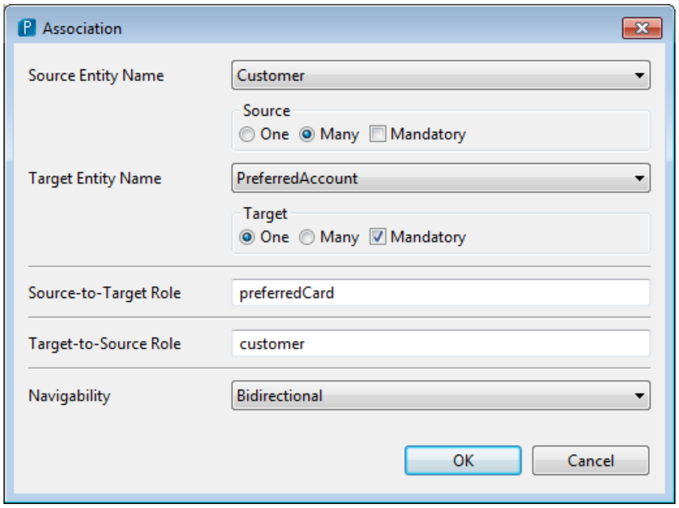
Note that role names are used, such as preferredCard for the association from Customer to preferredAccount. While role names are optional, they help distinguish relationships between entities.

The required associations are (1) many customers can be using one preferred account, (2) each customer could have many shopping carts, and (3) each shopping cart could contain many items.

Add the associations between entities, as follows:

1. Right-click on Customer, choose Add Association.

   Complete the dialog to associate many customers with one preferred account, as shown:



Association

Source Entity Name: Customer

Source: ○ One ● Many □ Mandatory

Target Entity Name: PreferredAccount

Target: ● One ○ Many ☑ Mandatory

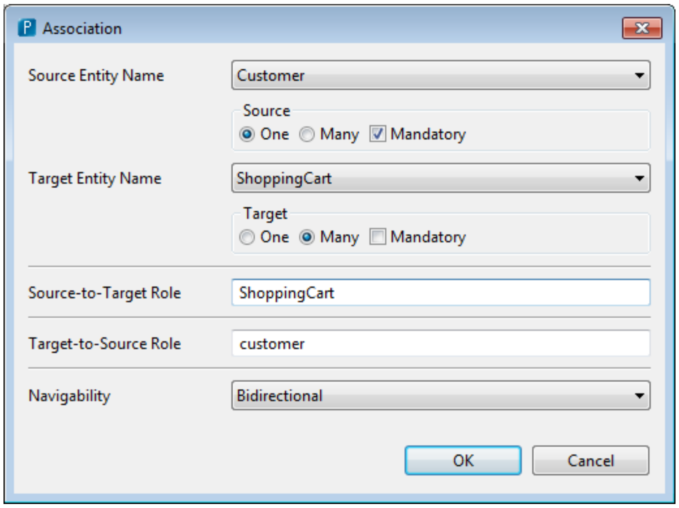
Source-to-Target Role: preferredCard

Target-to-Source Role: customer

Navigability: Bidirectional

2. Right-click on `Customer`, choose `Add Association`.

   Complete the dialog to associate each customer with many shopping carts, as shown:



Association

Source Entity Name: Customer

Source: ● One ○ Many ☑ Mandatory

Target Entity Name: ShoppingCart

Target: ○ One ● Many □ Mandatory

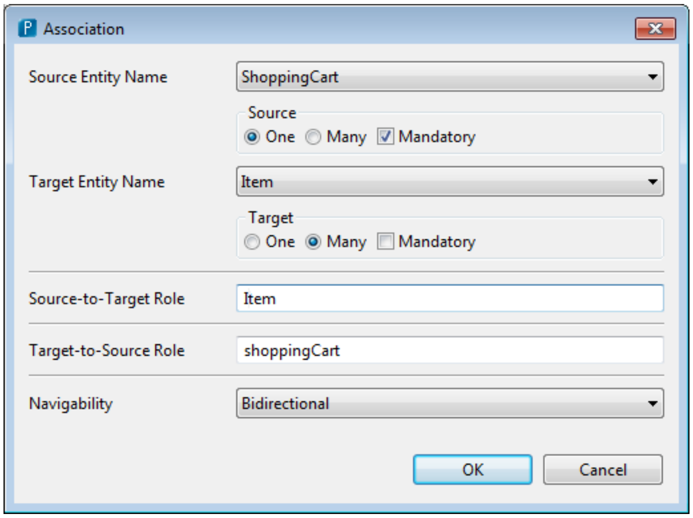
Source-to-Target Role: ShoppingCart

Target-to-Source Role: customer

Navigability: Bidirectional

3. Right-click on `ShoppingCart`, choose `Add Association`.

   Complete the dialog to associate many items with one shopping cart, as shown:



Source Entity Name: ShoppingCart

Source: ● One ○ Many ☑ Mandatory

Target Entity Name: Item

Target: ○ One ● Many □ Mandatory

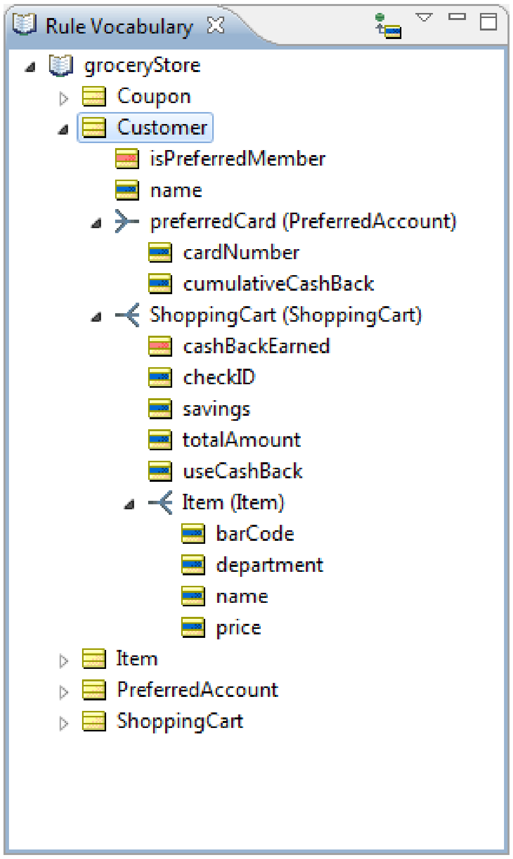
Source-to-Target Role: Item

Target-to-Source Role: shoppingCart

Navigability: Bidirectional

The completed Vocabulary for the tutorial

The **Vocabulary** for this tutorial is now defined, and looks like this when **Customer** is expanded:



```
groceryStore
  Coupon
  Customer
    isPreferredMember
    name
    preferredCard (PreferredAccount)
      cardNumber
      cumulativeCashBack
    ShoppingCart (ShoppingCart)
      cashBackEarned
      checkID
      savings
      totalAmount
      useCashBack
      Item (Item)
        barCode
        department
        name
        price
  Item
  PreferredAccount
  ShoppingCart
```

3

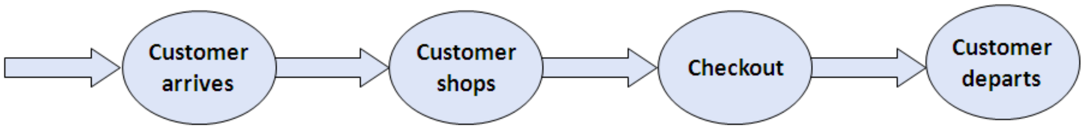
Business process and rules

This chapter discusses business processes and rules.

For details, see the following topics:

- The high-level business process
- The low-level process & rules
- Organizing the business rules

The high-level business process



At a high level, this is the basic process followed by every customer making purchases at a store.

While there may be several steps involved in this process, rule modelers are most concerned with those steps where decisions are made. In this case, the **Checkout** step contains the rule-based decisions that are built into the store's cash registers.

The low-level process & rules

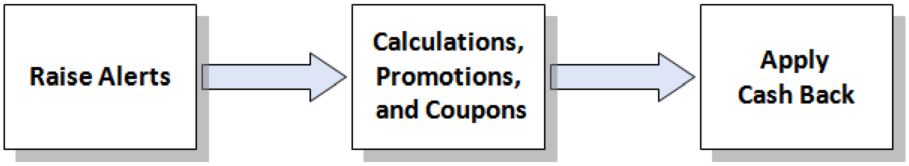
The **Checkout** step is where our rules will be applied.

Where a natural sequence or "flow" of logical steps can be identified within a single decision step, it often makes sense to organize the steps using separate Rulesheets for each logical step, then combine them into a "Ruleflow".

We will do that in this Scenario.

According to the Scenario, there are a few general categories of activity performed by rules during the **Checkout** step:

1. Identify warning/alert situations.
2. Calculate totals, apply promotions and generate coupons.
3. Apply cash back (when applicable).



Organizing the business rules

First Rulesheet: Raise Alerts

- Liquor purchases (any items from the Liquor department) can only be made by shoppers of legal age

Second Rulesheet: Calculations, Promotions, and Coupons

- Preferred Shoppers receive a coupon for one free balloon for every item purchased from the Floral department. Expiration date: none
- Preferred Shoppers receive a coupon for $2 off their next purchase when 3 or more Soda/Juice items are purchased in a single visit. Expiration date: one year from date of issue
- Preferred Shoppers receive a coupon for 10% off their next gasoline purchase at any chain-owned convenience store with any purchase of $75 or more. Expiration date: 3 months from date of issue
- Preferred Shoppers earn 2% cash back on all purchases at any branch

Third Rulesheet: Apply Cash Back

- A Preferred Shopper account tracks the accumulated cash back, so that the customer can apply it to reduce any visit's total amount. The cashier will ask a Preferred Shopper if they want to apply a cash back balance to their current purchase.
- Once a Preferred Shopper chooses to apply their cash back balance, the cumulative cash back total maintained by the system is reset to zero, and the accumulation of cash back begins again at the customer's next purchase.

4

Modeling and testing the 'checks' Rulesheet

This chapter shows how to model and test rules in the first Rulesheet.

For details, see the following topics:

Preparing to model the first Rulesheet

Before we build or model anything based on the vocabulary, we need to think about how to approach this part of the problem.

Perspective of the rules

Once an approach has been chosen, we need to choose the "perspective" in the Vocabulary that best represents the terms required by the rules themselves.

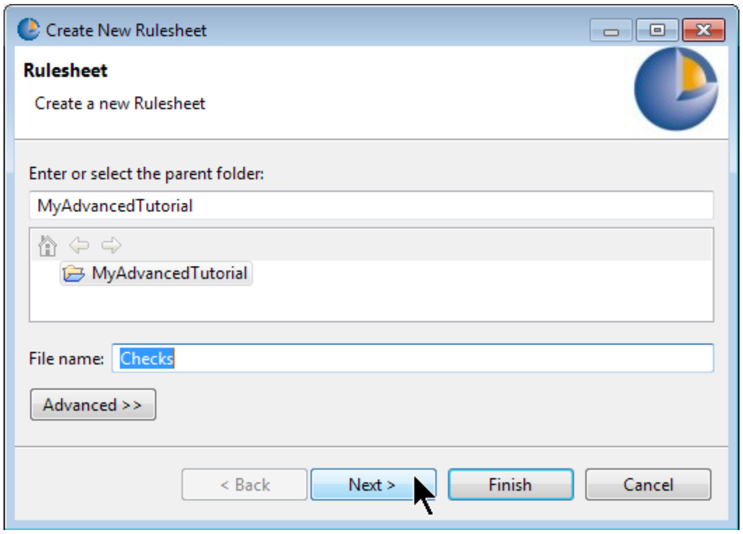
This perspective may change from Rulesheet to Rulesheet.

For this first Rulesheet, beginning with **Customer** as the "root" entity and working with the associated **shoppingCart** and its **items** makes sense because it is a Customer's transaction that is processed by the checkout process step. The contents of the transaction are the **shoppingCart** and its *associated* **items**.

Creating the first Rulesheet and its scope

To create the first Rulesheet and define its scope:

1. Select the menu command **New > Rulesheet**.
2. Choose the parent folder `MyAdvancedTutorial`, name the file `checks`, click **Next**, as shown:



Create New Rulesheet

Rulesheet

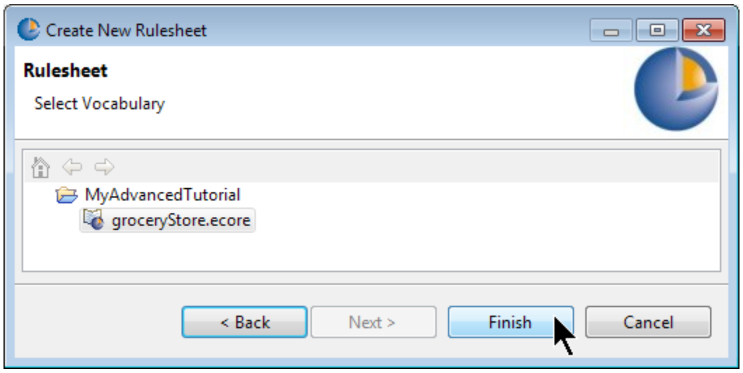
Create a new Rulesheet

Enter or select the parent folder: MyAdvancedTutorial

MyAdvancedTutorial

File name: Checks

3. Click on your vocabulary file name, `groceryStore.ecore`, and then click **Finish**, as shown:



Create New Rulesheet

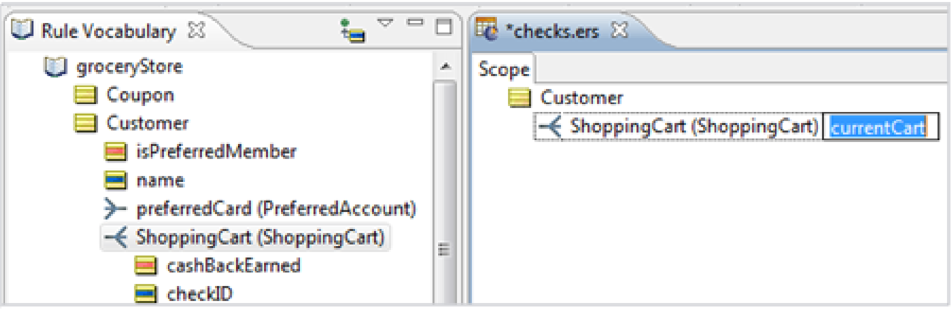
Rulesheet

Select Vocabulary

MyAdvancedTutorial

groceryStore.ecore

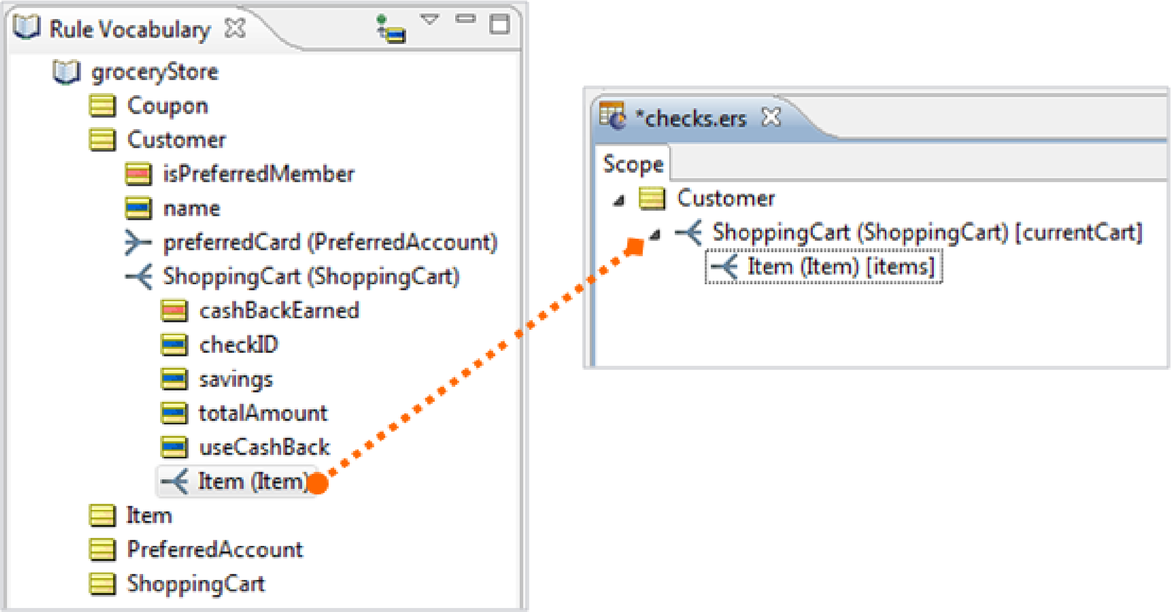
4. Display the **Scope** section of the Rulesheet by either clicking ▦ in the toolbar, or selecting **Rulesheet** > **Advanced View** from Studio's menubar.
5. In the **Rule Vocabulary**, click and drag the `Customer` entity to the **Scope** area of the rulesheet.
6. Click and drag the `ShoppingCart` *association* within the customer, and drop it on the `Customer` in the rulesheet, as shown:



7. Double-click the ShoppingCart in the rulesheet to open its alias entry box. Enter **currentCart**. When we model rules involving a customer's shopping cart, we can refer to it by this alias.

Note: **Scope** is a powerful and important concept. It helps us tell the Corticon rule engine which data to use when evaluating and executing rules In our example, we want the cash register system to process *not just any* shopping cart, but *customers*' shopping carts. This ownership role between a customer and his shopping cart is what the association means. We'll incorporate this association in the rules we build by using the alias that represents it. Scope is such an important concept that an entire chapter is devoted to it in the *Rule Modeling Guide*.

8. Click and drag the `Items` *association* within the customer's shopping cart in the Vocabulary, and drop it on the `ShoppingCart` in the rulesheet, as shown:



9. Double-click the Items in the rulesheet to open its alias entry box. Enter **items**.

Note: Aliases will always insert themselves automatically when terms are dragged and dropped from the Scope section or Vocabulary window to the Rulesheet. Using aliases is generally optional – they simplify and provide a better flow of meaning in rule expressions. However, there are certain modeling cases and scenarios that **require** the use of an alias, such as applying collection operators to sets or collections of data in rules. As we will be working with the collection of items in a customer's shopping cart, it is good to have the **items** alias defined and ready. Since all Studio expressions are case-sensitive, it's better to drag and drop terms instead of typing them manually, as it minimizes the chance of errors.

Modeling and testing the first business rule

This section shows how to model and test the first business rule.

Deconstructing the scenario's barcodes

The first business rule requires the system to examine all items in a customer's shopping cart and determine which items (if any) come from the Liquor department.

The following tables show the **Barcode** key and the department codes for items in these grocery stores.

Grocery Store Barcode Key	
sample barcode: xx–yyy–zzzzz	
xx	store code
yyy	department code
zzzzz	item number

Grocery Store Department Codes	
Department	Code
Produce	260
Canned Goods	265
Meat	270
Deli	275
Frozen Foods	280
Soda/Juice	285
Floral	290
Bakery	295
Housewares	300
Detergent & Cleaning Supplies	305
School Supplies	310
Liquor	291

Modeling the first business rule

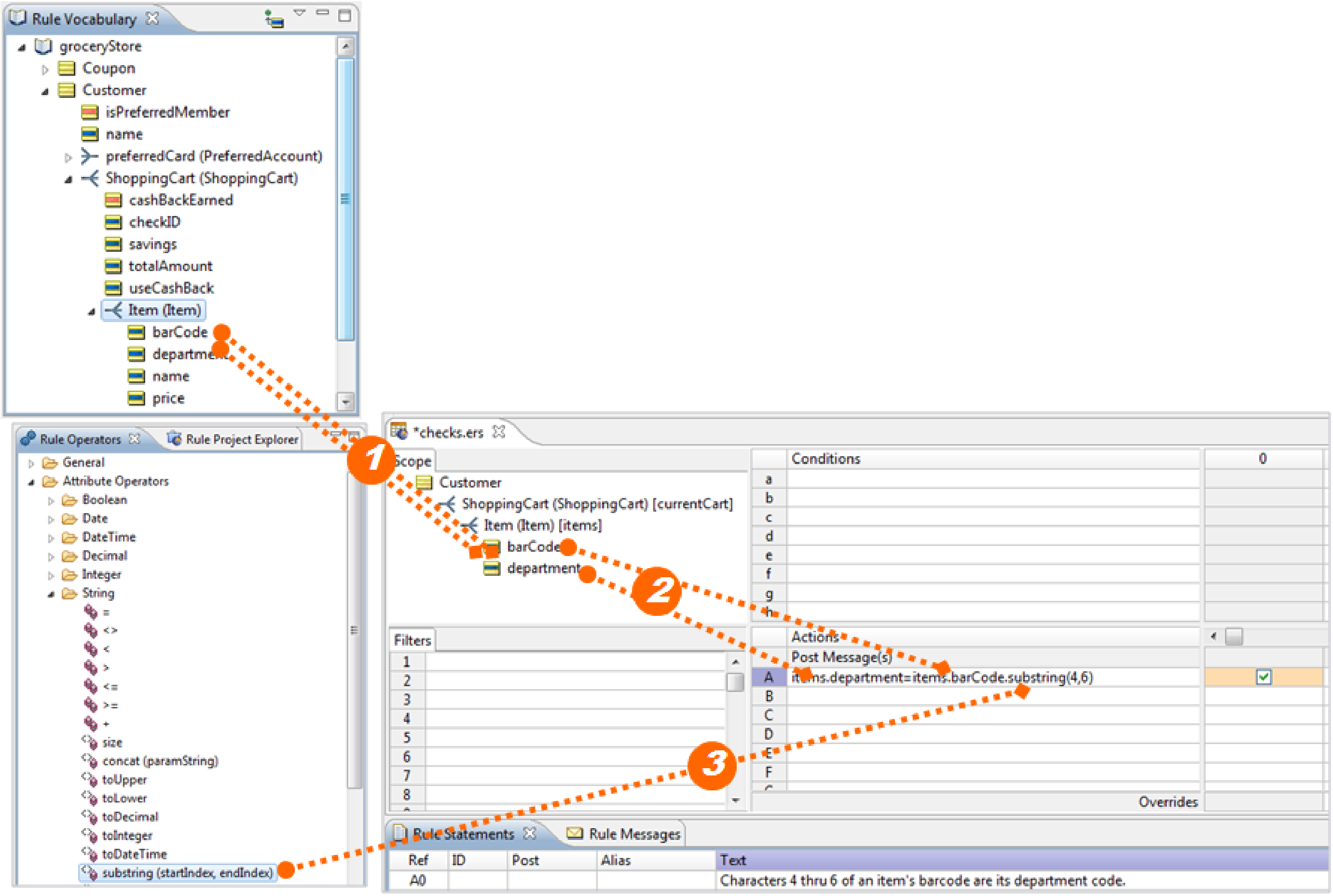
To locate items in a customer's shopping cart that come from the Liquor department:

1. In the **Vocabulary**, drag the Customer's ShoppingCart's Item attributes `barCode` and `department` to the rulesheet's **Scope** view, dropping them on the corresponding `Item` line in the **Scope** view.

2. In the rulesheet's **Scope** view, drag `department` to line 1 in the **Actions** panel. The entry displays as `items.department`.

3. Click at the end of the Action definition to extend the definition with

   `= items.barCode.substring(4,6)`, as shown (names are case-sensitive!):



Actions, A: items.department=items.barCode.substring(4,6) — column 0: ☑

Rule Statements:

Ref	ID	Post	Alias	Text
A0				Characters 4 thru 6 of an item's barcode are its department code.

4. In column **0**, select the checkbox as this is a non-conditional rule. The rule performs its calculation or action in all cases.

5. On the **Rule Statements** tab, in the first line's **Ref** area enter `A0`, and then, in its **Text** area, enter the descriptive text `Characters 4 through 6 of an item's bar code are its department code.`

Note: Because the alias **items** represents the *collection* of items in a customer's shopping cart, this rule will evaluate and process *each* item in a customer's shopping cart, extract the department code for each, and then assign that code to the item's **department** attribute. For *all* items in a given customer's shopping cart, this rule will execute *once per item*. This iteration is a natural behavior of the rules engine: *it will automatically process all data that matches the rule's scope*.

Terms can dragged from the Vocabulary, and automatically added to the Scope window. Over time, the Scope window becomes a reduced version of the project's Vocabulary, containing only the terms used by rules in that Rulesheet.

Logical Analysis

Typically, you would check for **Conflicts** and **Completeness** before testing with data. But these are meaningful only for columns containing Conditions. Since Column 0 has no Conditions, it is not necessary to perform these checks now. The steps for performing these checks, and taking any necessary corrective actions, are detailed in the *Basic Rule Modeling Tutorial* and will not be restated in this guide.

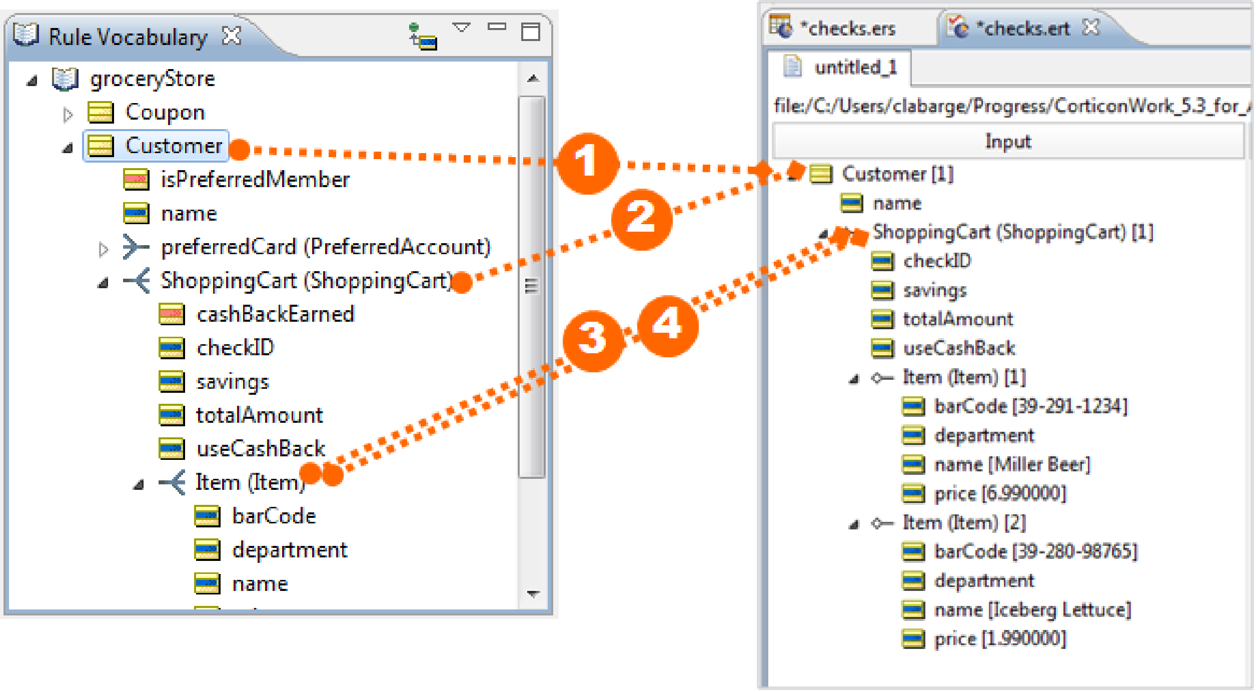
Testing the first rule

It is a good practice to test rules as you develop them.

Note: It is critical to drop the items from the Vocabulary into the Input panel of the Ruletest in the order indicated so that we duplicate the Scope of the rule which will be processing this data.

To test the Rulesheet:

1. Select the menu command **New > Ruletest**. Choose the parent folder `MyAdvancedTutorial` and name the file `checks`.
2. In the **Rule Vocabulary**, drag the `Customer` to the Ruletest's **Input** column.
3. In the **Rule Vocabulary**, drag the Customer's `ShoppingCart`, and then drop it on the `Customer` line in the Ruletest.
4. In the **Rule Vocabulary**, drag the Customer's ShoppingCart's `Item` to the Ruletest's **Input** column, and then drop it on the `ShoppingCart`.
5. As we will test two items, drag the Customer's ShoppingCart's `Item` again to the Ruletest's **Input** column, and then drop on the `ShoppingCart`. A second item is created for the test.
6. Double-click each attribute of the items, and then use the entry box to enter the test data as shown in the following illustration.



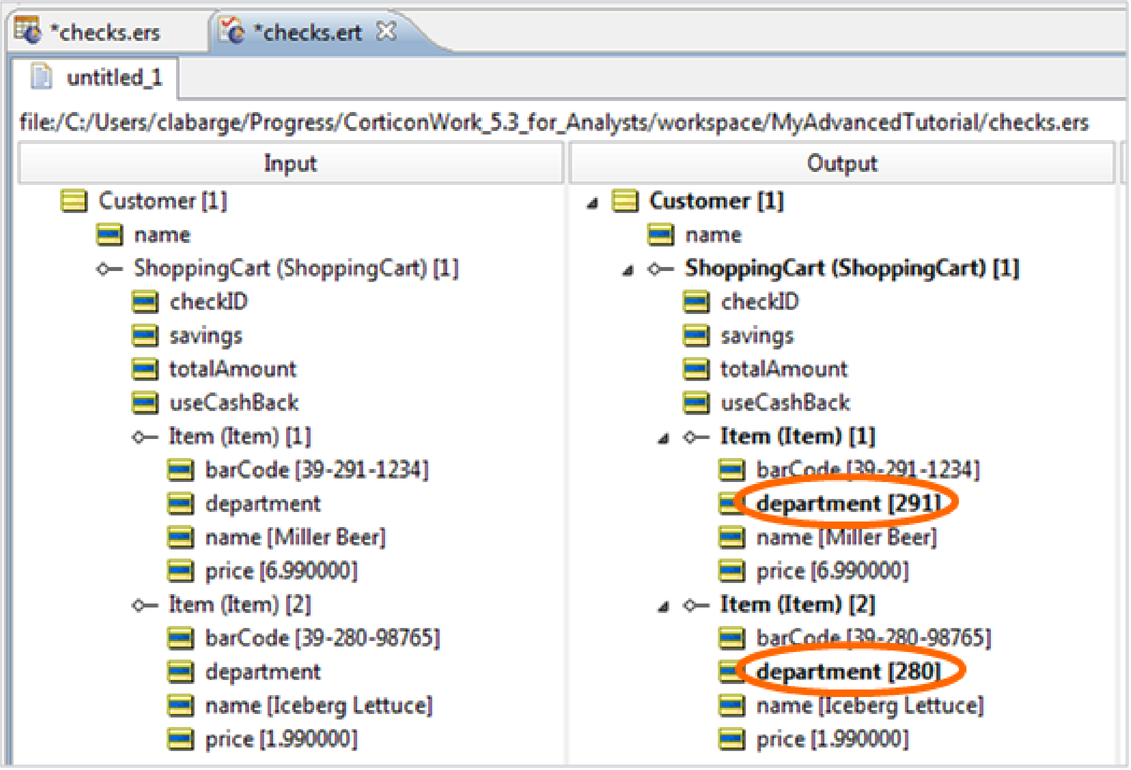
Note that you do not enter the department as our rule will extract it from the barcode.

7. Choose the menu command **Ruletest > Run All Tests**.

Results of the rule test

Our first rule has worked as expected.

Characters 4-6 have been successfully parsed in each item's **barCode** and assigned to its **department** attribute.



*checks.ers | *checks.ert

untitled_1

file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/checks.ers

```
Input                                     Output
Customer [1]                              Customer [1]
    name                                      name
    ShoppingCart (ShoppingCart) [1]           ShoppingCart (ShoppingCart) [1]
        checkID                                   checkID
        savings                                   savings
        totalAmount                               totalAmount
        useCashBack                               useCashBack
        Item (Item) [1]                           Item (Item) [1]
            barCode [39-291-1234]                     barCode [39-291-1234]
            department                                department [291]
            name [Miller Beer]                        name [Miller Beer]
            price [6.990000]                          price [6.990000]
        Item (Item) [2]                           Item (Item) [2]
            barCode [39-280-98765]                    barCode [39-280-98765]
            department                                department [280]
            name [Iceberg Lettuce]                    name [Iceberg Lettuce]
            price [1.990000]                          price [1.990000]
```

Using the collection operator

This section shows how to extend the first business rule to use the collection operator.

Extending the first business rule

Now that department codes are readily available for every item in a customer's shopping cart, we need to determine if any came from the Liquor department.

This type of question requires us to "examine" the collection of **items** and see if there exists at least one item with **department** = **291**. Since we only need one "**check ID**" alert per checkout transaction, a collection operator is appropriate.

A collection operator will evaluate *once per collection* and not once per item as the first part of this rulesheet did.

Making use of the **items** alias, we've added a **Condition** that determines if any Liquor items exist in the customer's shopping cart. An **Action** assigns a value of **true** to the shopping cart's **checkID** attribute if any are found. We're assuming that the **checkID** term will act as the alerting mechanism to signal the cashier that an ID check is required during this checkout transaction.

To create the collection test:

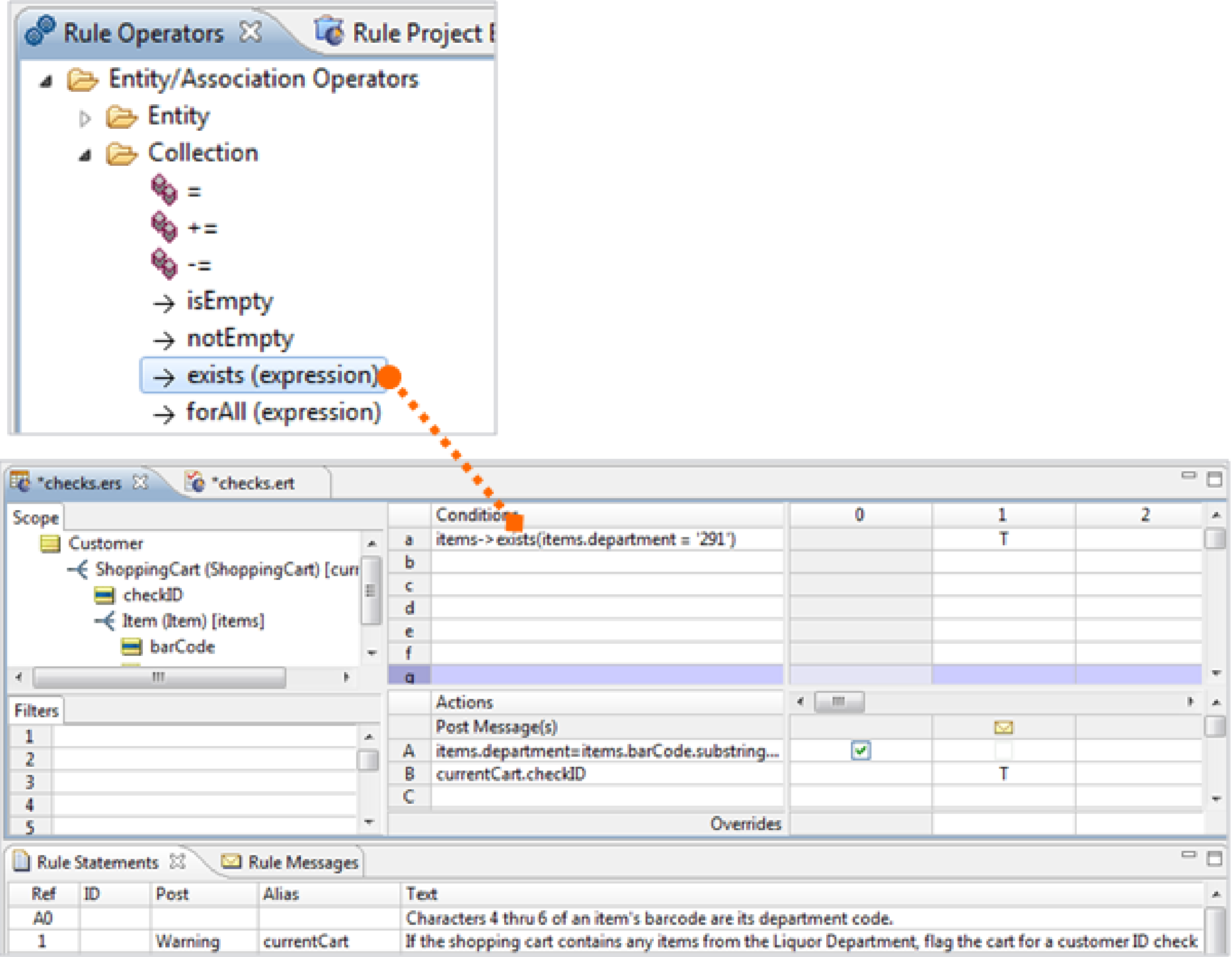
1. On the rulesheet, enter on line **a** of the **Conditions** section:,

   ```
   items -> exists(items.department='291')
   ```

   and then enter (or pulldown) the value `T` in column **1**

   Note: The **items** alias was mandatory here as it refers to the collection, and the `-> exists` operator means that the result will evaluate as either `true` or `false`. (For more information on collections and collection operators, see the *Rule Modeling Guide* and the *Rule Language Guide*.

2. On line `B` of the **Actions - Post Messages** section, enter `currentCart.checkID`, and in column **1** of that line enter `T`.
3. In the **Rule Statements** section, enter a **Ref** to column `1`, select that it will **Post** as a `Warning` against the **Alias** `currentCart`, and display the **Text** shown to the cashier that an ID check is required during this checkout transaction.



	Conditions	0	1	2
a	items->exists(items.department = '291')		T	
b				
c				
d				
e				
f				
g				

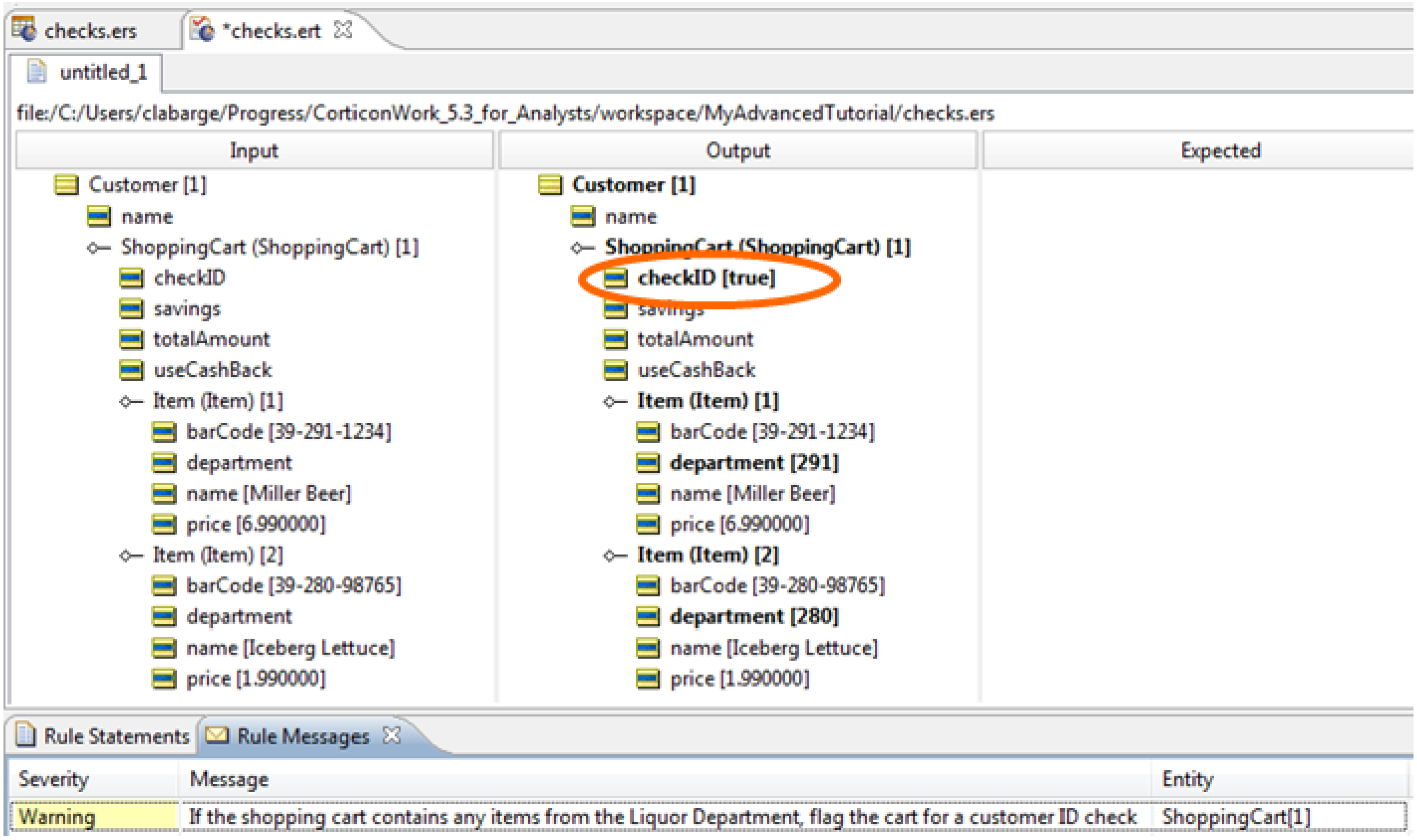
	Actions			
	Post Message(s)		✉	
A	items.department=items.barCode.substring...	☑	□	
B	currentCart.checkID		T	
C				
	Overrides			

Ref	ID	Post	Alias	Text
A0				Characters 4 thru 6 of an item's barcode are its department code.
1		Warning	currentCart	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check

Results of the complete business rule test

On the Ruletest page, run the test. The rule performs as intended, displaying the correct warning. Our Condition/Action rule works!.

A customer's shopping cart containing an item from the Liquor department has been identified, and the **checkID** attribute is set to **true** to alert the cashier to check the customer's ID. The business rule statement has also been posted in the **Message Box**.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/checks.ers

Input	Output	Expected
Customer [1]	Customer [1]	
name	name	
ShoppingCart (ShoppingCart) [1]	ShoppingCart (ShoppingCart) [1]	
checkID	checkID [true]	
savings	savings	
totalAmount	totalAmount	
useCashBack	useCashBack	
Item (Item) [1]	Item (Item) [1]	
barCode [39-291-1234]	barCode [39-291-1234]	
department	department [291]	
name [Miller Beer]	name [Miller Beer]	
price [6.990000]	price [6.990000]	
Item (Item) [2]	Item (Item) [2]	
barCode [39-280-98765]	barCode [39-280-98765]	
department	department [280]	
name [Iceberg Lettuce]	name [Iceberg Lettuce]	
price [1.990000]	price [1.990000]	

Severity	Message	Entity
Warning	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check	ShoppingCart[1]

You can verify the rule by changing the department code of the first item to, say, `280`, and running it (no warning), and then changing it back to `291` to run it again (warning issued).

Rule Models vs. Business Rules

There isn’t always a one-to-one correlation between the Business Rules defined in a business scenario and the corresponding rules modeled in Studio. Often, as we see in this example, one Business Rule requires more than one rule in Studio. This is normal. A good guideline is to keep your individual rule models relatively simple and let them work together to perform more complex logic defined by the Business Rules. In this first Rulesheet, two rule columns work together to accomplish the goal of the Scenario’s “check ID” Business Rule.

Adding to scope

As we are going to flow Ruelesheets together, we can perform functions on one Rulesheet that will be used on a later Rulesheet. In this case, the Rulesheet's purpose is to raise alerts, so determining whether the customer is a preferred card holder seems appropriate at this point.

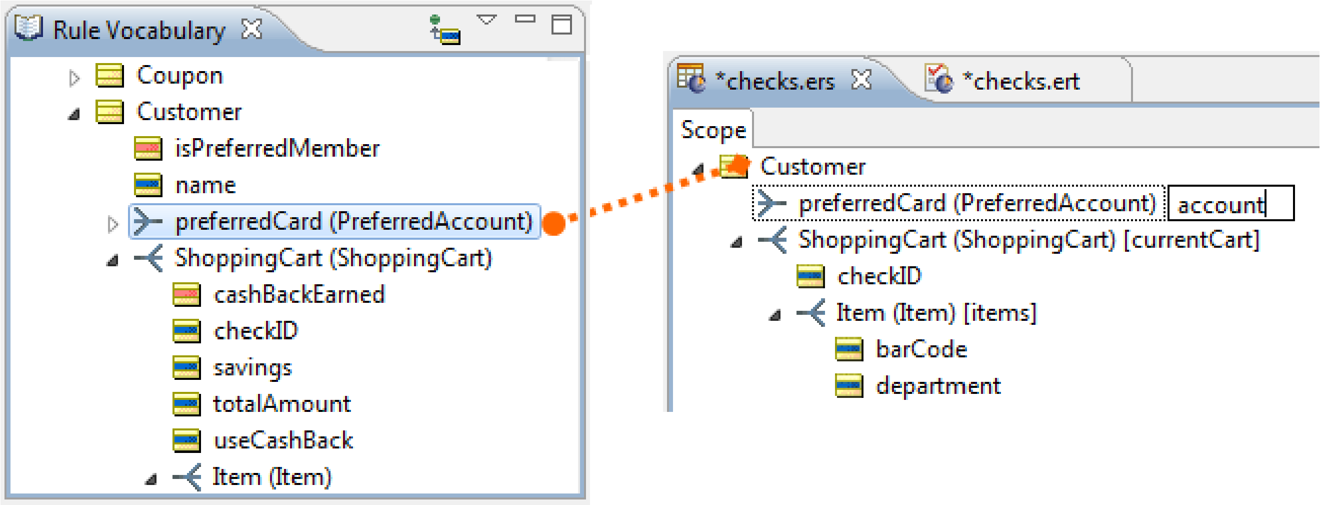
You might feel that this rule belongs in the next rulesheet where we define rules for promotions and coupons. But consider that the first Rulesheet could be under the control of the administrative team, while the next Rulesheet 'belongs' to marketing, and the third Rulesheet is defined (and audited) by the accounting group.

To add the preferred customer test to the Rulesheet:

1. In the **Vocabulary**, drag the Customer's `PreferredCard` association to the Rulesheet's **Scope** view, and then drop it on the Customer in the Rulesheet.

Note: Not all customers have **Preferred Accounts**, but those who do will have an associated **preferredCard**. Customers holding a **preferredCard** are eligible for various promotions, such as coupons for discounts on gasoline purchases. The **account** alias defined here prepares us to examine the collection it represents in the next rule.

2. Double-click `preferredCard` in the Rulesheet's **Scope** panel, and then enter `account` as the alias.



Note:

That alias represents a "potential collection", that is, a customer will have a **Preferred Card** *only* if they have a **Preferred Account**. And the "many-to-one" cardinality of the association means a customer will have *at most one* account. Other customers (as with a family) can share the same **Preferred Account**.

For Customers who do not have **Preferred Accounts**, the alias **account** represents an *empty collection* (the collection contains no elements).

Modeling and testing the second business rule

This section shows how to model and test the second business rule.

Modeling another condition/action rule

The **->notEmpty** collection operator checks a collection for the existence of *at least one* element in the set. Because **->notEmpty** is "acting on" a collection, the **account** alias must be used with it.

If the condition is true, we know the customer has an account. We've added an action that assigns the **isPreferredMember** attribute (from the **Customer** entity) the value of **true** and posts an informational message.

Now, whenever we need to know if a customer is a preferred customer, we simply refer to the value of their **isPreferredMember** attribute. This method of "flagging" an entity with a boolean attribute is convenient when modeling larger Ruleflows. The value of the flag, like all attributes, will carry over to other rules on this and other Rulesheets in the same Ruleflow.

To add a rule that checks if the customer has a Preferred Card account:

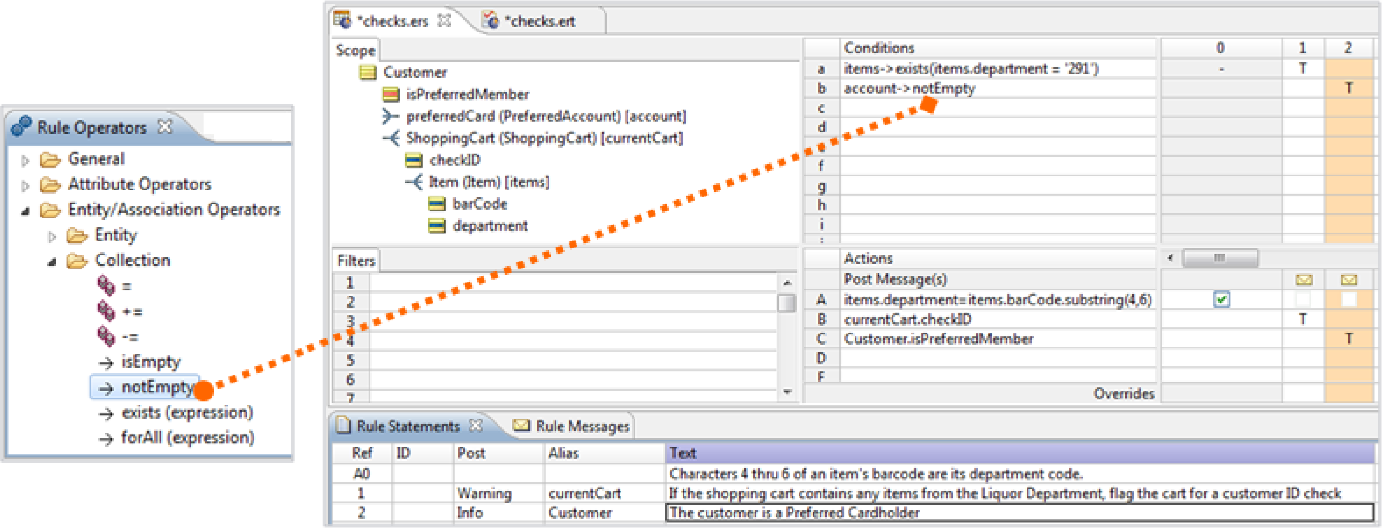
1. On the rulesheet, enter on line **b** of the **Conditions** section:,

   ```
   account -> notEmpty
   ```

   and then enter (or pulldown) the value `T` in column **2**

2. On line `C` of the **Actions - Post Messages** section, enter `Customer.isPreferredMember`, and in column **2** of that line enter `T`.

3. In the **Rule Statements** section, enter a **Ref** to column `2`, select that it will **Post** as `Info` against the **Alias** `Customer`, and display the **Text** `The customer is a Preferred Cardholder.`



	Conditions	0	1	2
a	items->exists(items.department = '291')	-	T	
b	account->notEmpty			T
c				
d				
f				
g				
h				
i				

	Actions			
	Post Message(s)		✉	✉
A	items.department=items.barCode.substring(4,6)	☑	□	□
B	currentCart.checkID		T	
C	Customer.isPreferredMember			T
D				
F				
	Overrides			

Ref	ID	Post	Alias	Text
A0				Characters 4 thru 6 of an item's barcode are its department code.
1		Warning	currentCart	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check
2		Info	Customer	The customer is a Preferred Cardholder

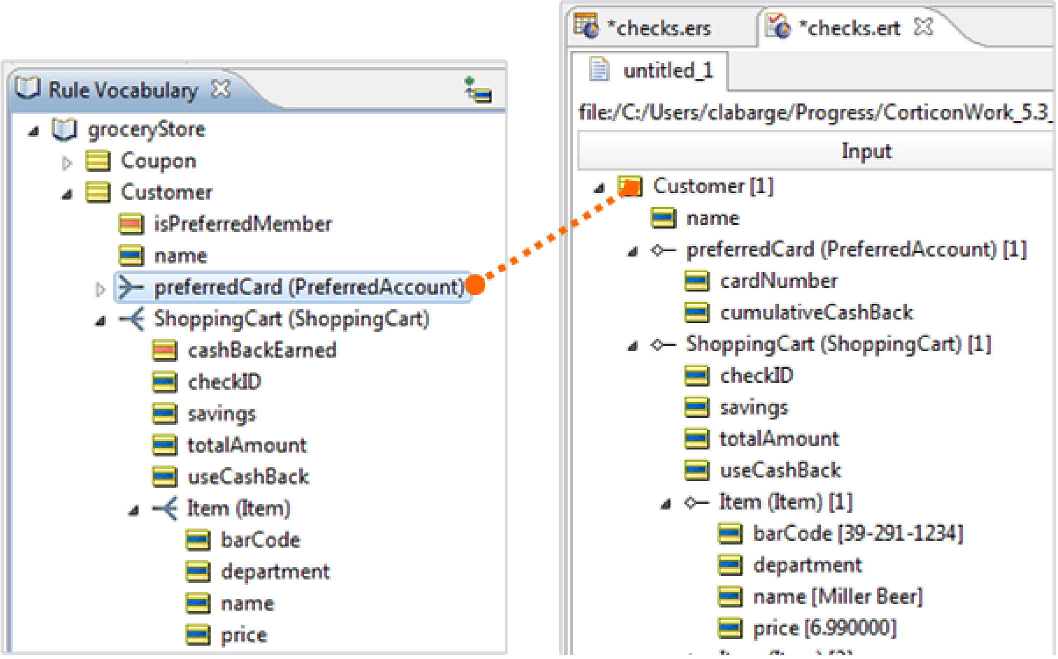
The boolean condition in row b that does this using the **notEmpty** collection operator. If the **account** alias is not empty, we know the customer has such an account.

Testing the second condition/action rule

To test the rule, it needs to detect the presence of a **Preferred Card** account associated with this customer.

To test the second Condition/Action rule:

1. Drag and drop the **preferredCard** association onto the **Customer** entity in the **Ruletest Input**, as shown.



Rule Vocabulary

- groceryStore
  - Coupon
  - Customer
    - isPreferredMember
    - name
    - preferredCard (PreferredAccount)
    - ShoppingCart (ShoppingCart)
      - cashBackEarned
      - checkID
      - savings
      - totalAmount
      - useCashBack
      - Item (Item)
        - barCode
        - department
        - name
        - price

*checks.ers | *checks.ert

untitled_1

file:/C:/Users/clabarge/Progress/CorticonWork_5.3_

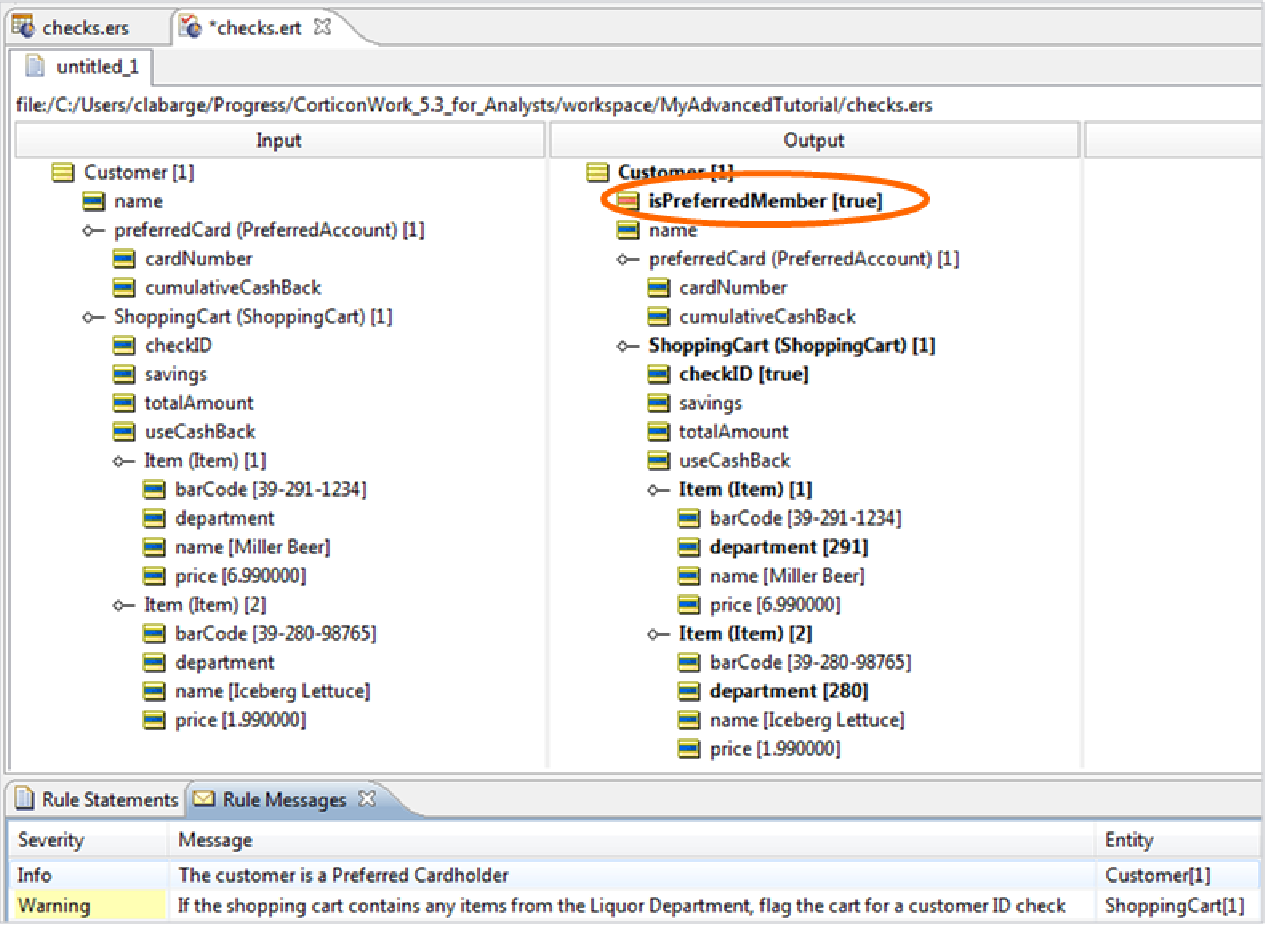
Input

- Customer [1]
  - name
  - preferredCard (PreferredAccount) [1]
    - cardNumber
    - cumulativeCashBack
  - ShoppingCart (ShoppingCart) [1]
    - checkID
    - savings
    - totalAmount
    - useCashBack
    - Item (Item) [1]
      - barCode [39-291-1234]
      - department
      - name [Miller Beer]
      - price [6.990000]

2. Execute the Ruletest.

Results of testing the second condition/action rule

Choose **Run** on the Ruletest page.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/checks.ers

```
Input                                    Output
Customer [1]                             Customer [1]
  name                                     isPreferredMember [true]
  preferredCard (PreferredAccount) [1]     name
    cardNumber                             preferredCard (PreferredAccount) [1]
    cumulativeCashBack                       cardNumber
  ShoppingCart (ShoppingCart) [1]            cumulativeCashBack
    checkID                                ShoppingCart (ShoppingCart) [1]
    savings                                  checkID [true]
    totalAmount                              savings
    useCashBack                              totalAmount
    Item (Item) [1]                          useCashBack
      barCode [39-291-1234]                  Item (Item) [1]
      department                               barCode [39-291-1234]
      name [Miller Beer]                       department [291]
      price [6.990000]                         name [Miller Beer]
    Item (Item) [2]                            price [6.990000]
      barCode [39-280-98765]                 Item (Item) [2]
      department                               barCode [39-280-98765]
      name [Iceberg Lettuce]                   department [280]
      price [1.990000]                         name [Iceberg Lettuce]
                                               price [1.990000]
```

Severity	Message	Entity
Info	The customer is a Preferred Cardholder	Customer[1]
Warning	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check	ShoppingCart[1]

Notice that the **isPreferredMember extended transient** attribute has been inserted and assigned a value of **true**, and that an informational message has been posted.

Our second Condition/Action rule has worked as we expected.

Modeling and testing the price summation rule

This section shows how to model and test the price summation rule.

Modeling the price summation rule

The first rulesheet also will include one more non-conditional **Action**, one that will calculate the **totalAmount** of all items in a customer's shopping cart.

Note: This is accomplished by using the `->sum` operator to add up the `price` attributes of all elements in the `items` alias, then assigning that value to the `totalAmount` attribute.

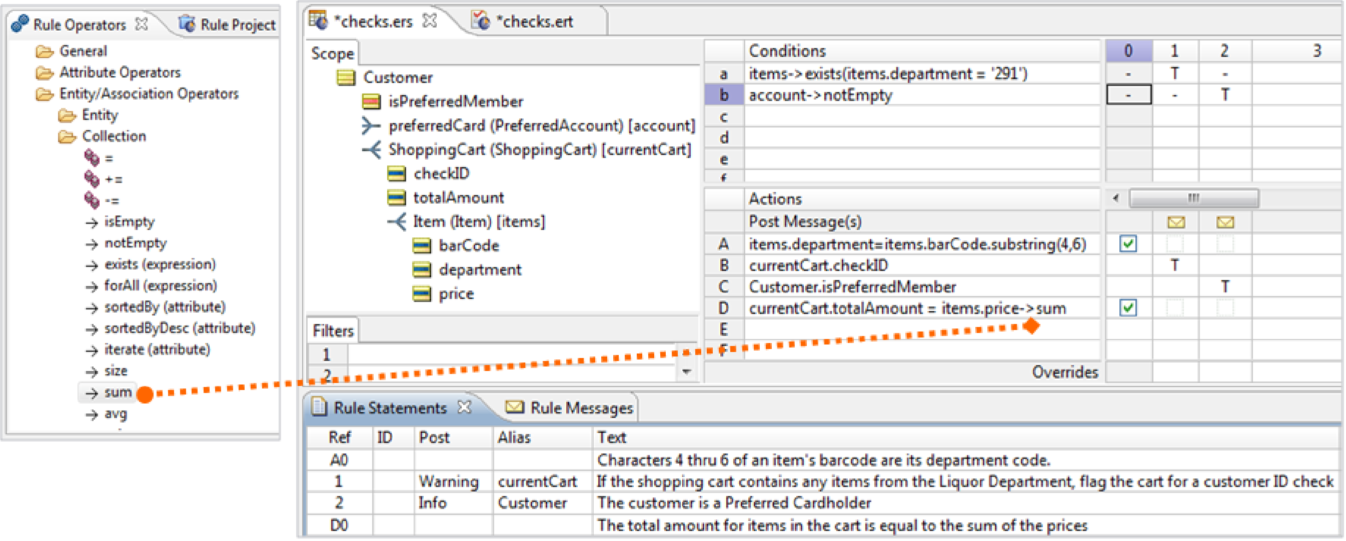
To calculate the total price of the items in the current shopping cart:

1. On line `D` of the **Actions** section, enter

```
currentCart.totalAmount=items.price ->sum
```

2. In column 0, select the checkbox as this is a non-conditional rule. The rule performs its calculation in all cases.
3. In the **Rule Statements** section, enter a **Ref** to column `D0`, and display the **Text** as shown.

Note: You could drag the operator `sum` to the line instead of typing it, as shown:



	Conditions	0	1	2	3
a	items->exists(items.department = '291')	-	T	-	
b	account->notEmpty	-	-	T	
c					
d					
e					
f					

	Actions				
	Post Message(s)		✉	✉	
A	items.department=items.barCode.substring(4,6)	☑			
B	currentCart.checkID		T		
C	Customer.isPreferredMember			T	
D	currentCart.totalAmount = items.price->sum	☑			
E					
F					
	Overrides				

Ref	ID	Post	Alias	Text
A0				Characters 4 thru 6 of an item's barcode are its department code.
1		Warning	currentCart	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check
2		Info	Customer	The customer is a Preferred Cardholder
D0				The total amount for items in the cart is equal to the sum of the prices

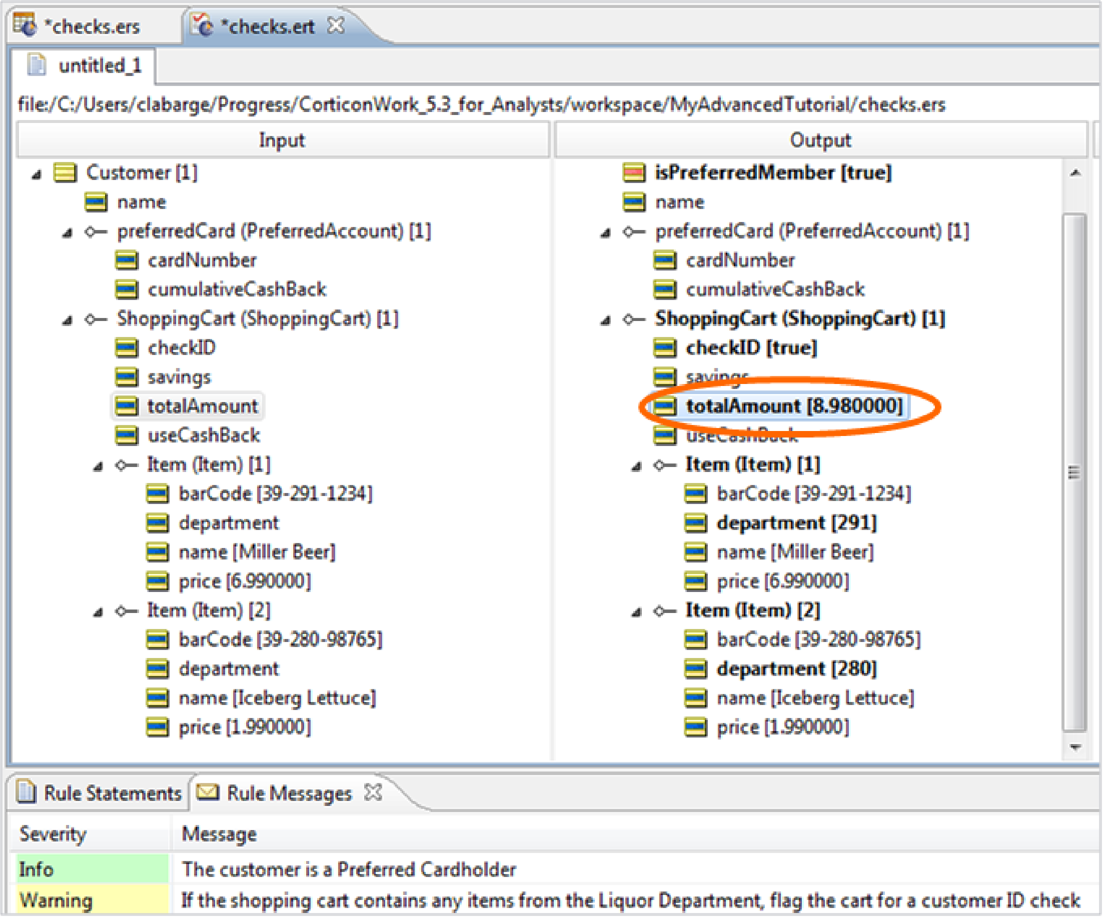
Note: Adding **Rule Statements** is good practice, even when you do not post them.

Testing the price summation rule

To test the third rule on this Rulesheet, simply go to the ruletest and run it.

Price summation test results

The `totalAmount` is the sum of the `price` of the two items.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/checks.ers

Input

```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack
    ShoppingCart (ShoppingCart) [1]
        checkID
        savings
        totalAmount
        useCashBack
        Item (Item) [1]
            barCode [39-291-1234]
            department
            name [Miller Beer]
            price [6.990000]
        Item (Item) [2]
            barCode [39-280-98765]
            department
            name [Iceberg Lettuce]
            price [1.990000]
```

Output

```
isPreferredMember [true]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack
    ShoppingCart (ShoppingCart) [1]
        checkID [true]
        savings
        totalAmount [8.980000]
        useCashBack
        Item (Item) [1]
            barCode [39-291-1234]
            department [291]
            name [Miller Beer]
            price [6.990000]
        Item (Item) [2]
            barCode [39-280-98765]
            department [280]
            name [Iceberg Lettuce]
            price [1.990000]
```

Severity	Message
Info	The customer is a Preferred Cardholder
Warning	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check

Summary – the test results

The Ruletest for Rulesheet one shows that:

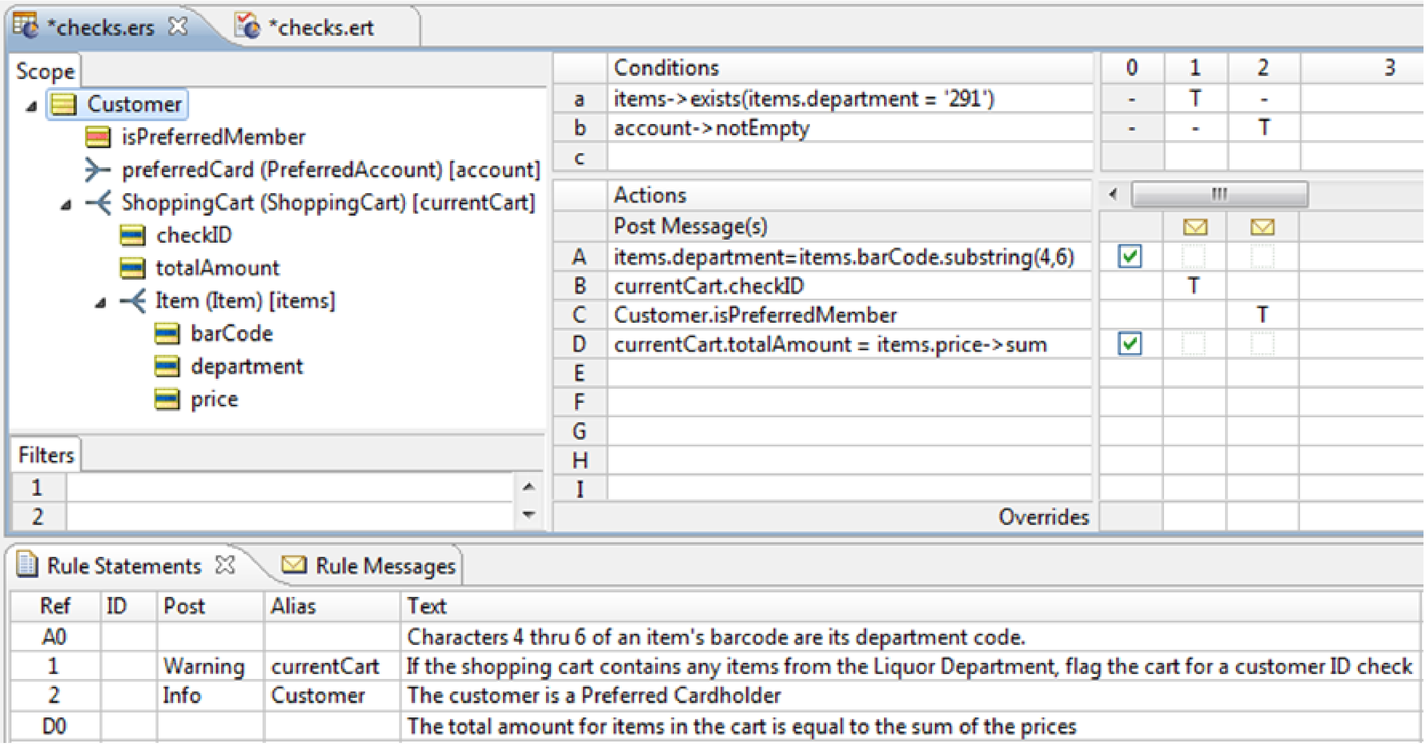
- An item in the cart was from department **291**, the Liquor department (identified by the **barCode**). A **checkID** alert was issued and a warning message was posted.
- The **isPreferredMember** attribute has a value of **true** because a **preferredCard** entity is associated with the customer; the appropriate informational message has been posted.
- The **totalAmount** shows that the two item prices have been added together correctly.

The rules on this Rulesheet have all functioned as designed.

Rulesheet one is complete and tested

Rulesheet one did the administrative essentials. It made sure that laws were supported, that customers were recognized, and then added up the groceries.

The completed Rulesheet looks like this:



- Customer
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
  - ShoppingCart (ShoppingCart) [currentCart]
    - checkID
    - totalAmount
    - Item (Item) [items]
      - barCode
      - department
      - price

1	
2	

	Conditions	0	1	2	3
a	items->exists(items.department = '291')	-	T	-	
b	account->notEmpty	-	-	T	
c					

	Actions			
	Post Message(s)		✉	✉
A	items.department=items.barCode.substring(4,6)	☑	☐	☐
B	currentCart.checkID		T	
C	Customer.isPreferredMember			T
D	currentCart.totalAmount = items.price->sum	☑	☐	☐
E				
F				
G				
H				
I				
	Overrides			

Ref	ID	Post	Alias	Text
A0				Characters 4 thru 6 of an item's barcode are its department code.
1		Warning	currentCart	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check
2		Info	Customer	The customer is a Preferred Cardholder
D0				The total amount for items in the cart is equal to the sum of the prices

Subsequent Rulesheets isolate rules that might be managed and tested by different departments. Marketing might be handling the promotions and coupons, while accounting might be controlling the cashback programs.

5

Modeling and testing the 'coupons' Rulesheet

This chapter shows how to model and test rules for the second Rulesheet.

For details, see the following topics:

Model the second Rulesheet

This Rulesheet provides promotions to **Preferred Account** holders when they spend a minimum amount or buy items from specific departments in the store. According to the scenario's Business Rules, these promotions vary to include discounts, rebates, or even free gifts when items are purchased in specific amounts or from specific departments. The promotions will change frequently so modeling them in Corticon will make future changes much easier.

Calculations, Promotions, and Coupons

For this Rulesheet, we will build and test rules so that:

- Preferred Shoppers earn 2% cash back on all purchases at any branch
- Preferred Shoppers receive a coupon for one free balloon for every item purchased from the Floral department. Expiration date: none
- Preferred Shoppers receive a coupon for $2 off their next purchase when 3 or more Soda/Juice items are purchased in a single visit. Expiration date: one year from date of issue
- Preferred Shoppers receive a coupon for 10% off their next gasoline purchase at any chain-owned convenience store with any purchase of $75 or more. Expiration date: 3 months from date of issue

Note: When multiple Rulesheets are included in a Ruleflow (a single `.erf` file), the Rulesheets will execute in a sequence determined by their Rulesheet order in the Ruleflow Editor. For more information about sequencing Rulesheets as well as additional details on the Ruleflow Editor, see the *Corticon Studio Quick Reference Guide*.

Scope revisited

Let's create the second Rulesheet. The **Scope** for the `coupons` Rulesheet is similar to the first Rulesheet.

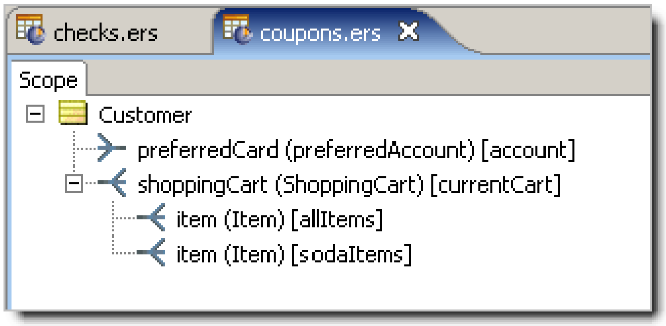
A customer's shopping cart is assigned the alias **currentCart**, just like on the `checks` Rulesheet. On the `coupons` Rulesheet, there are two new aliases to define the **currentCart.item** perspective of our data. For now, we will simply define the two aliases **allItems** and **sodaItems** to represent the same perspective, but we will differentiate between them shortly.

To create the new Rulesheet's scope:

1. Select the menu command **New > Rulesheet**. Choose the parent folder `MyAdvancedTutorial` and name the file `coupons`.
2. Display the **Scope** section of the Rulesheet. Either click ▦ in the toolbar, or select **Rulesheet** > **Advanced View** from Studio's menubar.
3. In the **Rule Vocabulary**, click and drag the **Customer** entity to the **Scope** area of the Rulesheet.
4. In the scope section, click and drag the `ShoppingCart` *association* within the customer, and drop it on the `Customer` in the Rulesheet.
5. Double-click `ShoppingCart` in the Rulesheet to open its alias entry box. Enter **currentCart**.
6. In the scope section, click and drag the `preferredCard` *association* within the customer, and drop it on the `Customer` in the Rulesheet.
7. Double-click `preferredCard` in the Rulesheet to open its alias entry box. Enter **account**.
8. In the scope section, click and drag the `item` *association* within the customer's shopping cart in the Vocabulary, and drop it on the `shoppingCart` in the Rulesheet

9. Double-click the `item` in the Rulesheet to open its alias entry box. Enter `allItems`.

10. Repeat dragging the `item` *association* to the `ShoppingCart` in the Rulesheet, this time entering the alias `sodaItems`, as shown:



checks.ers | coupons.ers

Scope

```
⊟ Customer
    preferredCard (preferredAccount) [account]
  ⊟ shoppingCart (ShoppingCart) [currentCart]
      item (Item) [allItems]
      item (Item) [sodaItems]
```

Ruleflow

Now that multiple Rulesheets will be in use, we will create a Ruleflow file.

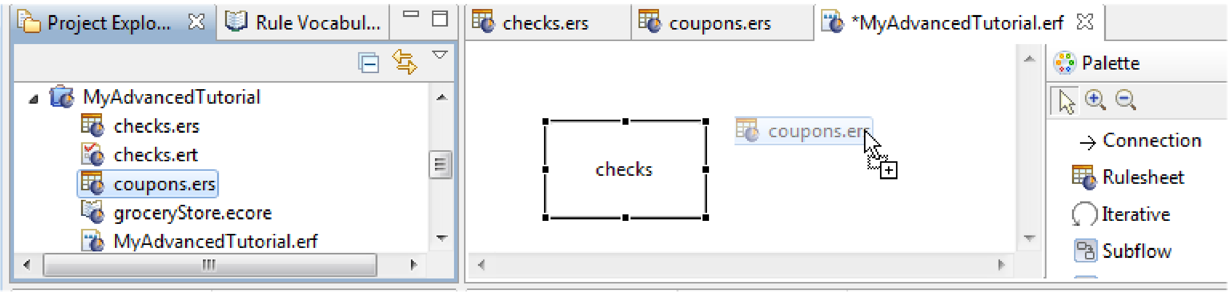
When you prepare to test Rulesheet two, you want the decisions and calculations from Rulesheet one, `checks.ers`, to be fired before Rulesheet two, `coupons.ers`, is evaluated.

The Rulesheet processing sequence is visible here in the **Ruleflow** diagram. For more information on the Ruleflow Editor's purpose and functionality, see the *Quick Reference Guide*.

To create the Ruleflow:

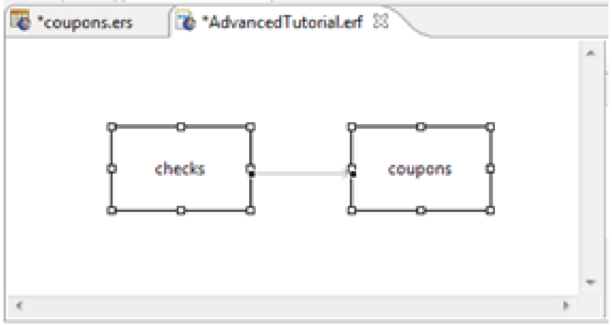
1. Select the menu command **New > Ruleflow**. Choose the parent folder `MyAdvancedTutorial` and name the file `MyAdvancedTutorial` (as it is going to include all the Rulesheets that we create.)

2. Select the **ProjectExplorer**. Click and drag `checks.ers`, and then `coupon.ers` to the Ruleflow canvas, as shown:



3. Click on **Connection** on the **Ruleflow** palette, then click on the `checks` box, and drag and drop on the `coupons` box. The flow is created. When we test Rulesheet two, `coupons.ers`, we'll set the `Test Subject` as the Ruleflow so that the two Rulesheets run in sequence.

The Ruleflow for the first two Rulesheets is:



Filter expressions

Customers who are not **Preferred Card** holders are not eligible for the promotions defined in the original business rules. So we want to filter out non-preferred customers from evaluation by the Rulesheet. A **Filter** expression will limit or reduce the data in working memory to *only that subset* whose members satisfy the expression. A filter *does not* permanently remove or delete any data, it simply *excludes* data from evaluation by the rules in the same Rulesheet.

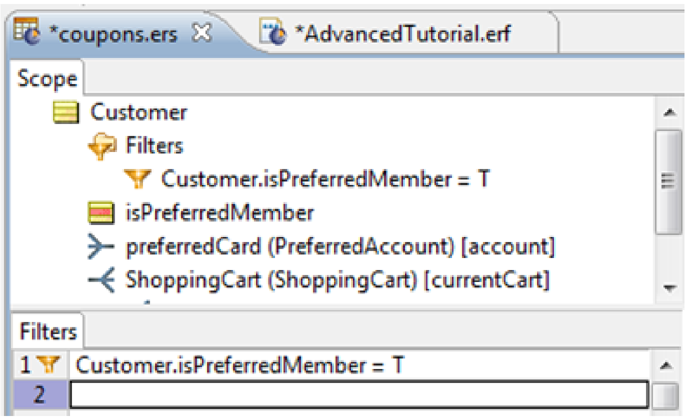
Note: Data filtered out in one Rulesheet is not also filtered out in other Rulesheets *unless* you include the Filter expression in those Rulesheets, too.

The following **Filter** expression "filters out" all non-preferred customers by allowing only those customers with **isPreferredMember** attribute value of **true** to pass (survive).

To create a filter:

In the `coupons` rulesheet Filters section, on line 1, enter:

```
Customer.isPreferredMember = T
```


*coupons.ers
*AdvancedTutorial.erf
Scope
Customer
Filters
Customer.isPreferredMember = T
isPreferredMember
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]
Filters
1
Customer.isPreferredMember = T
2

Note: Filter expressions can behave in ways more complex and powerful than the simple filter shown here. An entire chapter in the *Rule Modeling Guide* is devoted to them.

Calculating and testing cashBackEarned attribute

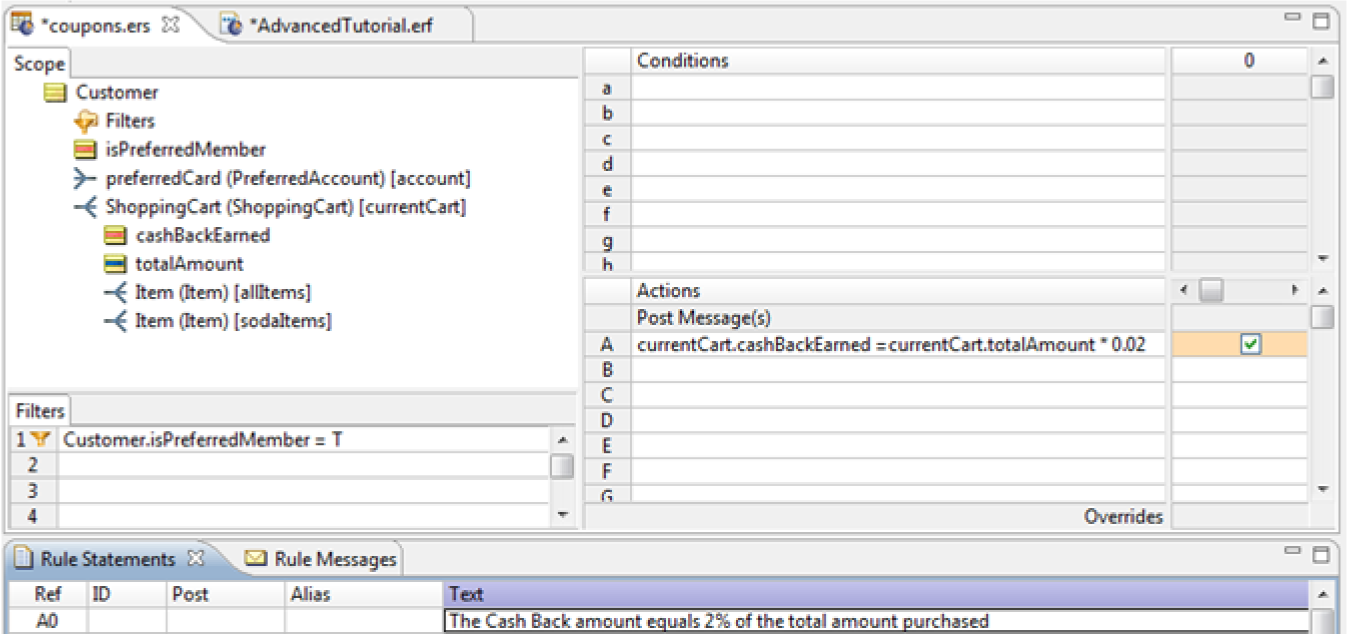
This section shows how to calculate and test the `cashBackEarned` attribute.

Calculating cashBackEarned amount

Preferred Shoppers earn 2% cash back on all purchases. That is expressed in the Rulesheet as follows:

1. In **Action** row **A**, enter `currentCart.cashBackEarned = currentCart.totalAmount*0.02`
2. In column **0** of that row, check that this is a non-conditional action for Preferred Customers.
3. Rule Statement reference **A0** is entered with only the Text entry `The Cash Back amount equals 2% of the total amount purchased.`

The result looks like this:



Filters	
1	Customer.isPreferredMember = T
2	
3	
4	

	Actions	0
	Post Message(s)	
A	currentCart.cashBackEarned =currentCart.totalAmount * 0.02	☑
B		
C		
D		
E		
F		
G		

Ref	ID	Post	Alias	Text
A0				The Cash Back amount equals 2% of the total amount purchased

Note:

Parameterizing Rules - You might want to use another Vocabulary attribute (a "**parameter**") to hold a value, such as the percentage used in this formula, rather than "hard-coding" it (as in **0.02**).

When the value of an attribute is abstracted to a parameter such as **cashBackRate**, it can be changed without changing any related rules.

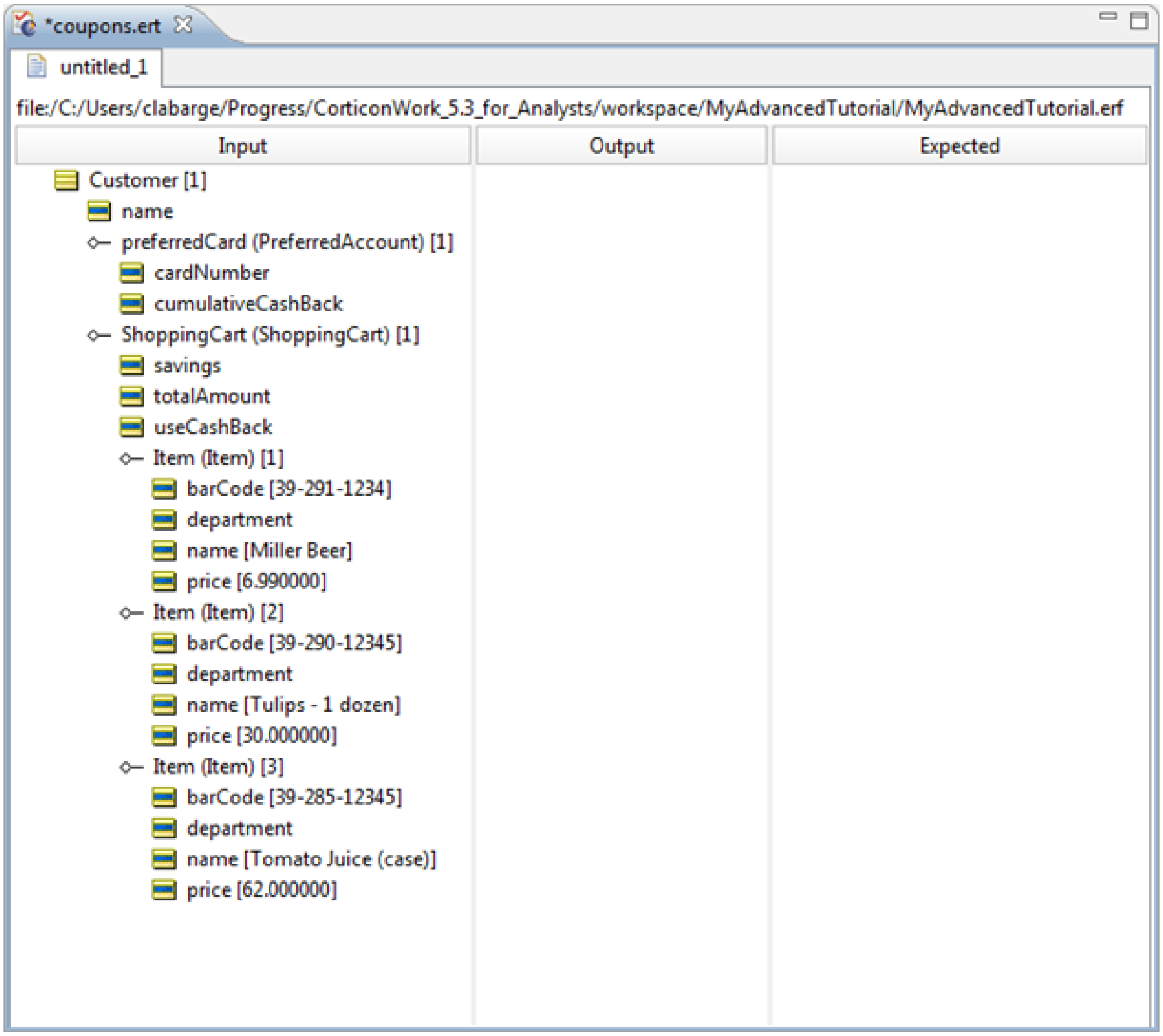
For more information on parameterization techniques, see the *Rule Modeling Guide*.

Testing cashBackEarned calculation

As this is our first test for Rulesheet two, `coupons.ers`, we could create the test entries. But they will be a lot like the test sheet for Rulesheet one, `checks.ers`. It is easy to copy a Ruletest.

To copy a Ruletest and adjust the details for the test:

1. In the Project Explorer view, right-click on `checks.ert`, choose **Copy**.
2. In the Project Explorer view, right-click then choose **Paste**. Name the copied file `coupons.ert`.
3. Double-click on `coupons.ert` to open it in its editor.
4. In the `coupons.ert` view, choose the menu command **Ruletest > Testsheet > ChangeTestSubject**. In the **Local** section expand `MyAdvancedTutorial`, and then choose our Ruleflow, `MyAdvancedTutorial.erf`.
5. For Item 2, double-click on each attribute to change its value as shown.
6. In the **Rule Vocabulary**, drag `Item` in the Customer's Shopping Cart to the `Shopping Cart` in the ruletest scope to create a third item. Double-click on each attribute to enter its values as shown:



*coupons.ert

untitled_1

file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

Input	Output	Expected

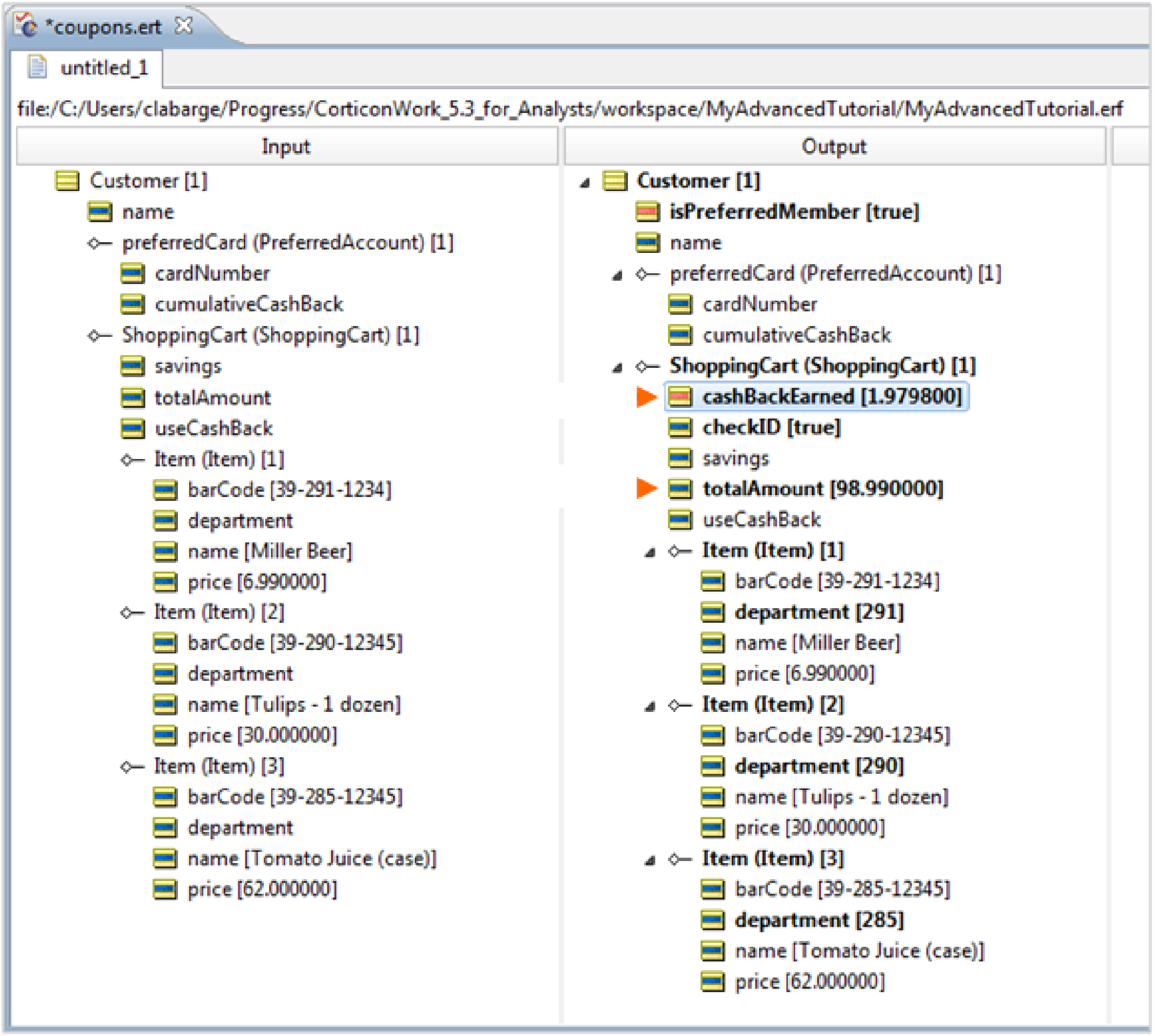
```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount
        useCashBack
        Item (Item) [1]
            barCode [39-291-1234]
            department
            name [Miller Beer]
            price [6.990000]
        Item (Item) [2]
            barCode [39-290-12345]
            department
            name [Tulips - 1 dozen]
            price [30.000000]
        Item (Item) [3]
            barCode [39-285-12345]
            department
            name [Tomato Juice (case)]
            price [62.000000]
```

7. Run the Ruletest.

CashBackEarned calculation test results

Notice that the `totalAmount` attribute now has a value of `$98.99` and the `cashBackEarned` attribute has been assigned a value of `$1.9798`, or 2% of $98.99.

Our rule has worked as expected.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

```
Input                                      Output
Customer [1]                               Customer [1]
  name                                       isPreferredMember [true]
  preferredCard (PreferredAccount) [1]       name
    cardNumber                               preferredCard (PreferredAccount) [1]
    cumulativeCashBack                         cardNumber
  ShoppingCart (ShoppingCart) [1]              cumulativeCashBack
    savings                                  ShoppingCart (ShoppingCart) [1]
    totalAmount                                cashBackEarned [1.979800]
    useCashBack                                checkID [true]
    Item (Item) [1]                            savings
      barCode [39-291-1234]                    totalAmount [98.990000]
      department                               useCashBack
      name [Miller Beer]                       Item (Item) [1]
      price [6.990000]                           barCode [39-291-1234]
    Item (Item) [2]                              department [291]
      barCode [39-290-12345]                     name [Miller Beer]
      department                                 price [6.990000]
      name [Tulips - 1 dozen]                  Item (Item) [2]
      price [30.000000]                          barCode [39-290-12345]
    Item (Item) [3]                              department [290]
      barCode [39-285-12345]                     name [Tulips - 1 dozen]
      department                                 price [30.000000]
      name [Tomato Juice (case)]               Item (Item) [3]
      price [62.000000]                          barCode [39-285-12345]
                                                 department [285]
                                                 name [Tomato Juice (case)]
                                                 price [62.000000]
```

Modeling and testing the cumulative cashBackEarned attribute

This section shows how to model and test the rule that defines the cumulative `cashBackEarned` attribute's value.

Modeling cumulative cashBackEarned

A customer's CashBack earned in this cart increases the amount in their CashBack account which has a balance of $10.00. Before we see how we can apply to the current cart, we need to determine the cumulative total. For that, we use the += operator. we also need to expose the calculated amount at the point of sale.

To add the current cashBack amount to the cumulative amount:

1. On the `coupons` Rulesheet's **Action** row **B**, enter

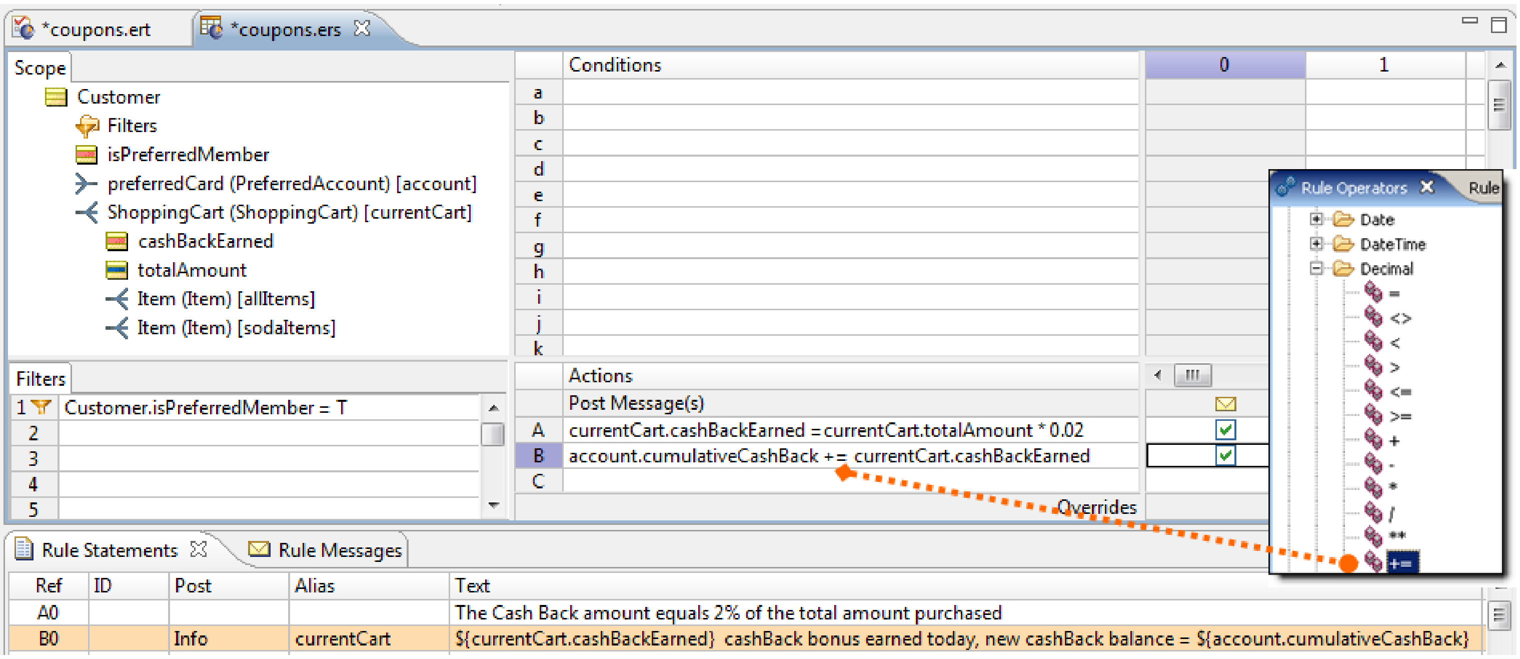
```
account.cumulativeCashBack += currentCart.cashBackEarned
```

Then check column **0** of this row, as this too is a non-conditional action.

2. In the Rule Statements panel, enter the reference `B0`, then choose **Post** `Info` and **Alias** `currentCart`

3. Enter the text and variables as follows:

   ```
   ${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack
   balance is ${account.cumulativeCashBack}.
   ```



Scope
- Customer
  - Filters
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
  - ShoppingCart (ShoppingCart) [currentCart]
    - cashBackEarned
    - totalAmount
    - Item (Item) [allItems]
    - Item (Item) [sodaItems]

Filters	
1	Customer.isPreferredMember = T
2	
3	
4	
5	

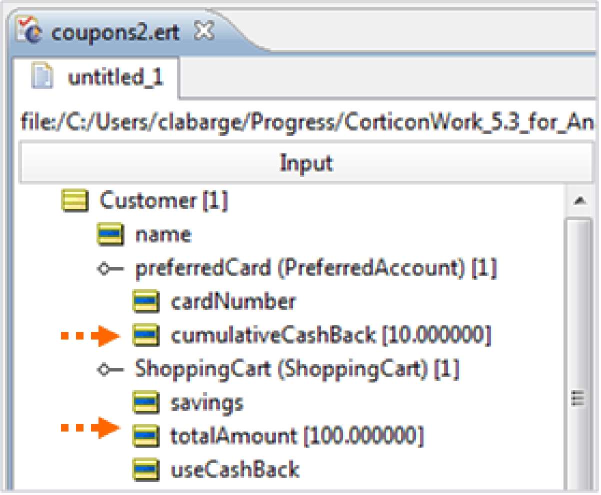
	Conditions	0	1
a			
b			
c			
d			
e			
f			
g			
h			
i			
j			
k			
	Actions		
	Post Message(s)	✉	
A	currentCart.cashBackEarned = currentCart.totalAmount * 0.02	☑	
B	account.cumulativeCashBack += currentCart.cashBackEarned	☑	
C			
	Overrides		

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
A0				The Cash Back amount equals 2% of the total amount purchased
B0		Info	currentCart	${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance = ${account.cumulativeCashBack}

Testing the cumulative cashBackEarned model

For this test, we adjusted the items (change `Miller Beer` to `8.00`) to make the `totalAmount` of `$100` for the `shoppingCart` and a `cumulativeCashBack` balance of `$10`.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_An

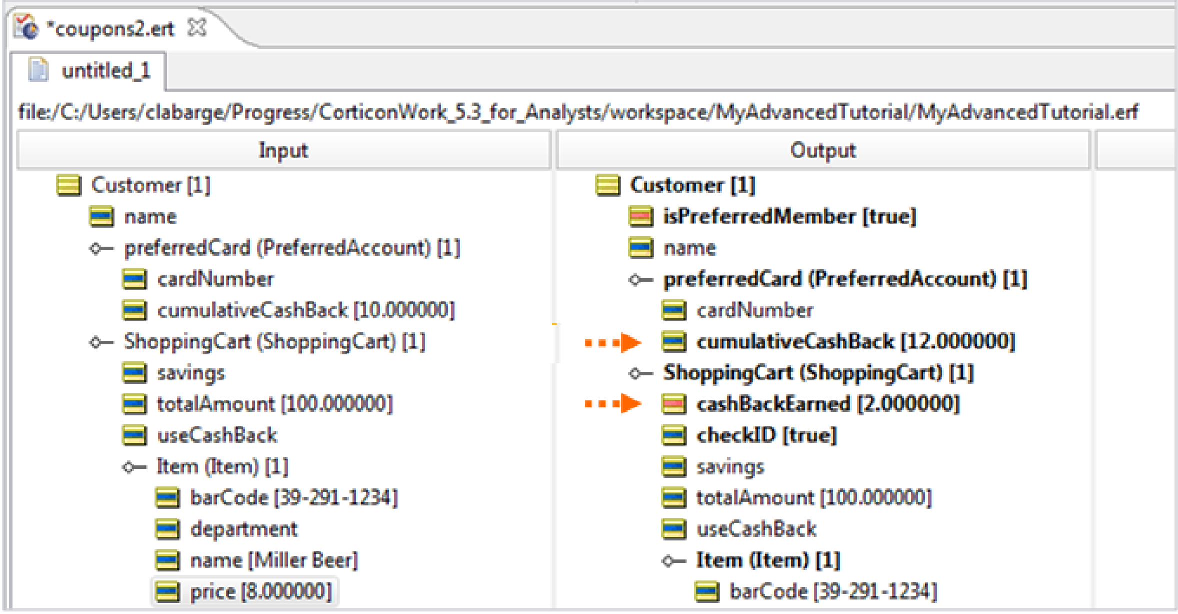
Input
- Customer [1]
  - name
  - preferredCard (PreferredAccount) [1]
    - cardNumber
    - cumulativeCashBack [10.000000]
  - ShoppingCart (ShoppingCart) [1]
    - savings
    - totalAmount [100.000000]
    - useCashBack

Note: When building Ruletests, it’s easy to forget that a Rulesheet’s **Filters**, if not satisfied, *might prevent your rules from executing.* The Rulesheet being tested here has a **Filter** expression that “filters out” all customers who aren’t **Preferred Card** members, so we needed to include an associated **preferredCard** entity in our test to ensure the **Filter** is satisfied, and our new rule model has a chance to execute.

Cumulative cashBackEarned test results

The results testsheet shows that the `cashBackEarned` has been calculated as **$2**, and `cumulativeCashBack` amount has been incremented from its original value of **$10** to a new value of **$12**.

Our rule model works as expected.



*coupons2.ert

untitled_1

file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

Input

```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [10.000000]
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount [100.000000]
        useCashBack
        Item (Item) [1]
            barCode [39-291-1234]
            department
            name [Miller Beer]
            price [8.000000]
```

Output

```
Customer [1]
    isPreferredMember [true]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [12.000000]
    ShoppingCart (ShoppingCart) [1]
        cashBackEarned [2.000000]
        checkID [true]
        savings
        totalAmount [100.000000]
        useCashBack
        Item (Item) [1]
            barCode [39-291-1234]
```

Note: The message has *embedded* attributes showing the dollar amounts. Later you will see how name values can also be embedded. This feature is discussed in the *Rule Language Guide*'s "Special Syntax" chapter.

Modeling and testing the first business rule

This section shows how to model and test the first business rule.

Modeling condition/action rule 1

The first Condition on this Rulesheet identifies any items purchased from department **290**, the **Floral department.** For each item identified, we want to give the customer a coupon for a free balloon. We use the `.new` operator to create a coupon.

To generate the floral department coupon:

1. In the `coupons.ers` rulesheet's **Conditions** section, on line **a** enter:

   ```
   allItems.department
   ```

   Then in column 1, enter `'290'`. (remember to use the single-quote marks to specify this text string.)

2. In **Action** row **C**, enter

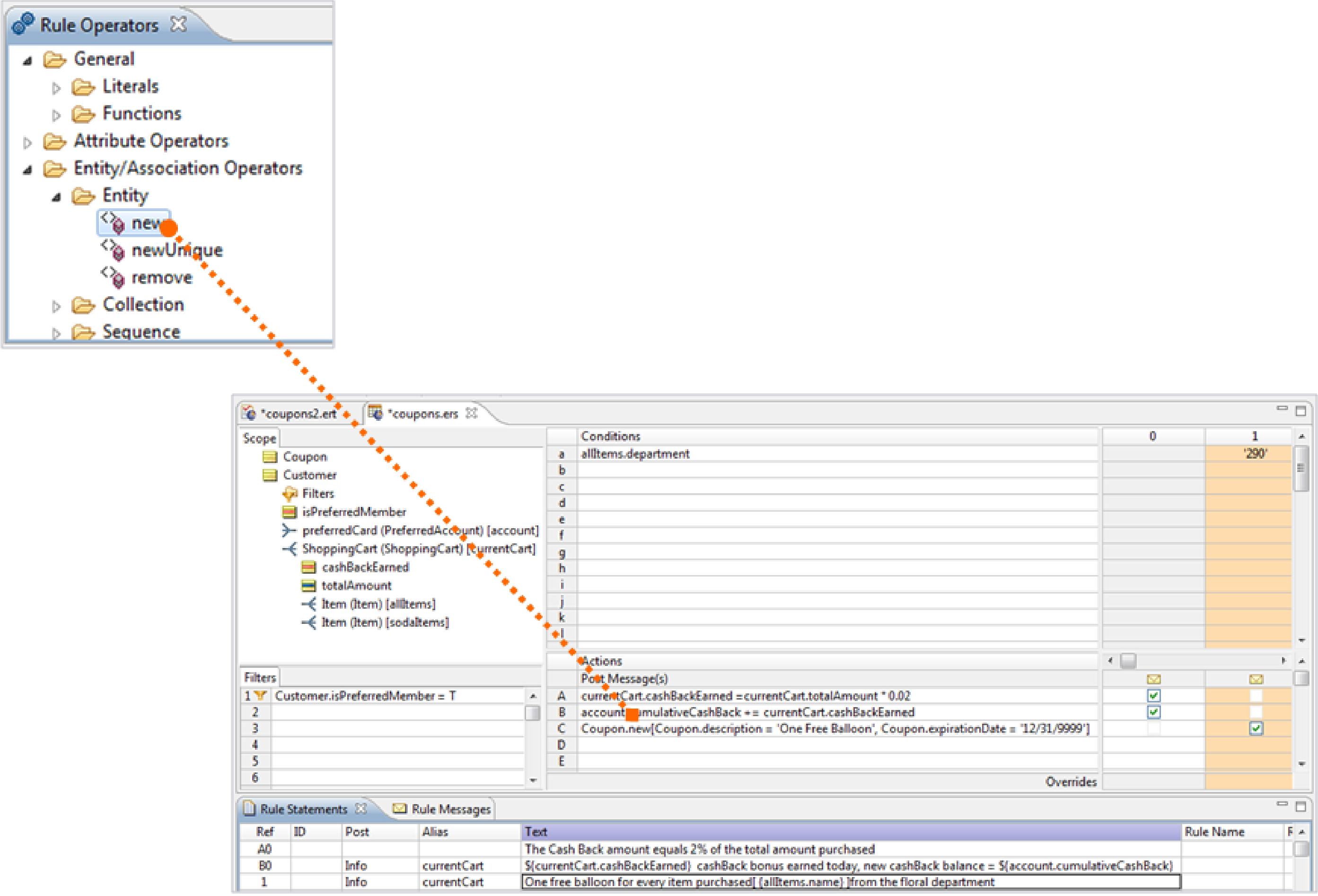
   ```
   Coupon.new[Coupon.description = 'One Free Balloon',
   Coupon.expirationDate='12/31/9999']
   ```

   Then check column **1** of this row, as this is a ***conditional*** action for the specified department.

3. In the Rule Statements panel, enter the reference `1`, then choose **Post** `Info` and **Alias** `currentCart`

4. Enter the text:

   ```
   One free balloon for every item purchased[{allItems.name}] from the
   floral department.
   ```



Filters	
1	Customer.isPreferredMember = T
2	
3	
4	
5	
6	

	Conditions	0	1
a	allItems.department		'290'
b			
c			
d			
e			
f			
g			
h			
i			
j			
k			
l			

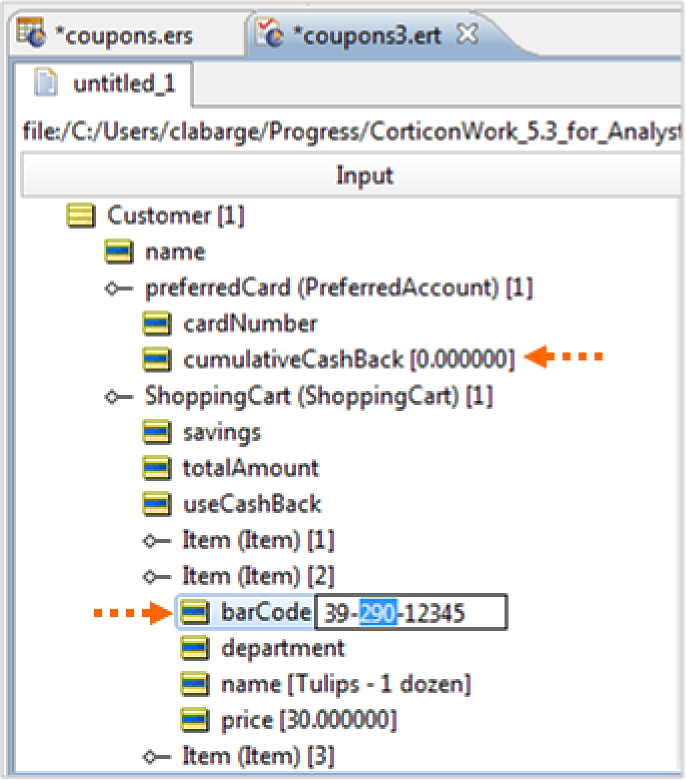
	Actions		
	Post Message(s)	✉	✉
A	currentCart.cashBackEarned =currentCart.totalAmount * 0.02	☑	□
B	account.cumulativeCashBack += currentCart.cashBackEarned	☑	□
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate = '12/31/9999']	□	☑
D			
E			
	Overrides		

Ref	ID	Post	Alias	Text	Rule Name
A0				The Cash Back amount equals 2% of the total amount purchased	
B0		Info	currentCart	${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance = ${account.cumulativeCashBack}	
1		Info	currentCart	One free balloon for every item purchased[{allItems.name}]from the floral department	

Note: The assignment of the value `12/31/9999` to the **expirationDate** attribute is one way to indicate that there is no realistic expiration of the coupon. You can design other ways to get the same effect. For example, the entity **Coupon** could have a boolean attribute named **expires**, to which a **true** or **false** value could be assigned inside the **.new** expression.

Testing condition/action rule 1

In this Ruletest, we want to confirm that when a Floral item has been purchased (department **290**) that a new **Coupon** is created entitling the customer to one free balloon.


*coupons.ers
*coupons3.ert
untitled_1
file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analyst
Input

```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [0.000000]
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount
        useCashBack
        Item (Item) [1]
        Item (Item) [2]
            barCode 39-290-12345
            department
            name [Tulips - 1 dozen]
            price [30.000000]
        Item (Item) [3]
```

The Ruletest so far **has** a floral item, `Tulips` so our test should work.

Test Results

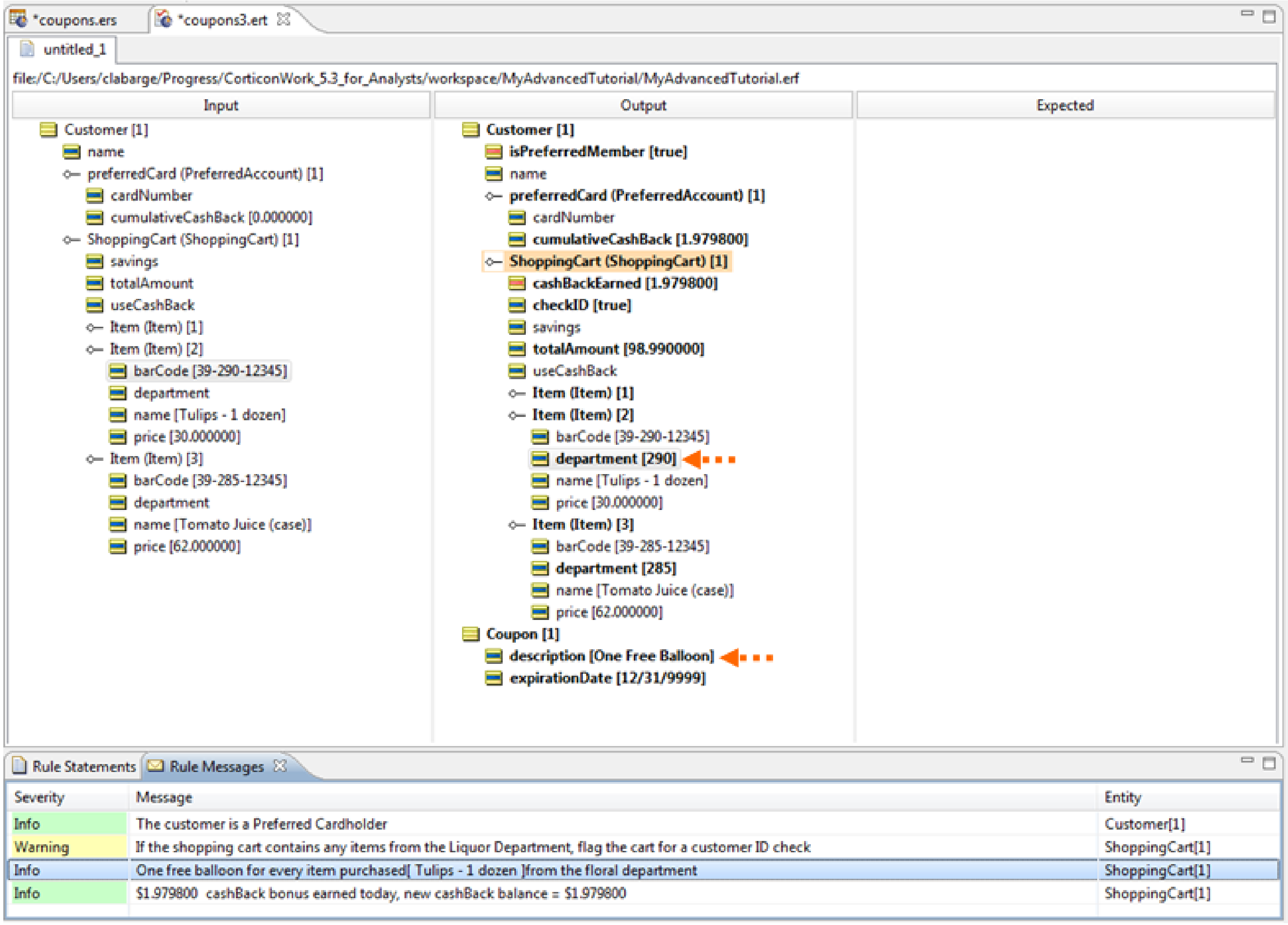
Now run the test. Note that the test data not only generates the coupon and numbers we expected, the liquor department item still fires the ID-check requirement.

Notice we also set **cumulativeCashBack** to **0** for this test. The formula (in Action row B, column 0) depends on a real value of **cumulativeCashBack** to increment. If its initial value is **null**, the rule will not fire.

More discussion on null values and their effects on rule execution can be found in the "Troubleshooting" chapter of the *Rule Modeling Guide*.

Results of testing condition/action rule 1

The floral item generated the results we expected, generating the `cashBack` information and the coupon notice.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

Input

```
Customer [1]
  name
  preferredCard (PreferredAccount) [1]
    cardNumber
    cumulativeCashBack [0.000000]
  ShoppingCart (ShoppingCart) [1]
    savings
    totalAmount
    useCashBack
    Item (Item) [1]
    Item (Item) [2]
      barCode [39-290-12345]
      department
      name [Tulips - 1 dozen]
      price [30.000000]
    Item (Item) [3]
      barCode [39-285-12345]
      department
      name [Tomato Juice (case)]
      price [62.000000]
```

Output

```
Customer [1]
  isPreferredMember [true]
  name
  preferredCard (PreferredAccount) [1]
    cardNumber
    cumulativeCashBack [1.979800]
  ShoppingCart (ShoppingCart) [1]
    cashBackEarned [1.979800]
    checkID [true]
    savings
    totalAmount [98.990000]
    useCashBack
    Item (Item) [1]
    Item (Item) [2]
      barCode [39-290-12345]
      department [290]
      name [Tulips - 1 dozen]
      price [30.000000]
    Item (Item) [3]
      barCode [39-285-12345]
      department [285]
      name [Tomato Juice (case)]
      price [62.000000]
Coupon [1]
  description [One Free Balloon]
  expirationDate [12/31/9999]
```

Expected

Severity	Message	Entity
Info	The customer is a Preferred Cardholder	Customer[1]
Warning	If the shopping cart contains any items from the Liquor Department, flag the cart for a customer ID check	ShoppingCart[1]
Info	One free balloon for every item purchased[Tulips - 1 dozen]from the floral department	ShoppingCart[1]
Info	$1.979800 cashBack bonus earned today, new cashBack balance = $1.979800	ShoppingCart[1]

Modeling and testing the second business rule using filters

This section shows how to model and test the second business rule using filters.

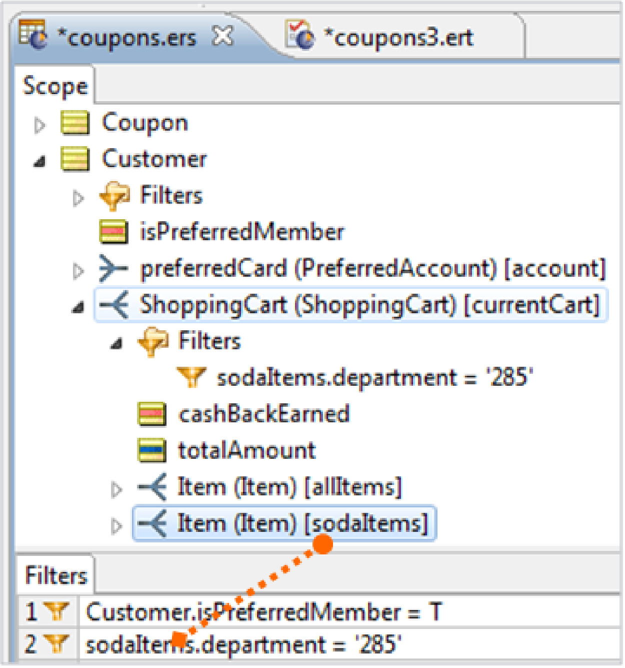
Modeling condition/action rule 2 using filter

Filters

The next Business Rule to be modeled creates a "**$2 off**" coupon when a customer buys three or more items from the **Soda** department. When determining whether any items from the **Floral** department were in the shopping cart, we used the **allItems** alias in Condition/Action rule 1. But to determine if three or more items were purchased from the Soda department, we do not want to count *all items* in the shopping cart, just those from the Soda department. The **sodaItems** alias we defined earlier in the **Scope** section will be focused on only those items.

A **Filter** expression will reduce the size of the collection, and trigger the coupon if the required count of items that *survived* the filter hits the threshold.

Note: To reduce the collection of items in the shopping cart to only those we want to count, we will use a **Filter** expression to filter the **SodaItems** alias. **Filters** row 2 ensures that the “surviving” members of the **SodaItems** alias all have a **department** value of **285**, which is that part of the **barCode** that identifies the Soda Department. If you drag item from the Vocabulary window, you may need to edit the spelling to the **sodaItems** alias. This is a case where dragging the **sodaItems** alias directly from the **Scope** window may be more convenient, although doing so requires you to type `.department` manually.



*coupons.ers   *coupons3.ert

Scope
- Coupon
- Customer
  - Filters
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
  - ShoppingCart (ShoppingCart) [currentCart]
    - Filters
      - sodaItems.department = '285'
    - cashBackEarned
    - totalAmount
    - Item (Item) [allItems]
    - Item (Item) [sodaItems]

Filters

1	Customer.isPreferredMember = T
2	sodaItems.department = '285'

To generate the soda department coupon:

1. In the `coupons.ers` rulesheet's **Filters** section, on line **2** enter:

   `sodaItems.department='285'.`

2. In the Rulesheet's **Conditions** section, on line **b** enter:

   `sodaItems -> size >=3`

   Then in column 2, enter `T`.

   Note: The `->size` operator counts the number of elements in the SodaItems collection of data. If three or greater, then the coupon is issued to the customer.

3. In **Action** row **D**, enter (as one line of text):

   ```
   Coupon.new[Coupon.description = '$2 off next purchase',
   Coupon.expirationDate=today.addYears(1)]
   ```

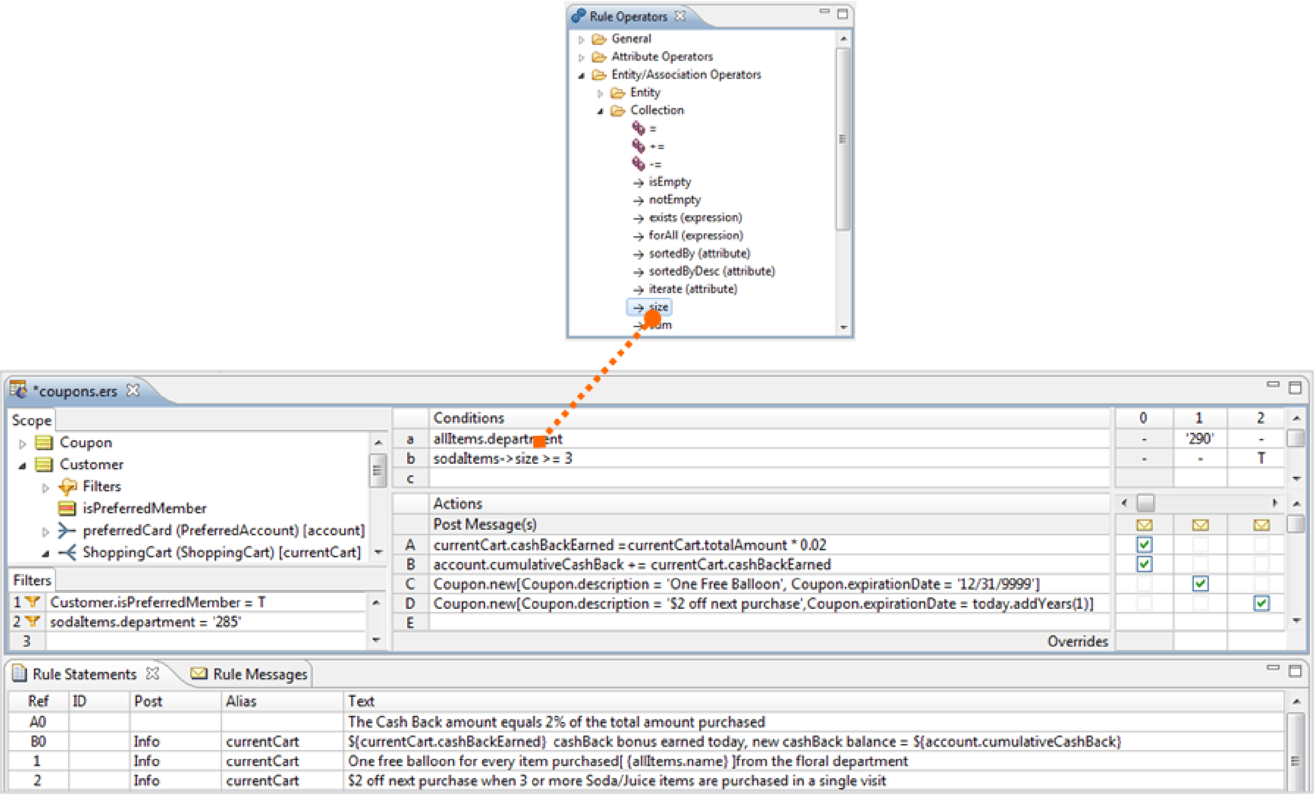
   Then check column **2** of this row, as this is a conditional action for the specified department.

   Note: The `expirationDate` attribute gets its value from use of the `.addYears` operator, set here to 1, so we know that the coupon will expire one year from its date of issue.

4. In the **Rule Statements** panel, enter the reference `2`, then choose **Post** `Info` and **Alias** `currentCart`.

5. Enter the text:

   `$2 off next purchase when 3 or more Soda/Juice items are purchased in a single visit.`



Filters	
1	Customer.isPreferredMember = T
2	sodaItems.department = '285'
3	

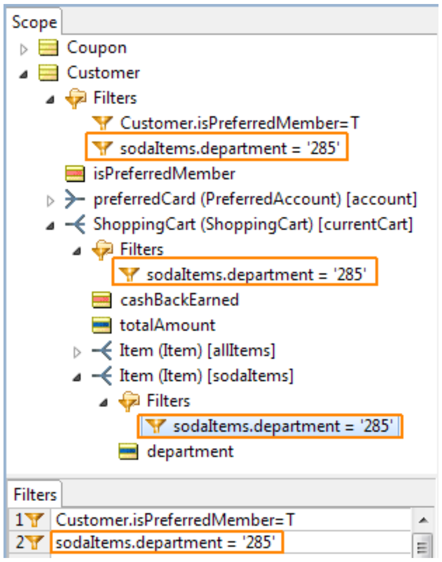
	Conditions	0	1	2
a	allItems.department	-	'290'	-
b	sodaItems->size >= 3	-	-	T
c				

	Actions			
	Post Message(s)	✉	✉	✉
A	currentCart.cashBackEarned =currentCart.totalAmount * 0.02	☑	□	□
B	account.cumulativeCashBack += currentCart.cashBackEarned	☑	□	□
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate = '12/31/9999']	□	☑	□
D	Coupon.new[Coupon.description = '$2 off next purchase',Coupon.expirationDate = today.addYears(1)]	□	□	☑
E				
	Overrides			

Ref	ID	Post	Alias	Text
A0				The Cash Back amount equals 2% of the total amount purchased
B0		Info	currentCart	${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance = ${account.cumulativeCashBack}
1		Info	currentCart	One free balloon for every item purchased[{allItems.name}]from the floral department
2		Info	currentCart	$2 off next purchase when 3 or more Soda/Juice items are purchased in a single visit

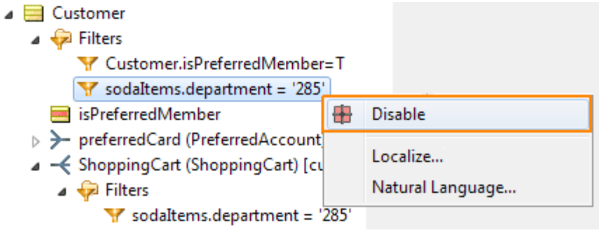
Limiting Filters

The filter we created is applied to every relevant level in the scope:

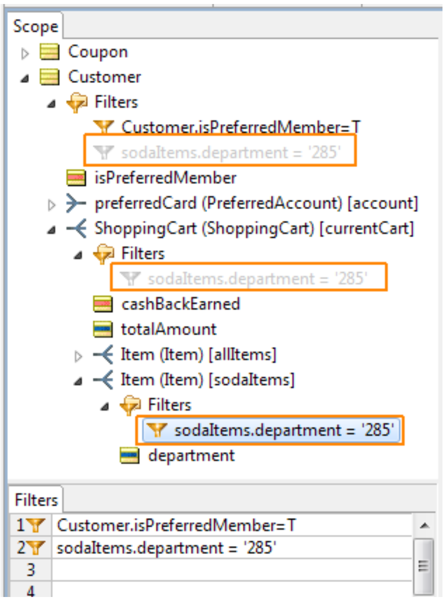

Scope
Coupon
Customer
Filters
Customer.isPreferredMember=T
sodaItems.department = '285'
isPreferredMember
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]
Filters
sodaItems.department = '285'
cashBackEarned
totalAmount
Item (Item) [allItems]
Item (Item) [sodaItems]
Filters
sodaItems.department = '285'
department

Filters	
1	Customer.isPreferredMember=T
2	sodaItems.department = '285'

It is a ***full filter***, applying to the Customer, the currentCart, and -- the level we want to filter -- the Items. We can disable it at selected levels to make it a ***limiting filter*** by right-clicking on a filter level and then selecting **Disable**:


Customer
Filters
Customer.isPreferredMember=T
sodaItems.department = '285'
isPreferredMember
preferredCard (PreferredAccount
ShoppingCart (ShoppingCart) [cu
Filters
sodaItems.department = '285'
Disable
Localize...
Natural Language...

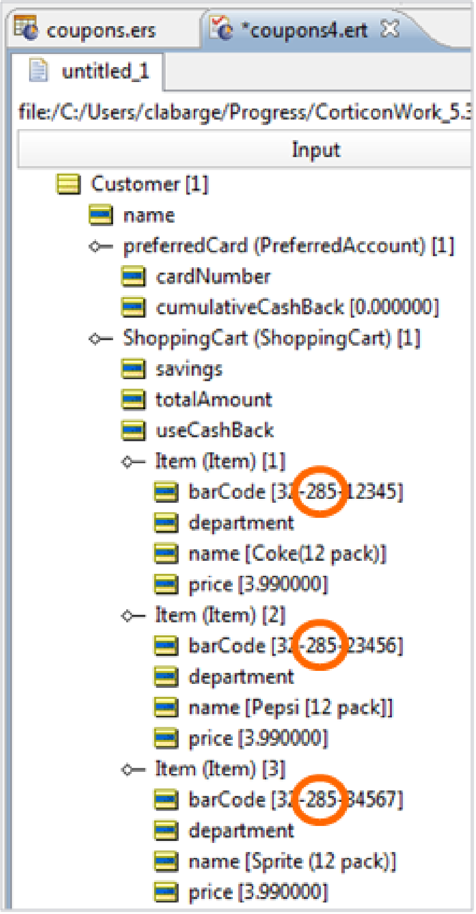
When we disable the filter on the Customer and currentCart, their values are greyed out, as shown:


Scope
Coupon
Customer
Filters
Customer.isPreferredMember=T
sodaItems.department = '285'
isPreferredMember
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]
Filters
sodaItems.department = '285'
cashBackEarned
totalAmount
Item (Item) [allItems]
Item (Item) [sodaItems]
Filters
sodaItems.department = '285'
department
Filters
1 Customer.isPreferredMember=T
2 sodaItems.department = '285'
3
4

Note: Refer to the *Corticon Studio: Rule Language Guide* for details about these operators, as well as other operators available in Corticon.

Testing condition/action rule 2

The Preferred Customer's shopping cart must have three or more items from the **Soda Department** (department **285**). We can change the test data to generate the coupon and numbers we expect by editing the attributes of each of the three items such that they are in department `285`, and have the names and prices as shown:



file:/C:/Users/clabarge/Progress/CorticonWork_5.3

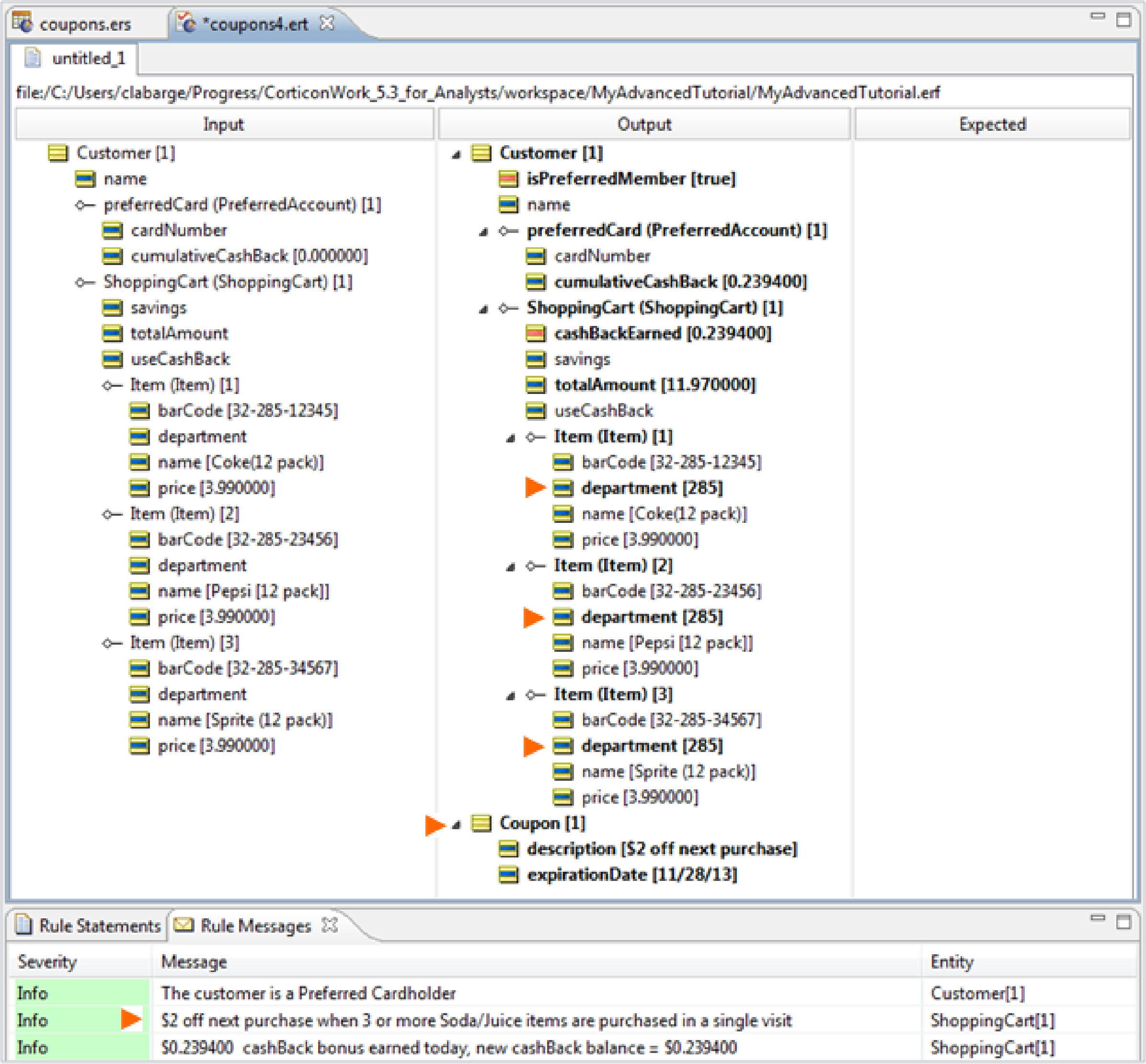
Input

- Customer [1]
  - name
  - preferredCard (PreferredAccount) [1]
    - cardNumber
    - cumulativeCashBack [0.000000]
  - ShoppingCart (ShoppingCart) [1]
    - savings
    - totalAmount
    - useCashBack
    - Item (Item) [1]
      - barCode [32-285-12345]
      - department
      - name [Coke(12 pack)]
      - price [3.990000]
    - Item (Item) [2]
      - barCode [32-285-23456]
      - department
      - name [Pepsi [12 pack]]
      - price [3.990000]
    - Item (Item) [3]
      - barCode [32-285-34567]
      - department
      - name [Sprite (12 pack)]
      - price [3.990000]

Condition/action rule 2 test results

The three items from the **Soda** department generated a coupon for $2 off the next purchase, and an **expirationDate** of one year from today.

Coupon rule 2 works as expected.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

Input

```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [0.000000]
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount
        useCashBack
        Item (Item) [1]
            barCode [32-285-12345]
            department
            name [Coke(12 pack)]
            price [3.990000]
        Item (Item) [2]
            barCode [32-285-23456]
            department
            name [Pepsi [12 pack]]
            price [3.990000]
        Item (Item) [3]
            barCode [32-285-34567]
            department
            name [Sprite (12 pack)]
            price [3.990000]
```

Output

```
Customer [1]
    isPreferredMember [true]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [0.239400]
    ShoppingCart (ShoppingCart) [1]
        cashBackEarned [0.239400]
        savings
        totalAmount [11.970000]
        useCashBack
        Item (Item) [1]
            barCode [32-285-12345]
          ▶ department [285]
            name [Coke(12 pack)]
            price [3.990000]
        Item (Item) [2]
            barCode [32-285-23456]
          ▶ department [285]
            name [Pepsi [12 pack]]
            price [3.990000]
        Item (Item) [3]
            barCode [32-285-34567]
          ▶ department [285]
            name [Sprite (12 pack)]
            price [3.990000]
▶ Coupon [1]
    description [$2 off next purchase]
    expirationDate [11/28/13]
```

Expected

Severity	Message	Entity
Info	The customer is a Preferred Cardholder	Customer[1]
Info ▶	$2 off next purchase when 3 or more Soda/Juice items are purchased in a single visit	ShoppingCart[1]
Info	$0.239400 cashBack bonus earned today, new cashBack balance = $0.239400	ShoppingCart[1]

Regression Testing

You can retest the performance of other rules by simply modifying the items. Change the department code of one of the items to `291` (liquor) and another one to `290` (floral). When you run the test, the alert to check identification (liquor) and the free balloon coupon (floral) are generated. The soda coupon does not get generated as you no longer have three soda items. All three rules are behaving as intended.

Modeling and testing the third business rule

This section shows how to model and test the third business rule.

Modeling condition/action rule 3

The third condition on our Rulesheet is used to identify when a customer's `totalAmount` crosses the **$75** threshold defined in the scenario. The reward is a new coupon (again, using the **.new** operator) for **10% off** a future gasoline purchase at our store's gas pumps.

The `expirationDate` attribute derives its value from the **.addMonths** operator, set here to 3, so the coupon will expire three months from its date of issue.

As always, best practice recommends adding the corresponding Rule Statement, explaining in clear language what the business rule does.

To generate the minimum purchase coupon:

1. In the `coupons.ers` rulesheet's **Conditions** section, on line **c** enter:

   ```
   currentCart.totalAmount > 75
   ```

   Then in column 3, enter `T`.

2. In **Action** row **E**, enter

   ```
   Coupon.new[Coupon.description = '10% off next gas purchase',
   Coupon.expirationDate=today.addMonths(3)]
   ```

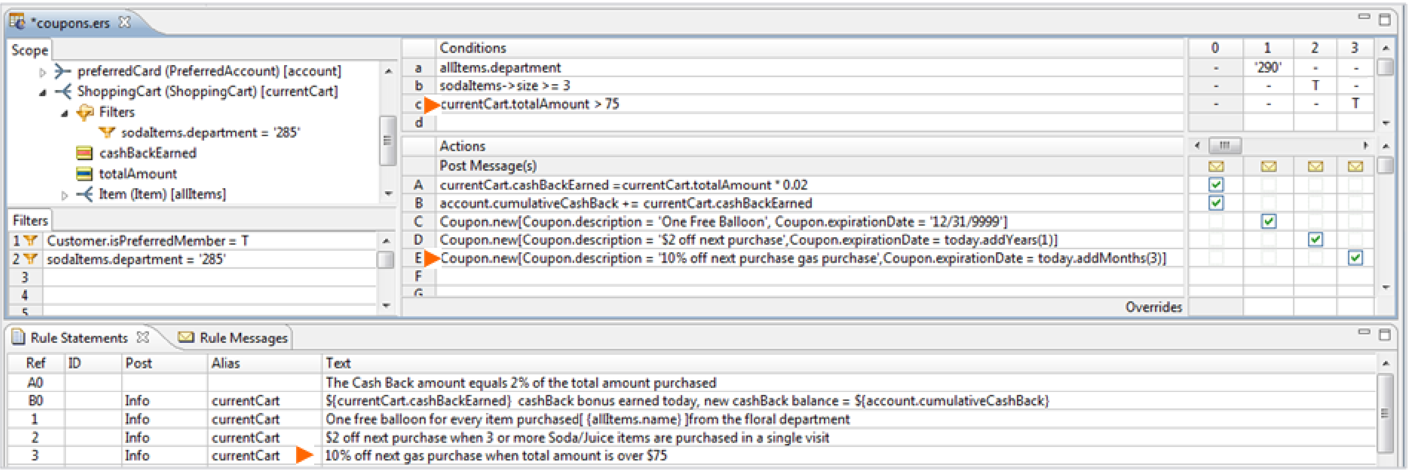
   Then check column **3** of this row, as this is a conditional action for the specified department.

3. In the Rule Statements panel, enter the reference `3`, then choose **Post** `Info` and **Alias** `currentCart`

4. Enter the text:

   ```
   10% off next gas purchase when total is over $75.
   ```

The Rulesheet now looks like this:



*coupons.ers
Scope
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]
Filters
sodaItems.department = '285'
cashBackEarned
totalAmount
Item (Item) [allItems]

Filters	
1	Customer.isPreferredMember = T
2	sodaItems.department = '285'
3	
4	
5	

	Conditions	0	1	2	3
a	allItems.department	-	'290'	-	-
b	sodaItems->size >= 3	-	-	T	-
c	currentCart.totalAmount > 75	-	-	-	T
d					

	Actions				
	Post Message(s)	✉	✉	✉	✉
A	currentCart.cashBackEarned =currentCart.totalAmount * 0.02	☑			
B	account.cumulativeCashBack += currentCart.cashBackEarned	☑			
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate = '12/31/9999']		☑		
D	Coupon.new[Coupon.description = '$2 off next purchase',Coupon.expirationDate = today.addYears(1)]			☑	
E	Coupon.new[Coupon.description = '10% off next purchase gas purchase',Coupon.expirationDate = today.addMonths(3)]				☑
F					
G					
	Overrides				

Ref	ID	Post	Alias	Text
A0				The Cash Back amount equals 2% of the total amount purchased
B0		Info	currentCart	${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance = ${account.cumulativeCashBack}
1		Info	currentCart	One free balloon for every item purchased[{allItems.name}]from the floral department
2		Info	currentCart	$2 off next purchase when 3 or more Soda/Juice items are purchased in a single visit
3		Info	currentCart	10% off next gas purchase when total amount is over $75

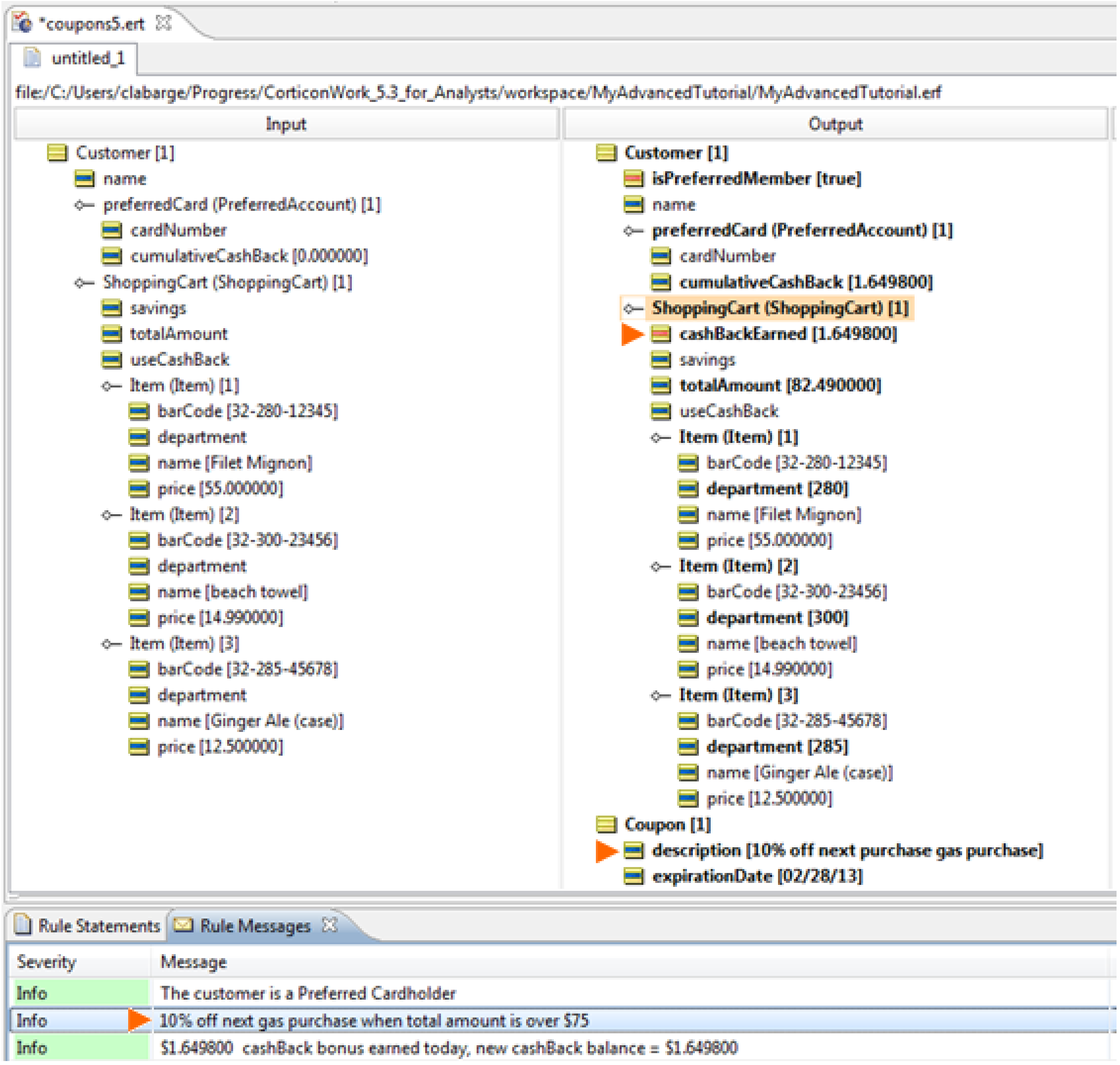
Testing condition/action rule 3

For a Total Amount test, we need to include items in the shopping cart that add up to more than **$75** in order to generate a **10% off** gas coupon for the customer. The data should be at a total of $100, but if it is not, change the price of one or more one items to push the total or $75.

Condition/action rule 3 test results

The items have been totaled and the amount exceeds the **$75** threshold so the gas discount **Coupon** was created and the **info** message was posted.

All the Condition/Action rule works as intended.



*coupons5.ert ☒

untitled_1

file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutorial.erf

```
Input
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [0.000000]
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount
        useCashBack
        Item (Item) [1]
            barCode [32-280-12345]
            department
            name [Filet Mignon]
            price [55.000000]
        Item (Item) [2]
            barCode [32-300-23456]
            department
            name [beach towel]
            price [14.990000]
        Item (Item) [3]
            barCode [32-285-45678]
            department
            name [Ginger Ale (case)]
            price [12.500000]
```

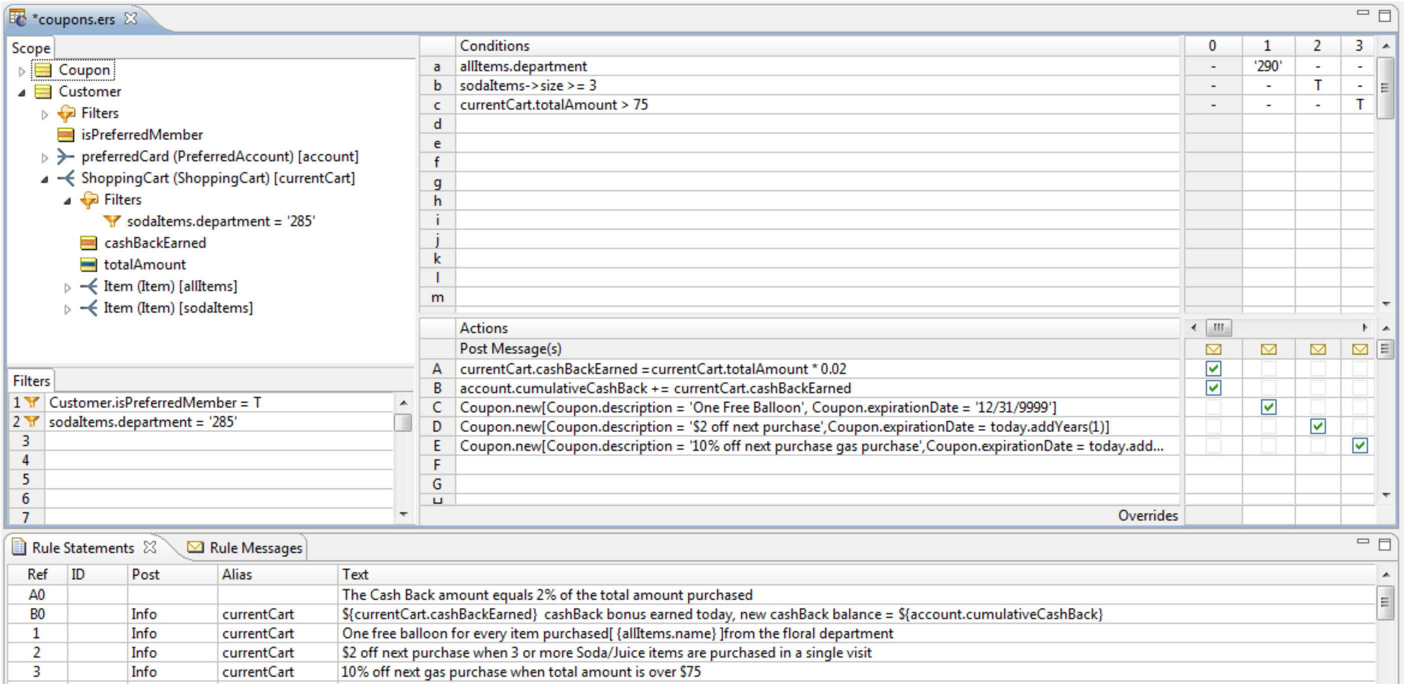
```
Output
Customer [1]
    isPreferredMember [true]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [1.649800]
    ShoppingCart (ShoppingCart) [1]
        cashBackEarned [1.649800]
        savings
        totalAmount [82.490000]
        useCashBack
        Item (Item) [1]
            barCode [32-280-12345]
            department [280]
            name [Filet Mignon]
            price [55.000000]
        Item (Item) [2]
            barCode [32-300-23456]
            department [300]
            name [beach towel]
            price [14.990000]
        Item (Item) [3]
            barCode [32-285-45678]
            department [285]
            name [Ginger Ale (case)]
            price [12.500000]
Coupon [1]
    description [10% off next purchase gas purchase]
    expirationDate [02/28/13]
```

Rule Statements | Rule Messages ☒

Severity	Message
Info	The customer is a Preferred Cardholder
Info	10% off next gas purchase when total amount is over $75
Info	$1.649800 cashBack bonus earned today, new cashBack balance = $1.649800

Rulesheet two is complete and tested

The completed second Rulesheet looks like this:



*coupons.ers

Scope

- Coupon
- Customer
  - Filters
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
  - ShoppingCart (ShoppingCart) [currentCart]
    - Filters
      - sodaItems.department = '285'
    - cashBackEarned
    - totalAmount
    - Item (Item) [allItems]
    - Item (Item) [sodaItems]

Filters	
1	Customer.isPreferredMember = T
2	sodaItems.department = '285'
3	
4	
5	
6	
7	

	Conditions	0	1	2	3
a	allItems.department	-	'290'	-	-
b	sodaItems->size >= 3	-	-	T	-
c	currentCart.totalAmount > 75	-	-	-	T
d					
e					
f					
g					
h					
i					
j					
k					
l					
m					

	Actions				
	Post Message(s)	✉	✉	✉	✉
A	currentCart.cashBackEarned =currentCart.totalAmount * 0.02	☑	□	□	□
B	account.cumulativeCashBack += currentCart.cashBackEarned	☑	□	□	□
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate = '12/31/9999']	□	☑	□	□
D	Coupon.new[Coupon.description = '$2 off next purchase',Coupon.expirationDate = today.addYears(1)]	□	□	☑	□
E	Coupon.new[Coupon.description = '10% off next purchase gas purchase',Coupon.expirationDate = today.add...	□	□	□	☑
F					
G					
	Overrides				

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
A0				The Cash Back amount equals 2% of the total amount purchased
B0		Info	currentCart	${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance = ${account.cumulativeCashBack}
1		Info	currentCart	One free balloon for every item purchased[{allItems.name}]from the floral department
2		Info	currentCart	$2 off next purchase when 3 or more Soda/Juice items are purchased in a single visit
3		Info	currentCart	10% off next gas purchase when total amount is over $75

6

Modeling and testing the 'use_cashBack' Rulesheet

This chapter shows how to model and test rules for the third Rulesheet.

For details, see the following topics:

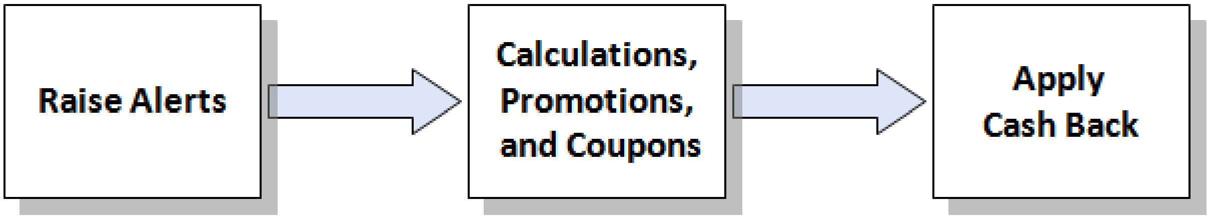
Modeling the third Rulesheet

The first Rulesheet calculated the **cashBack** earned by a **Preferred Card** member for each purchase and incremented the member's **cumulativeCashBack** amount.

Now, let's give the shopper the option of using the money in the **cumulativeCashBack** account to reduce the total amount due at checkout. We will assume that at time of checkout, the cashier asks the shopper if the shopper wants to apply the **cumulativeCashBack** amount to the current purchase **totalAmount**. If the shopper says "Yes", then we assume the shopping cart's useCashBack attribute is true. If the shopper answers "No" then the attribute is **false**.

If **useCashBack** is **true**, then we need to deduct it from the **totalAmount**, thereby reducing the amount the shopper pays.

Finally, when a shopper applies the balance in his **cumulativeCashBack** account, we need to reset that balance to zero.



Apply Cash Back

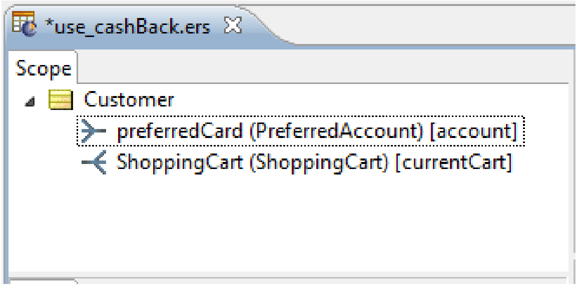
- A Preferred Shopper account will track the accumulated cash back and allow the customer to apply it to any visit's total amount. The cashier will ask a Preferred Shopper if he/she would like to apply a cash back balance to his/her current purchase
- Once a Preferred Shopper chooses to apply his cash back balance, the cumulative cash back total maintained by the system will be reset to zero, and the accumulation of cash back begins anew with the customer's next purchase.

Scope of the third Rulesheet

As we build the third and final Rulesheet, we will add it to the ruleflow immediately. By adding the third Rulesheet to the RuleFlow for the project, and chaining the Rulesheets into sequence, we'll be able to run all the tests from end to end when we complete and test the third Rulesheet.

To create the new Rulesheet's scope:

1. Select the menu command **New > Rulesheet**. Choose the parent folder `MyAdvancedTutorial` and name the file `use_cashBack`.
2. Display the **Scope** section of the Rulesheet. Either click ▦ in the toolbar, or select **Rulesheet** > **Advanced View** from Studio's menubar.
3. In the **Rule Vocabulary**, click and drag the **Customer** entity to the **Scope** area of the rulesheet.
4. Click and drag the `PreferredCard` *association* within the customer, and drop it on the `Customer` in the rulesheet.
5. Double-click `PreferredCard` in the Rulesheet to open its alias entry box. Enter **account**.
6. Click and drag the `ShoppingCart` *association* within the customer, and drop it on the `Customer` in the rulesheet.
7. Double-click the ShoppingCart in the rulesheet to open its alias entry box. Enter **currentCart**.



*use_cashBack.ers

Scope

Customer

preferredCard (PreferredAccount) [account]

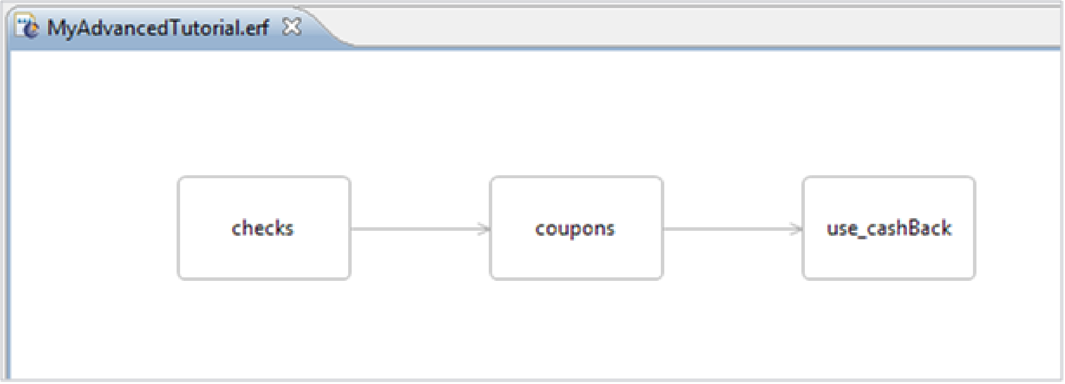
ShoppingCart (ShoppingCart) [currentCart]

Extending the Ruleflow

We are already testing using a Ruleflow as our test subject so just adding the new Rulesheet to it will draw it into the execution sequence.

To extend the Ruleflow:

1. Bring the Ruleflow file `MyAdvancedTutorial.erf` to the canvas.
2. In **ProjectExplorer**, click and drag `use_cashback.ers` to the ruleflow canvas to the right of `coupons`:
3. Click on **Connection** on the **Ruleflow** palette, then click on the `coupons` box, and drag and drop on the `use_cashBack` box to create the complete flow, as shown:



Save the updated Ruleflow. When we test Rulesheet three, the three Rulesheets will run in sequence.

Using filters

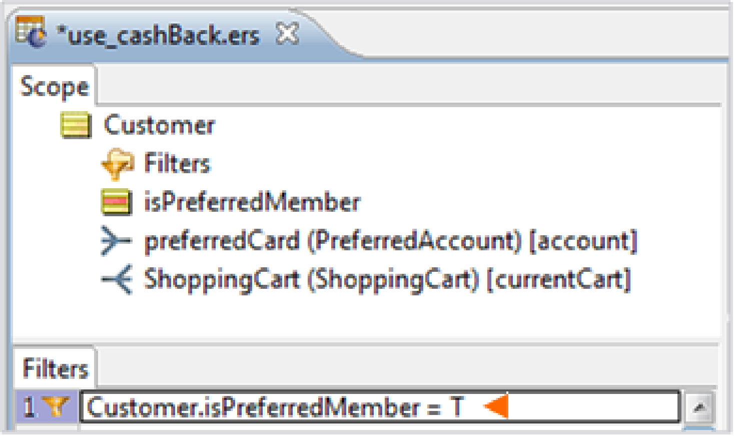
The first thing we want to accomplish with this Rulesheet is to make sure we only evaluate preferred customers since only they are eligible for the cash back and bonus incentives.

The expression in **Filters** row 1 of the Rulesheet "filters out" those customers that are not preferred members (because they do not have a Preferred Card).

To create a filter:

In the `use_cashBack` rulesheet's **Filters** section, on line 1, enter:

```
Customer.isPreferredMember = T
```



Customer
Filters
isPreferredMember
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]

1 Customer.isPreferredMember = T

Modeling and testing the first business rule

This section shows how to model and test the first business rule.

Modeling condition/action rule 1

We need to create one Condition/Action rule with one **Condition** and a few **Actions**. As illustrated, we want to process the **currentCart** when the shopper has chosen to apply their **cashBack** balance to the current purchase.

That will reduce the cumulative CashBack balance by total Amount in the cart.

To define Condition/Action rule one:

1. In the `use_cashback.ers` rulesheet's **Conditions** section, on line **a** enter:

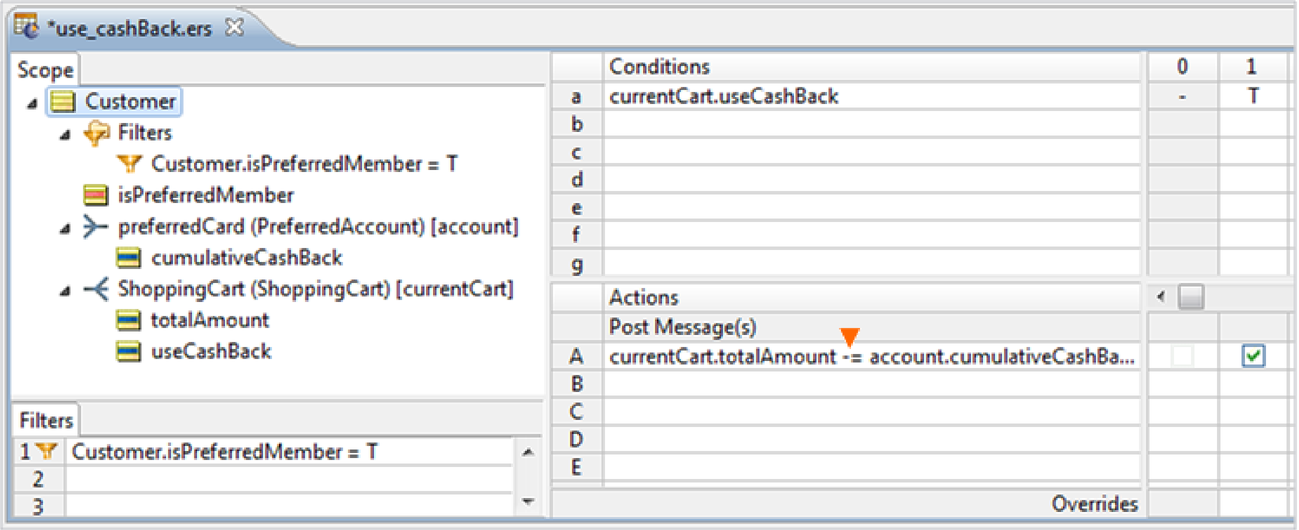
   ```
   currentCart.useCashBack
   ```

   Then, in column 1, enter or select `T`.

2. In **Action** row **A**, enter

   ```
   currentCart.totalAmount -= account.cumulativeCashBack
   ```

   Then check column **1** of this row, as this is a conditional action, triggered when the condition is `true`.



*use_cashBack.ers

Scope

- Customer
  - Filters
    - Customer.isPreferredMember = T
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
    - cumulativeCashBack
  - ShoppingCart (ShoppingCart) [currentCart]
    - totalAmount
    - useCashBack

	Filters
1	Customer.isPreferredMember = T
2	
3	

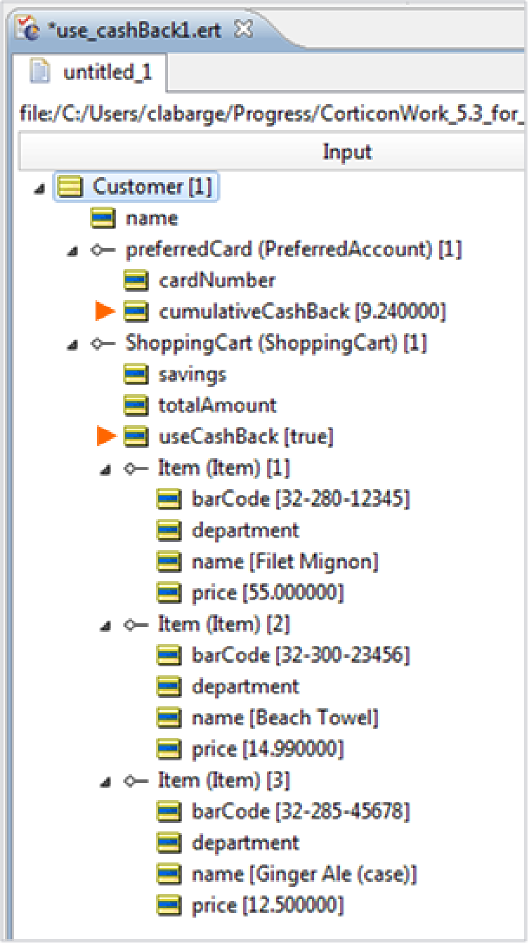
	Conditions	0	1
a	currentCart.useCashBack	-	T
b			
c			
d			
e			
f			
g			
	Actions		
	Post Message(s)		
A	currentCart.totalAmount -= account.cumulativeCashBa...	□	☑
B			
C			
D			
E			
	Overrides		

We will test the rule before adding more conditions or actions. We will postpone adding a **Rule Statement** until we have completed the rule model.

Testing condition/action rule 1

For the cash-back test, we have manually entered **$9.24** in the preferred customer's **cumulativeCashBack** attribute and indicated that the customer wants to apply this balance towards today's **totalAmount** (**useCashBack = true**)

We can use our current `coupons.ert` because we have set the ruleflow as its test subject.



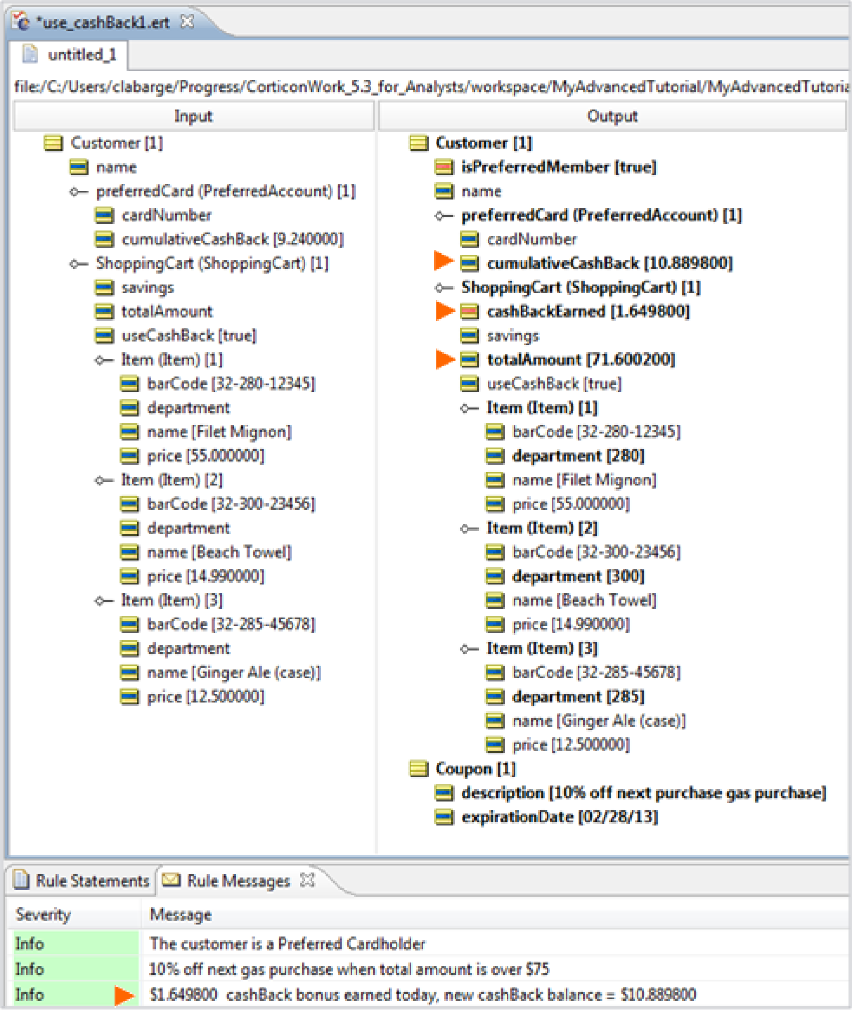
```
Customer [1]
    name
    preferredCard (PreferredAccount) [1]
        cardNumber
        cumulativeCashBack [9.240000]
    ShoppingCart (ShoppingCart) [1]
        savings
        totalAmount
        useCashBack [true]
        Item (Item) [1]
            barCode [32-280-12345]
            department
            name [Filet Mignon]
            price [55.000000]
        Item (Item) [2]
            barCode [32-300-23456]
            department
            name [Beach Towel]
            price [14.990000]
        Item (Item) [3]
            barCode [32-285-45678]
            department
            name [Ginger Ale (case)]
            price [12.500000]
```

According to our first **Condition/Action** rule, the **cumulativeCashBack** should first be incremented by the new cashBack earned by today's purchase, then subtracted from the **totalAmount** to arrive at the amount due.

Condition/action rule 1 test results

The Output panel shows the new **cashBackEarned** (**$1.64**) added to **cumulativeCashBack** (**$10.88**) and subtracted from **totalAmount** (**$71.60**).

We see these values embedded in the message Rulesheet two generated into the message area.



file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts/workspace/MyAdvancedTutorial/MyAdvancedTutoria

```
Input
Customer [1]
  name
  preferredCard (PreferredAccount) [1]
    cardNumber
    cumulativeCashBack [9.240000]
  ShoppingCart (ShoppingCart) [1]
    savings
    totalAmount
    useCashBack [true]
    Item (Item) [1]
      barCode [32-280-12345]
      department
      name [Filet Mignon]
      price [55.000000]
    Item (Item) [2]
      barCode [32-300-23456]
      department
      name [Beach Towel]
      price [14.990000]
    Item (Item) [3]
      barCode [32-285-45678]
      department
      name [Ginger Ale (case)]
      price [12.500000]
```

```
Output
Customer [1]
  isPreferredMember [true]
  name
  preferredCard (PreferredAccount) [1]
    cardNumber
    cumulativeCashBack [10.889800]
  ShoppingCart (ShoppingCart) [1]
    cashBackEarned [1.649800]
    savings
    totalAmount [71.600200]
    useCashBack [true]
    Item (Item) [1]
      barCode [32-280-12345]
      department [280]
      name [Filet Mignon]
      price [55.000000]
    Item (Item) [2]
      barCode [32-300-23456]
      department [300]
      name [Beach Towel]
      price [14.990000]
    Item (Item) [3]
      barCode [32-285-45678]
      department [285]
      name [Ginger Ale (case)]
      price [12.500000]
Coupon [1]
  description [10% off next purchase gas purchase]
  expirationDate [02/28/13]
```

Severity	Message
Info	The customer is a Preferred Cardholder
Info	10% off next gas purchase when total amount is over $75
Info	$1.649800 cashBack bonus earned today, new cashBack balance = $10.889800

We still need to reset the **cumulativeCashBack** attribute to **0**. Let's modify the rule to add the necessary logic.

Modifying the Rulesheet

Before we reset **cumulativeCashBack** to **0**, check that our preferred customer is aware of today's savings. Let's assign the value of **cumulativeCashBack** to the attribute named **savings**. Assume that this **savings** amount will be printed on a receipt provided to the customer.

Then, following this assignment, we can reset the **cumulativeCashBack** value to **0**, ready to begin accumulating new cashBack beginning with the preferred shopper's next purchase.

To write or model the rules for applying the customer's cashBack balance:

1. In the `use_cashBack.ers` Rulesheet's **Actions** section, enter line **B** as

   ```
   currentCart.savings=account.cumulativeCashBack
   ```

   Then check column **1** of this row.

2. Enter another action line,

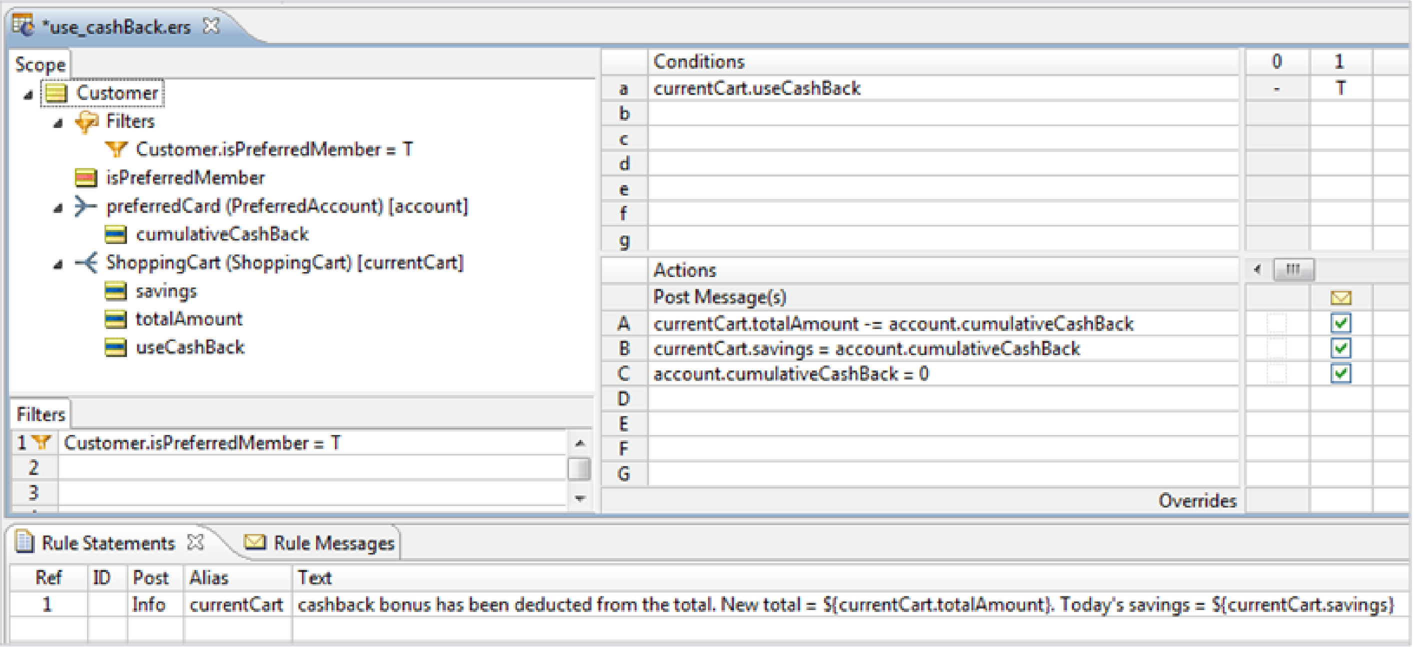
   ```
   account.cumulativeCashBack = 0
   ```

   Then check column **1** of this row.

3. In the Rule Statements panel, enter the reference `1`, then choose **Post** `Info` and **Alias** `currentCart`
4. Enter the text:

   ```
   cashback.bonus has been deducted from the total. New total =
   ${currentCart.totalAmount}. Today's savings = ${currentCart.savings}.
   ```

The rulesheet will look similar to the following:



Scope

- Customer
  - Filters
    - Customer.isPreferredMember = T
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
    - cumulativeCashBack
  - ShoppingCart (ShoppingCart) [currentCart]
    - savings
    - totalAmount
    - useCashBack

Filters

1	Customer.isPreferredMember = T
2	
3	

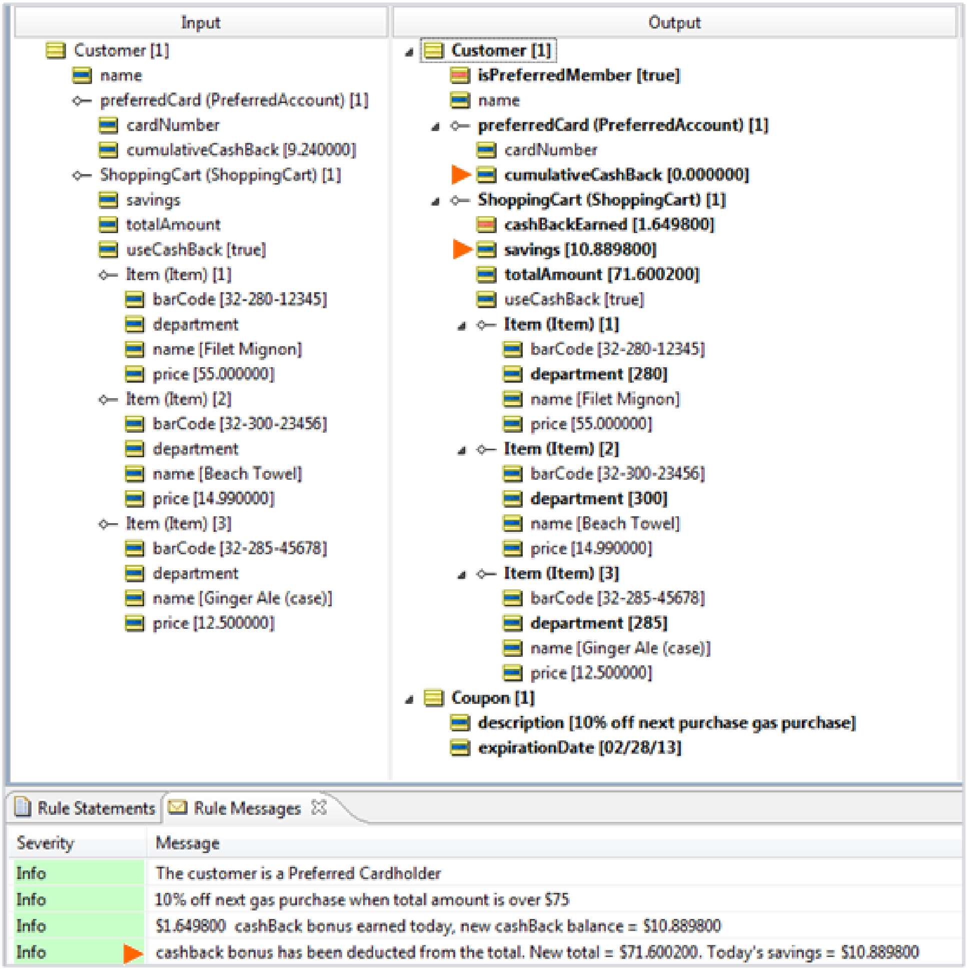
	Conditions	0	1
a	currentCart.useCashBack	-	T
b			
c			
d			
e			
f			
g			
	Actions		
	Post Message(s)		✉
A	currentCart.totalAmount -= account.cumulativeCashBack	☐	☑
B	currentCart.savings = account.cumulativeCashBack	☐	☑
C	account.cumulativeCashBack = 0	☐	☑
D			
E			
F			
G			
	Overrides		

Ref	ID	Post	Alias	Text
1		Info	currentCart	cashback bonus has been deducted from the total. New total = ${currentCart.totalAmount}. Today's savings = ${currentCart.savings}

Final condition/action rule 1 test results

Using the same Input Testsheet as in the previous test, we can see that **cumulativeCashBack** is now **0**, and **savings** has the value previously held by **cumulativeCashBack**.

We also receive the new Message describing the actions taken.



```
Input
  Customer [1]
    name
    preferredCard (PreferredAccount) [1]
      cardNumber
      cumulativeCashBack [9.240000]
    ShoppingCart (ShoppingCart) [1]
      savings
      totalAmount
      useCashBack [true]
      Item (Item) [1]
        barCode [32-280-12345]
        department
        name [Filet Mignon]
        price [55.000000]
      Item (Item) [2]
        barCode [32-300-23456]
        department
        name [Beach Towel]
        price [14.990000]
      Item (Item) [3]
        barCode [32-285-45678]
        department
        name [Ginger Ale (case)]
        price [12.500000]
```

```
Output
  Customer [1]
    isPreferredMember [true]
    name
    preferredCard (PreferredAccount) [1]
      cardNumber
      cumulativeCashBack [0.000000]
    ShoppingCart (ShoppingCart) [1]
      cashBackEarned [1.649800]
      savings [10.889800]
      totalAmount [71.600200]
      useCashBack [true]
      Item (Item) [1]
        barCode [32-280-12345]
        department [280]
        name [Filet Mignon]
        price [55.000000]
      Item (Item) [2]
        barCode [32-300-23456]
        department [300]
        name [Beach Towel]
        price [14.990000]
      Item (Item) [3]
        barCode [32-285-45678]
        department [285]
        name [Ginger Ale (case)]
        price [12.500000]
  Coupon [1]
    description [10% off next purchase gas purchase]
    expirationDate [02/28/13]
```

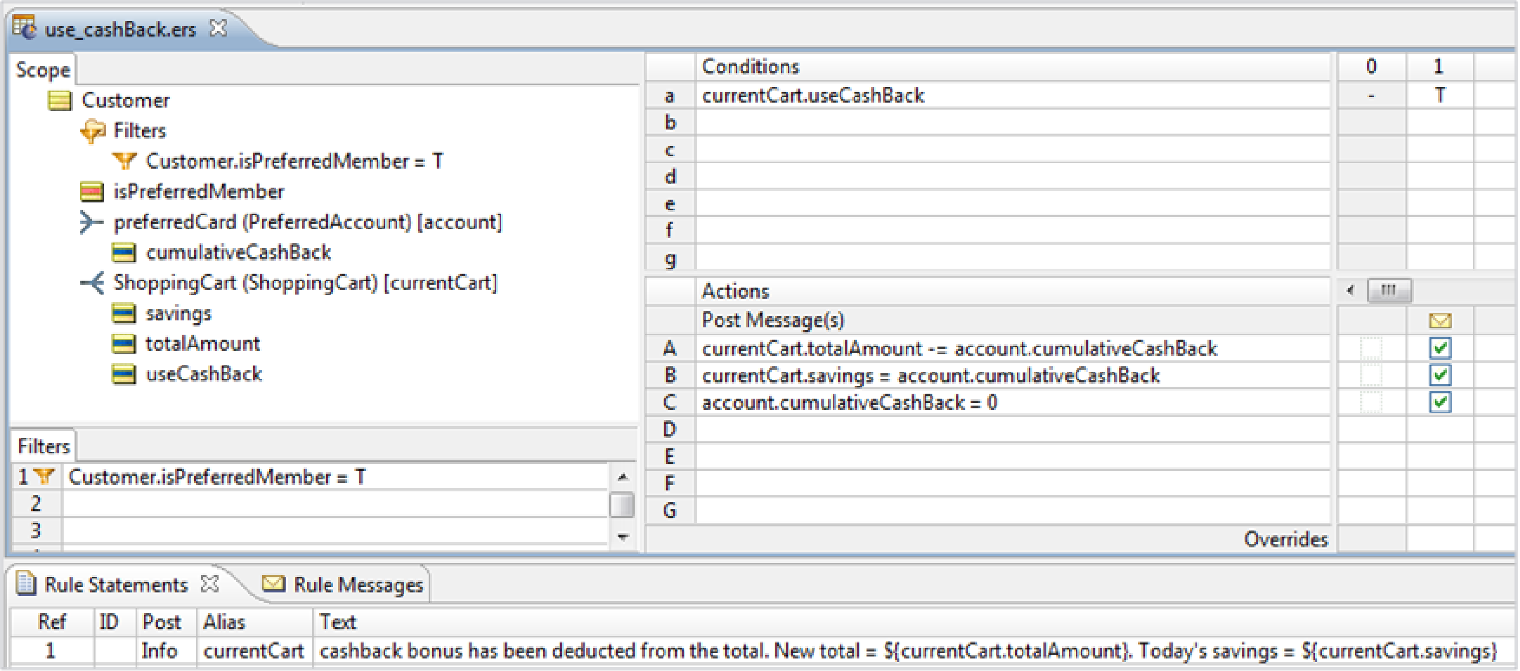
Severity	Message
Info	The customer is a Preferred Cardholder
Info	10% off next gas purchase when total amount is over $75
Info	$1.649800 cashBack bonus earned today, new cashBack balance = $10.889800
Info	cashback bonus has been deducted from the total. New total = $71.600200. Today's savings = $10.889800

The completed rule works as expected.

Since this was a cumulative test, we also can verify that the entire Ruleflow (all three Rulesheets) function as designed. The Scenario has now been fully modeled and tested.

Rulesheet three is complete and tested

The third Ruleshet is complete and tested.



Scope

- Customer
  - Filters
    - Customer.isPreferredMember = T
  - isPreferredMember
  - preferredCard (PreferredAccount) [account]
    - cumulativeCashBack
  - ShoppingCart (ShoppingCart) [currentCart]
    - savings
    - totalAmount
    - useCashBack

	Filters
1	Customer.isPreferredMember = T
2	
3	

	Conditions	0	1
a	currentCart.useCashBack	-	T
b			
c			
d			
e			
f			
g			

	Actions		
	Post Message(s)		✉
A	currentCart.totalAmount -= account.cumulativeCashBack	☐	☑
B	currentCart.savings = account.cumulativeCashBack	☐	☑
C	account.cumulativeCashBack = 0	☐	☑
D			
E			
F			
G			
	Overrides		

Ref	ID	Post	Alias	Text
1		Info	currentCart	cashback bonus has been deducted from the total. New total = ${currentCart.totalAmount}. Today's savings = ${currentCart.savings}

Perform Unit Testing

Try adding items, prices, counts and department codes that trigger the various rules. Then delete the Preferred Customer association from the Ruletest to see that the liquor test and basic cash register functions are correct while all the promotional and cashBack functions do not fire.

About Logical Validation

While these Rulesheets successfully model the Scenario's Business Rules, they are not "complete" from a logical standpoint. Studio's **Completeness Check** will reveal incomplete rules in each of the three Rulesheets.

Identifying and resolving incomplete or conflicting rules is a key part of unit testing of rule designs. The Completeness Check and the other Corticon Logical Analysis and Validation tools help you to validate your models. These topics are discussed in the *Basic Rule Modeling Tutorial* and in a focused chapter in the *Rule Modeling Guide*.

7

Summary

Congratulations on completing the Progress Corticon Advanced Rule Modeling Tutorial! You now know the mechanisms and functions of designing and testing business rules in Corticon, the best business rule modeling system available. You have learned to incorporate some powerful functionality into your rule modeling process, including:

Diagramming a Vocabulary – Based on the solution requirements identified in the Business Problem, you learned how to diagram a Vocabulary and then create a complex Corticon Vocabulary for use in rules modeling and testing.

Scope and Aliases – Scope showed you how the Corticon rules engine provides a context for associations between entities, and aliases that let you use a natural language in your rules.

Condition/Action Column 0 – The conditions and the actions based on conditions enable a Rulesheet to perform calculations and string manipulations that might be final data or in-process data for other rules in the Rulesheet, or downstream to Rulesheets in the same Ruleflow.

Filters – Filter expressions limit or reduce the data being evaluated to *only that subset* whose members satisfy the expression. A filter *does not* permanently remove or delete any data, it simply *excludes* data from evaluation by other rules in the *same* Rulesheet.

Collections and Collection Operators – You saw how Collections are a semantic grouping of a set of defined entities that are the *elements* of a collection identified by an Alias. Collection Operators analyze these groups of entities rather than individual entities.

Sequencing Rulesheets using Ruleflows – When a natural sequence or "flow" of logical steps was identified as a single decision step, you saw how it was practical to organize the flow using separate Rulesheets for each logical step. Rulesheets then execute in a sequence defined in the Ruleflow. Using multiple Rulesheets helped you assign rules to the teams that control the content as well as provide a granularity of logical construct that simplifies changes, testing, and re-use.

Extended Transient Attributes – Some attributes are "intermediate" or "temporary" value holders. These Extended Transient Attributes are used in the evaluation of logical steps in the rules, and then cleared, as they are intended for storage in a database.

Rule Statement Messaging – You have seen how Rule Statement Messaging lets you post messages of business significance (as in the ID check alert) as well as for feedback during testing. Even when Rule Statements are not posted, they are useful as documentation.

Embedding Attributes within Rule Statements – You have seen how you can easily embed attribute values in Rule Statements using a simple syntax.

A

Third party acknowledgments

One or more products in the Progress Corticon v5.3.3 release includes third party components covered by licenses that require that the following documentation notices be provided:

Progress Corticon v5.3.3 incorporates Apache Commons Discovery v0.2 from The Apache Software Foundation. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 - Copyright (c) 1999-2001 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowlegement: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

4. The names "The Jakarta Project", "Commons", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written permission of the Apache Group.

3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact dom4j-info@metastuff.com.

4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd.

5. Due credit should be given to the DOM4J Project - http://www.dom4j.org

THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,

INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Corticon v5.3.3 incorporates Jaxen v1.0. Such technology is subject to the following terms and conditions: JAXEN License - $Id: LICENSE,v 1.3 2002/04/22 11:38:45 jstrachan Exp $ - Copyright (C) 2000-2002 bob mcwhirter and James Strachan. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.

3. The name "Jaxen" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jaxen.org.

4. Products derived from this software may not be called "Jaxen", nor may "Jaxen" appear in their name, without prior written permission from the Jaxen Project Management (pm@jaxen.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the Jaxen Project (http://www.jaxen.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jaxen.org/. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE Jaxen AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Jaxen Project and was originally created by bob mcwhirter <bob@werken.com> and James Strachan <jstrachan@apache.org>. For more information on the Jaxen Project, please see <http://www.jaxen.org/>.

Progress Corticon v5.3.3 incorporates JDOM v1.0 GA. Such technology is subject to the following terms and conditions: $Id: LICENSE.txt,v 1.11 2004/02/06 09:32:57 jhunter Exp $ - Copyright (C) 2000-2004 Jason Hunter and Brett McLaughlin. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.

3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <request_AT_jdom_DOT_org>.

4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <request_AT_jdom_DOT_org>.

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)."

Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the JDOM Project and was originally created by Jason Hunter <jhunter_AT_jdom_DOT_org> and Brett McLaughlin <brett_AT_jdom_DOT_org>. For more information on the JDOM Project, please see <http://www.jdom.org/>.

Progress Corticon v5.3.3 incorporates Saxpath v1.0. Such technology is subject to the following terms and conditions: Copyright (C) 2000-2002 werken digital. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.

3. The name "SAXPath" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@saxpath.org.

4. Products derived from this software may not be called "SAXPath", nor may "SAXPath" appear in their name, without prior written permission from the SAXPath Project Management (pm@saxpath.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the SAXPath Project (http://www.saxpath.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.saxpath.org/ THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SAXPath AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the SAXPath Project and was originally created by bob mcwhirter <bob@werken.com> and James Strachan <jstrachan@apache.org>. For more information on the SAXPath Project, please see <http://www.saxpath.org/>.