

Advanced Rule Modeling

Corticon® Business Rules Modeling Studio 5.3.4

About this Guide

The **Basic Rule Modeling Tutorial** provided you with an introduction to the **Corticon Studio**, the easiest way to manage and automate your business rules. You learned how to capture rules from business specifications, model the rules, analyze them for logical errors and test the execution of your rules; all without programming.

Unlike the **Basic Rule Modeling Tutorial**, this manual does not attempt to capture or reproduce the mechanics of rule modeling. Instead, you will learn the concepts underlying some of 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.



Note

As you already know, the Ruleflows that you build using Studio may be deployed as executable, standards-based Decision Services that can be used by other software applications via Java Messaging or XML Web Services. Decision Services are in use today across the globe, automating many high-volume decision-intensive processes.

See the **Tutorial for Corticon Server – Deploying Web Services** for instructions on how to deploy and test as Decision Services the Ruleflows you build here.

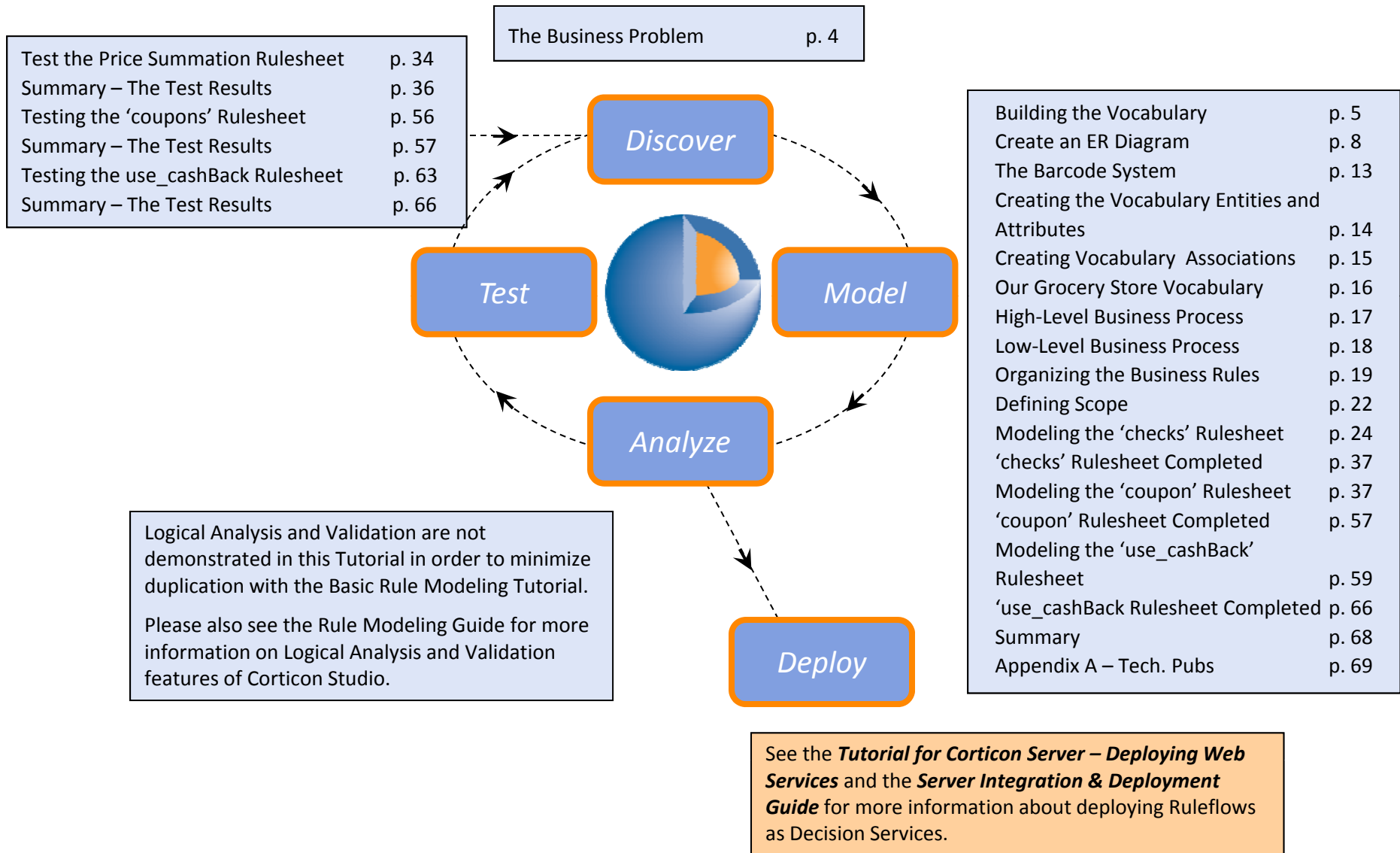


Note

This Tutorial is designed for hands-on use. We recommend that you type along with the instructions and illustrations presented.

Screenshots in this Tutorial will be cleanest and sharpest when printed using a **Postscript** printer driver (usually identified by “PS” in the printer name).

Table of Contents



The Business Problem

Discover



Scenario

The owner of a chain of grocery stores intends to build and install a system of business rule-based “smart” cash registers in all of its branches. Some branches are large supermarkets, and some are smaller “convenience” stores, which sell gasoline and other essentials.

In addition to minimum cash register functionality (adding up the prices 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 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.
- A Preferred Shopper will be eligible for special promotions and coupons as defined below:
 - 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

In compliance with local, state and federal laws, the chain needs to ensure that all purchases of liquor (any items from the Liquor department) are made by shoppers 21 or older. A simple alert or warning to the cashier will be sufficient to prompt an ID check.

Building the Vocabulary – Identifying the Terms

Model



Identifying the Terms of the Scenario

The owner of a chain of grocery stores intends to build and install a system of business rule-based “smart” cash registers in all of its branches. Some branches are large supermarkets, and some are smaller “convenience” stores, which sell gasoline and other essentials.

In addition to minimum cash register functionality (adding up the prices 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 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.
- A Preferred Shopper will be eligible for special promotions and coupons as defined below:
 - 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.

In compliance with local, state and federal laws, the chain needs to ensure that all purchases of liquor (any items from the Liquor department) are made by shoppers 21 or older. A simple alert or warning to the cashier will be sufficient to prompt an ID check.

Building the Vocabulary – Grouping the Terms

Model



Identifying the Terms of the Scenario

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 may be a Preferred Shopper and have a Preferred Shopper account that is identified by swiping 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 a family
- A Customer uses a Shopping Cart to carry items
- A Customer has a Name
- 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 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 Expiration Date
- A Coupon has an Issue Date

Building the Vocabulary – Organizing the Terms

Model



Identify the Business Terms

The terms that we will use to build a **Fact Model** or **ER Diagram** translate to the terms that we can use in our **Vocabulary**, first building our entities and then adding their attributes, including their data types and mode, then adding any associations that exist between entities in the diagram we create.



Attribute Mode

Most of these attributes use **Base** mode because their values will be sent in to the rules or sent back 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 or back to the client application. We'll discuss this more later in this tutorial.

Term	Type of Term	Data Type	Attribute Mode
Customer	Entity		
name	attribute of Customer	String	base
isPreferredMember	attribute of Customer	Boolean	extended transient
Item	Entity		
name	attribute of Item	String	base
price	attribute of Item	Decimal	base
department	attribute of Item	String	base
barCode	attribute of Item	String	base
ShoppingCart	Entity		
totalAmount	attribute of Shopping Cart	Decimal	base
cashBackEarned	attribute of Shopping Cart	Decimal	extended transient
savings	attribute of Shopping Cart	Decimal	base
useCashBack	attribute of Shopping Cart	Boolean	base
checkID	attribute of Shopping Cart	Boolean	base
PreferredAccount	Entity		
cardNumber	attribute of PreferredAccount	String	base
cumulativeCashBack	attribute of PreferredAccount	Decimal	base
Coupon	Entity		
issueDate	attribute of Coupon	Date	base
description	attribute of Coupon	String	base
expirationDate	attribute of Coupon	Date	base

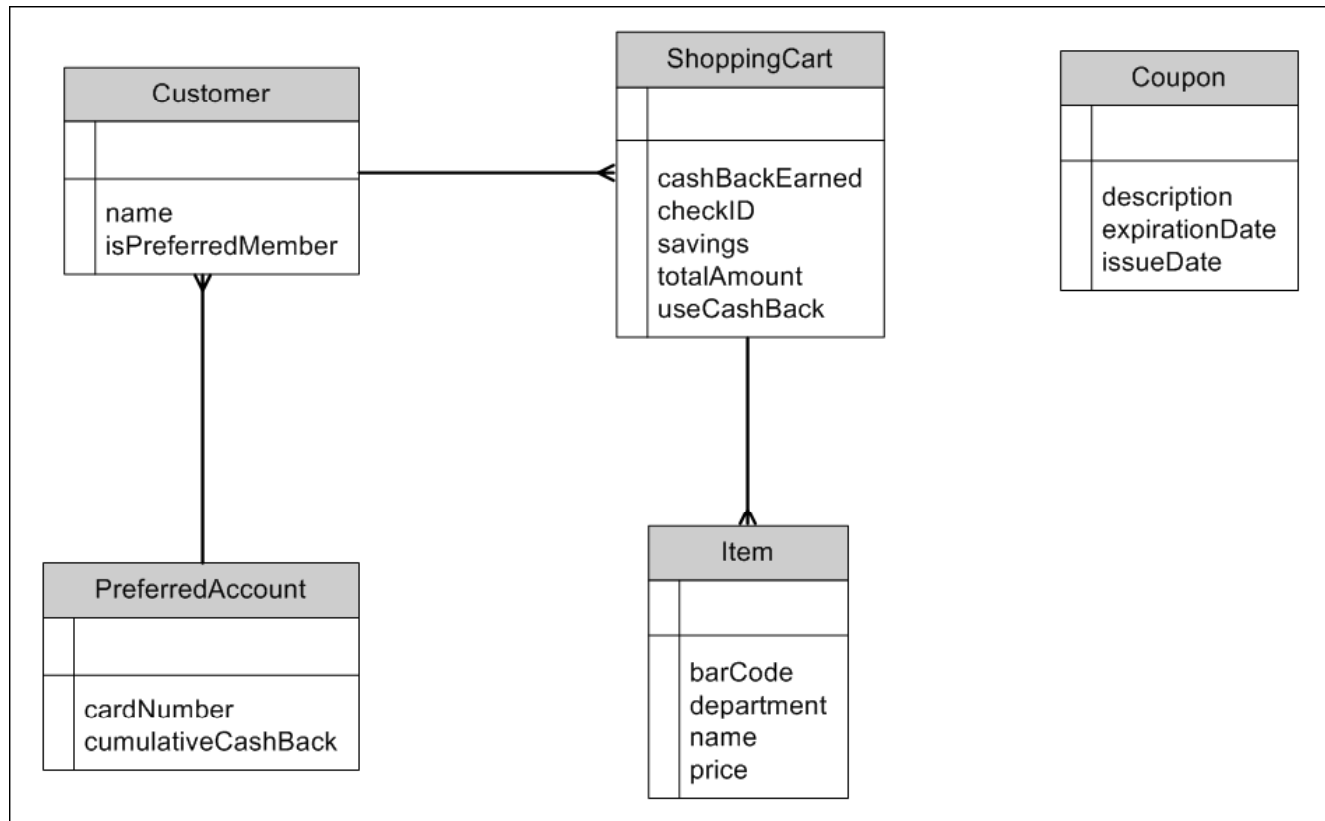
Create an ER Diagram

Model



Create an ER Diagram

The **Entity-Relationship (ER) Diagram** below is a graphical depiction of the entities and their respective attributes, as well as the associations (relationships) between entities, that we will be using in our Vocabulary during this exercise.



Define Entities

Model



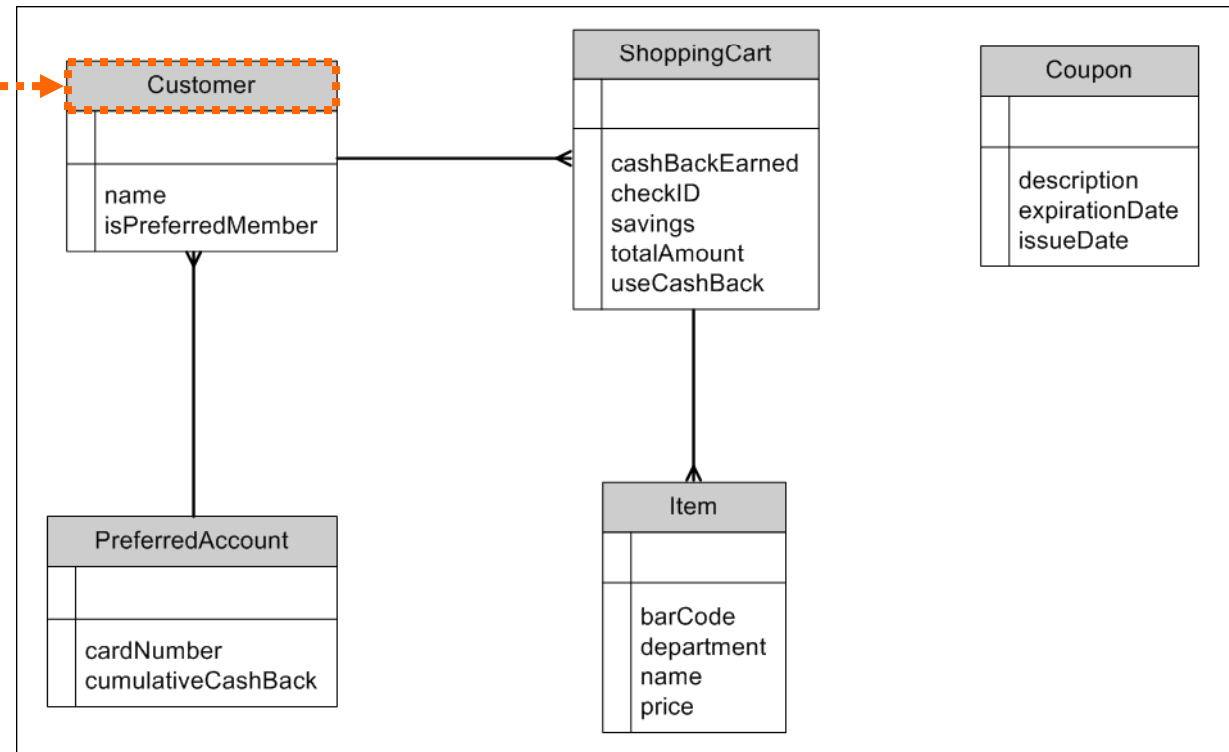
Entities

Based on our ER Diagram, we will need to build **Customer**, **Coupon**, **ShoppingCart**, **Item**, and **PreferredAccount** (our main business terms) into our Vocabulary.



Note

The Customer Entity.



Add Attributes to the Entities

Model



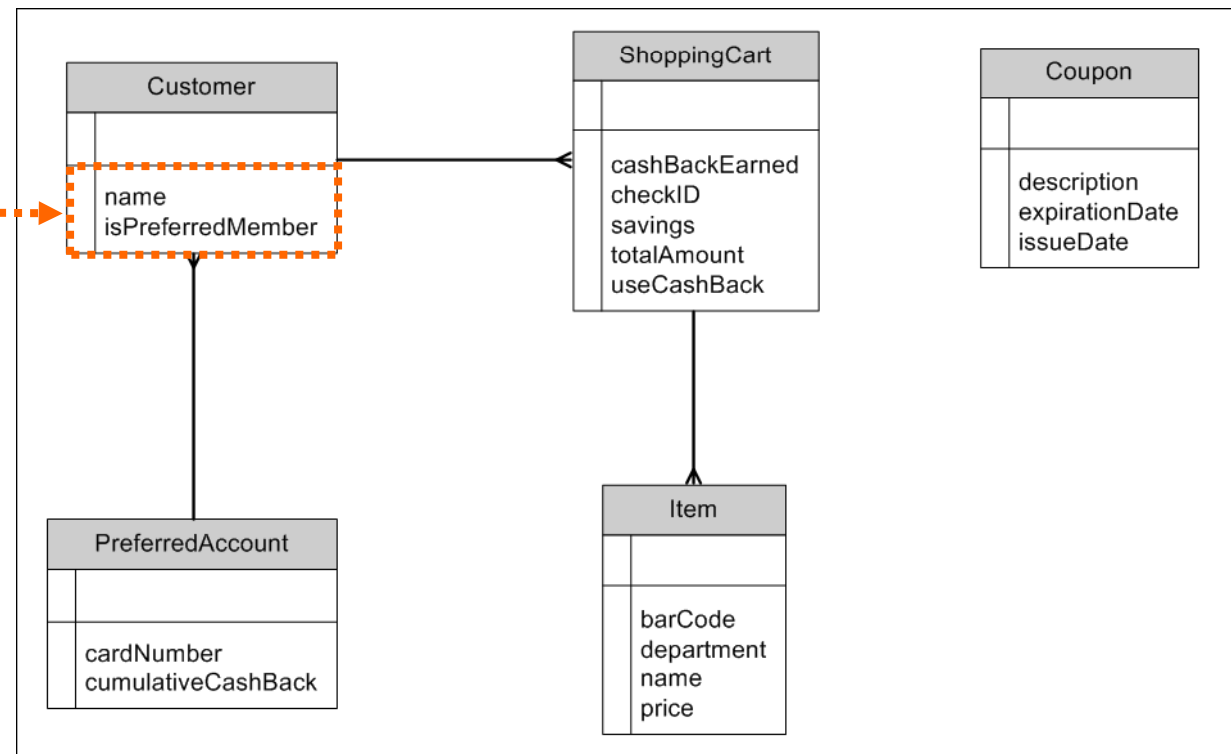
Attributes

Entities have properties or characteristics that distinguish them from other Entities, and which distinguish one instance of an Entity from another instance of the same Entity. We call these Attributes. Obviously, all customers can't be the *same* customer, so how do we distinguish between them? By defining attributes that will hold the values of each customer's name, etc...



Note

A customer's attributes hold values for each customer.



Extended Transient Attributes

Model



Extended Transient Attributes

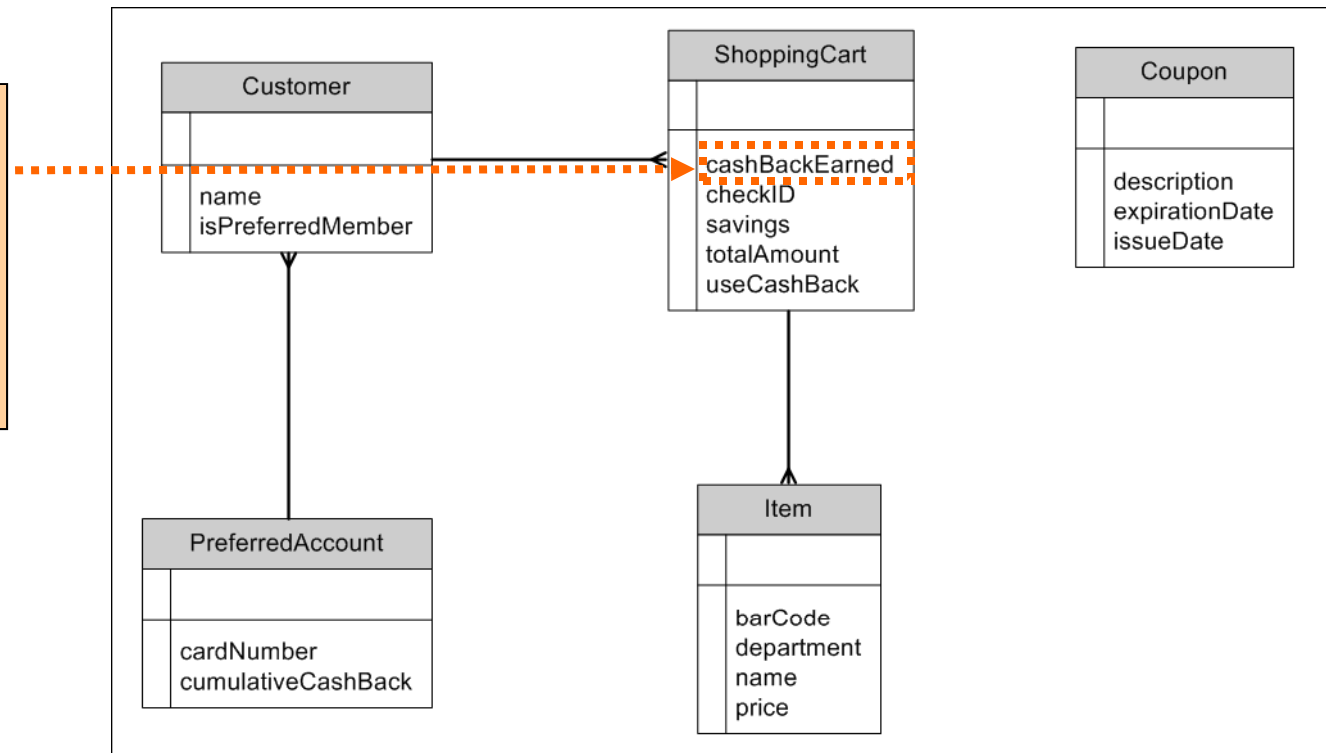
Some attributes are little more than “intermediate” or “temporary” value holders. We don’t need to return these values in a response, or save them in a database. In Studio, **Extended Transient** attributes fill this purpose. Because an extended transient is not part of the Decision Service’s response 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 fear 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).



Note

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

Model



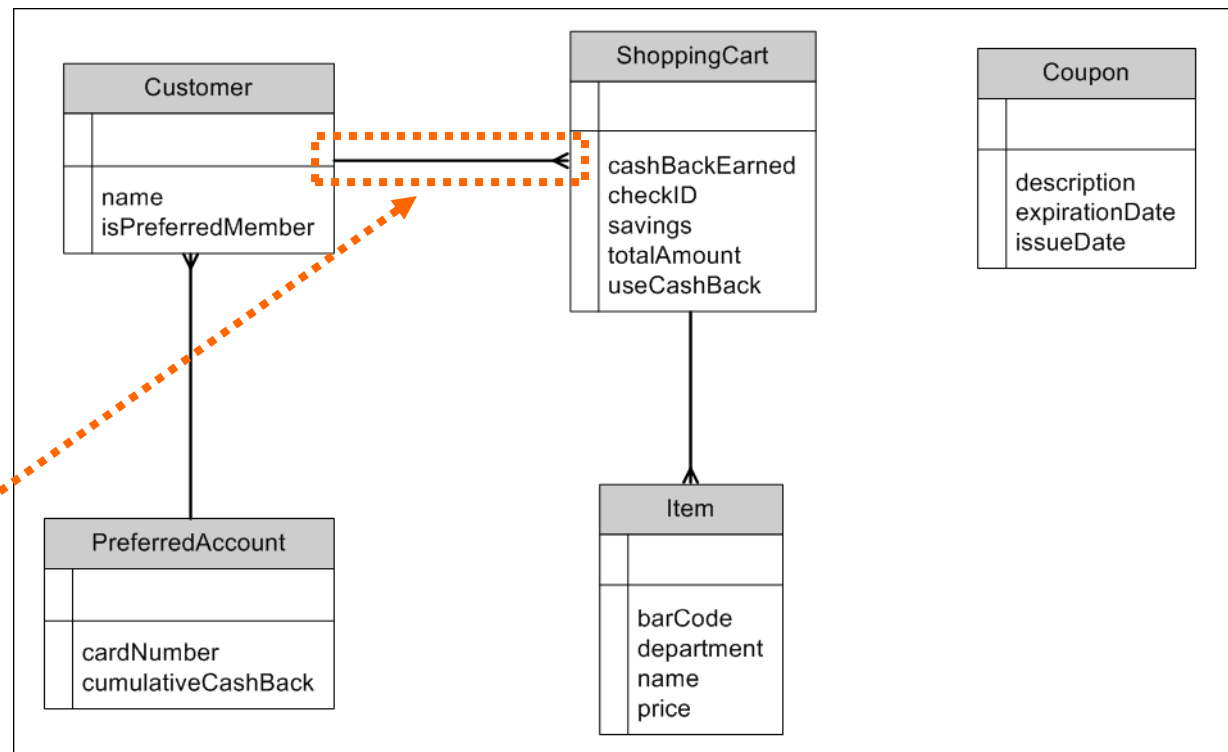
Associations

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.



Note

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



The Barcode System

Model



The Barcode System

As part of our scenario, the following tables display the **Barcode** key and the codes for various departments within a store. We will make use of the codes during our rule modeling to identify items being purchased from specific departments.

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

Corticon Licensing



We are ready to implement the Vocabulary design in Corticon Studio

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

We'll start Corticon Studio and create a new project. Then we'll create the Vocabulary for this tutorial

- Launch Corticon Studio or Corticon Studio for Analysts from the **Start** menu path **Program Files > Progress > Corticon**.
- Select the menu command **File > New > Rule Project**. Name the project MyAdvancedTutorial, and then click **Finish**.

IMPORTANT

About Corticon licensing

Corticon embeds a timed evaluation license in each Corticon Studio that lets you evaluate Corticon Studio features. Typically, you do not need to do anything to get started.

But, when you start Studio, if you see a **License Warning alert** it means that the license file is invalid, corrupted, or expired. Then, when you create or modify any Corticon files, you get an **Asset Locked Warning**, indicating that you can just review existing files.

Contact your Progress Corticon Technical Support or your Progress representative to obtain a workable license. Place the license file on your Studio machine, then launch Studio.

Choose **Window > Preferences**, then expand **Progress Corticon > Rule Modeling**. Click **Browse** and then navigate to choose your new, valid, unexpired license. When you click **OK**, and restart Studio the license update process is complete.

Creating the Vocabulary Entities and Attributes



We are ready to implement the Vocabulary design in Corticon Studio

Create the Vocabulary

Select the menu command **File > New > Rule Vocabulary**. Choose **MyAdvancedTutorial** as the parent folder, name the Vocabulary file **groceryStore**, and then click **Finish**.

Add Entities to the Vocabulary

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**.

Add Attributes to Each Entity *(see page 7 for details)*

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 each of the Customer Attributes shown on page 7.
3. Repeat these steps to add the attributes of all the other Entities shown on page 7.

All we need now is the Associations between Entities.

Creating the Vocabulary Associations



Associations in the Vocabulary

The **Customer** has associations between it and the **preferredAccount**, **ShoppingCart** and **Item** entities. The required associations are:

- many customers might be using one preferred account
- each customer might have several shopping carts
- each shopping cart might contain several items.

For detailed steps see:
Advanced Rule Modeling: Version 5.3
Organizing the terms and Creating the Vocabulary sections of the tutorial available through **online Eclipse help**

On **Customer**, add an **Association** as shown:

The 'Association' dialog box shows the following configuration:

- 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

On **Customer**, add an **Association** as shown:

The 'Association' dialog box shows the following configuration:

- 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

On **ShoppingCart**, add an **Association** as shown:

The 'Association' dialog box shows the following configuration:

- 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

Our Grocery Store Vocabulary

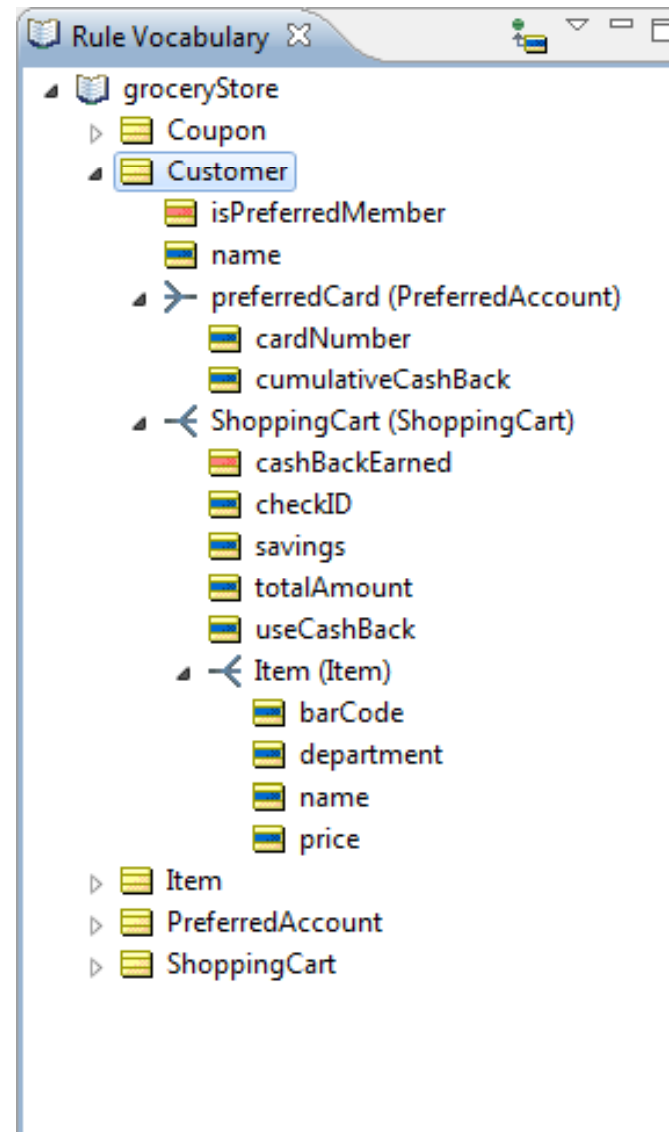
Model



Our New Vocabulary

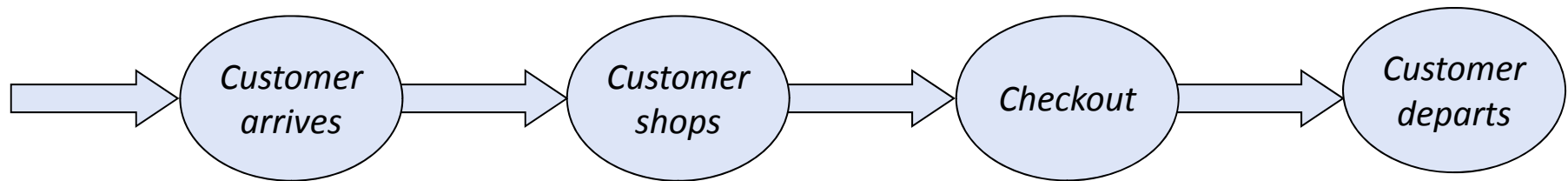
Here is the new **Vocabulary**. Notice that we are interested in working with the **Customer** root-level entity, and the associations between it and the **preferredAccount**, **Shopping Cart** and **Item** entities. Read on to find out why we are interested in this particular perspective or view of our Vocabulary

Also notice that we defined a role named **preferredCard** for the association from **Customer** to **preferredAccount**. Role names are optional but may help in describing or specifying relationships between two entities.



The High-Level Business Process

Model



Note

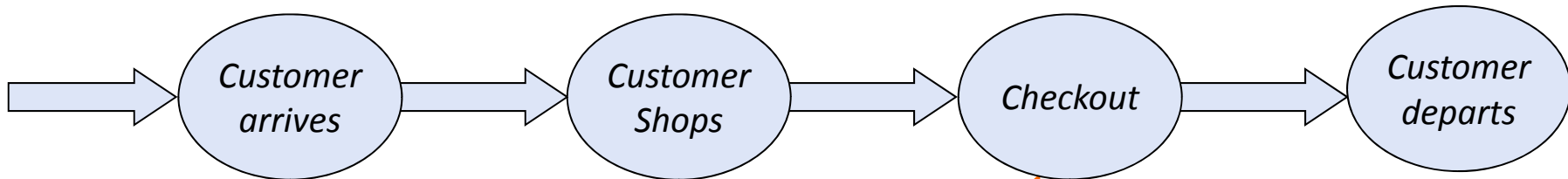
At a high level, this is the basic process followed by every customer making purchases at a store.

While there may be several steps involved in this process, we as 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.

On the next page, we'll "drill down" into the Checkout step and define more detail about the rules inside.

The Low-Level Process & Rules

Model



Note

If 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 combining them into a “Ruleflow”.

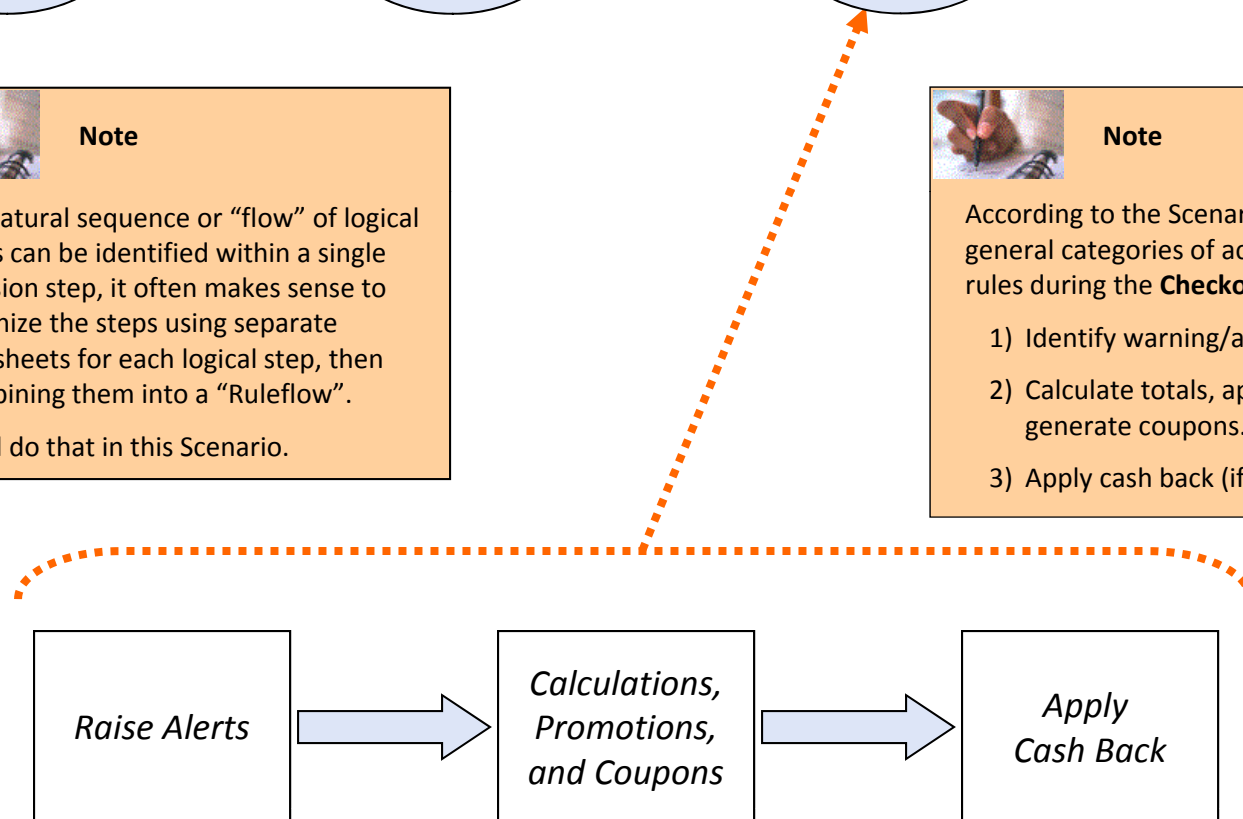
We’ll do that in this Scenario.



Note

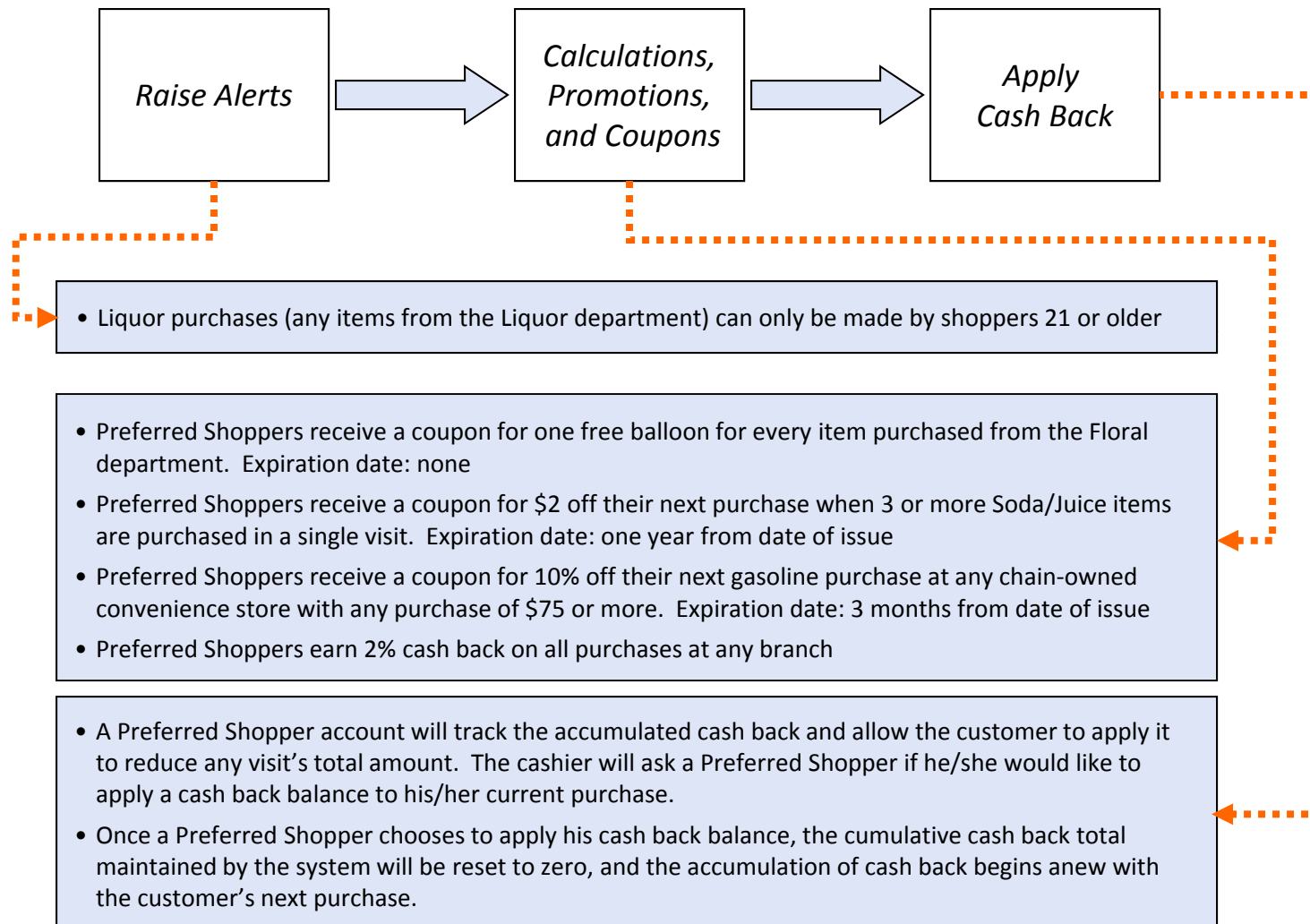
According to the Scenario, there are a few general categories of activity performed by rules during the **Checkout** step:

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



Organizing the Business Rules

Model



Preparing to Model the 'checks' Rulesheet

Model



Note

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

The 1st 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. According to the barcode chart, this means any item with numbers **291** occupying the 4th thru 6th characters of the barcode. If any are present, the cashier must be alerted to check the customer's identification.

So let's approach this Rulesheet as containing two rules: the first will determine the department code for every item in the shopping cart, and the second will determine if any of the items come from the Liquor department. If so, a rule will fire which raises an alert of some kind.

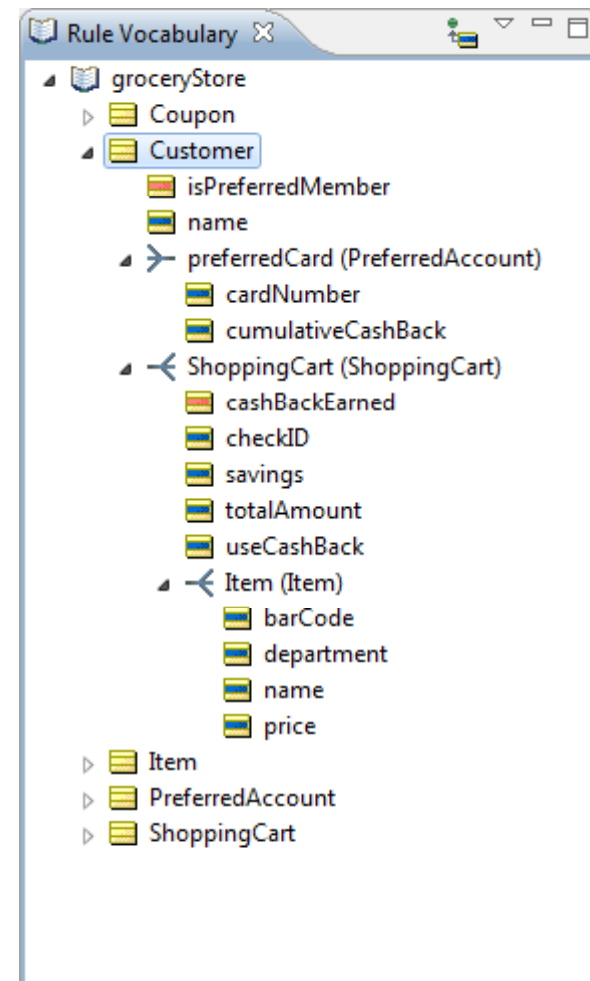


Action

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's a Customer's transaction that is processed by the checkout process step. The contents of the transaction are the **shoppingCart** and its associated **items**.



Defining Scope

Model

Action

Display the Scope section of the Rulesheet by clicking the  icon in Studio's toolbar, or select **Rulesheet>Advanced View** from Studio's menubar.

Drag and drop the **Customer** entity and then the highlighted **shoppingCart** (the one associated with Customer) into the **Scope** pane of the Rulesheet.

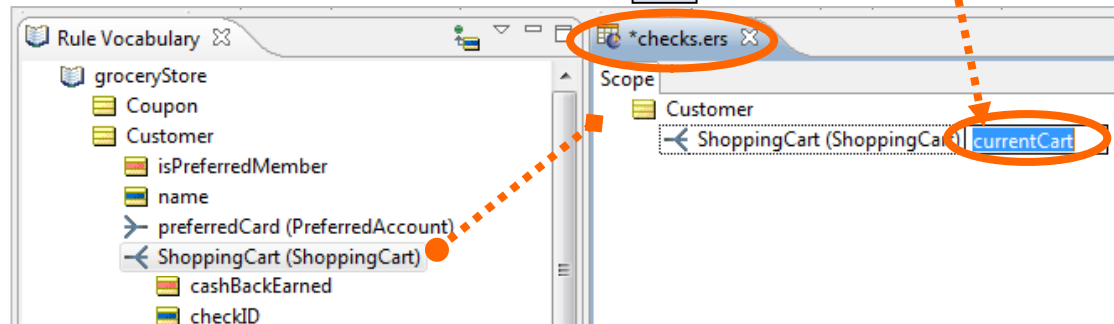
1

For detailed steps see:
**Advanced Rule Modeling:
Version 5.3 Creating the first
Rulesheet and its scope sections
of the tutorial available through
online Eclipse help.**

Action

Enter an **Alias** for this term by double-clicking it. Let's call a customer's shopping cart their **currentCart**. Henceforth, when we model rules involving a customer's shopping cart, we'll use the alias **currentCart**.

2



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 we devote an entire chapter to it in the **Rule Modeling Guide**.

Action

We've also named this Rulesheet **checks** as a way of reminding ourselves of the overall organization: this Rulesheet will perform any necessary checks and raise alerts as required.

Rulesheets can be renamed or saved to another location by selecting the **Save As** option on the **File** menu and renaming the Rulesheet and/or selecting the Project folder where you want to move it, if necessary.

3

Defining Scope

Model

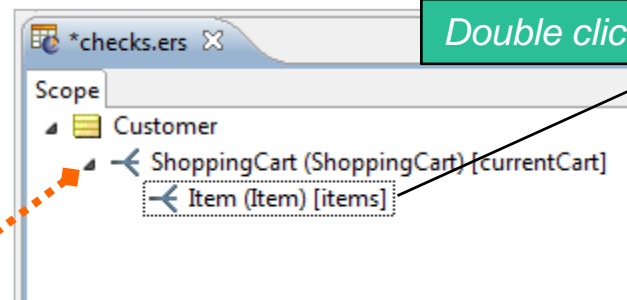
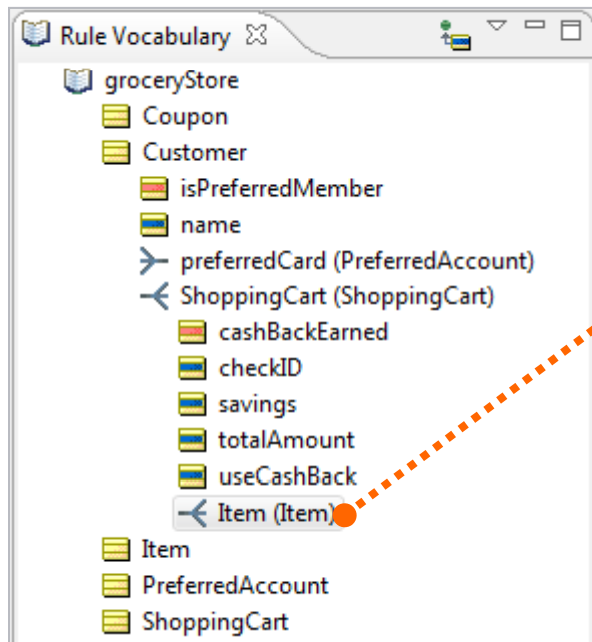


Action

We know from the **cardinality** of the association between **ShoppingCart** and **Item** that one shopping cart can contain many items. So it may prove convenient to define another alias that represents all of the items in a customer's shopping cart. We do this by dragging the associated item from the Vocabulary to the Scope window, dropping it on **shoppingCart**, and giving it an alias name by double-clicking it and typing **items** in the entry box.

Assigning meaningful alias names is good practice and using the plural form of **item** reminds us that the alias represents *all* of the items in the customer's shopping cart.

Using aliases is optional in many cases – they serve to simplify and shorten rule expressions. But in certain cases, using aliases is mandatory. Applying collection operators to sets or collections of data in rules requires the use of aliases. Since we'll be working with the collection of items in a customer's shopping cart a bit later, we must have the **items** alias defined and ready.



Double click to add an alias

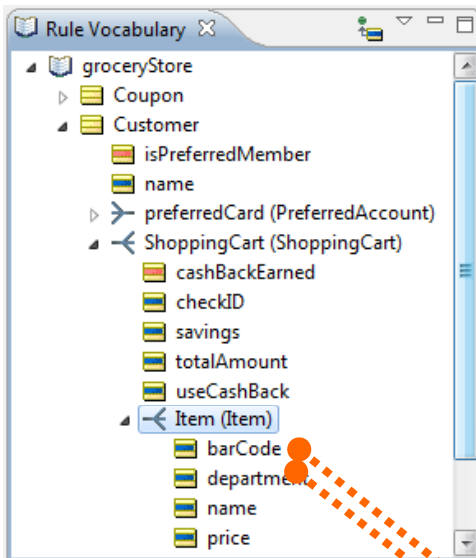


Note

Aliases will always insert themselves automatically when terms are dragged and dropped from the Scope section or Vocabulary window to the Rulesheet. Since all Studio expressions are case-sensitive, it's better to drag and drop terms instead of typing them manually – less chance of errors!

Modeling the checks Business Rule

Model



Our First Rule

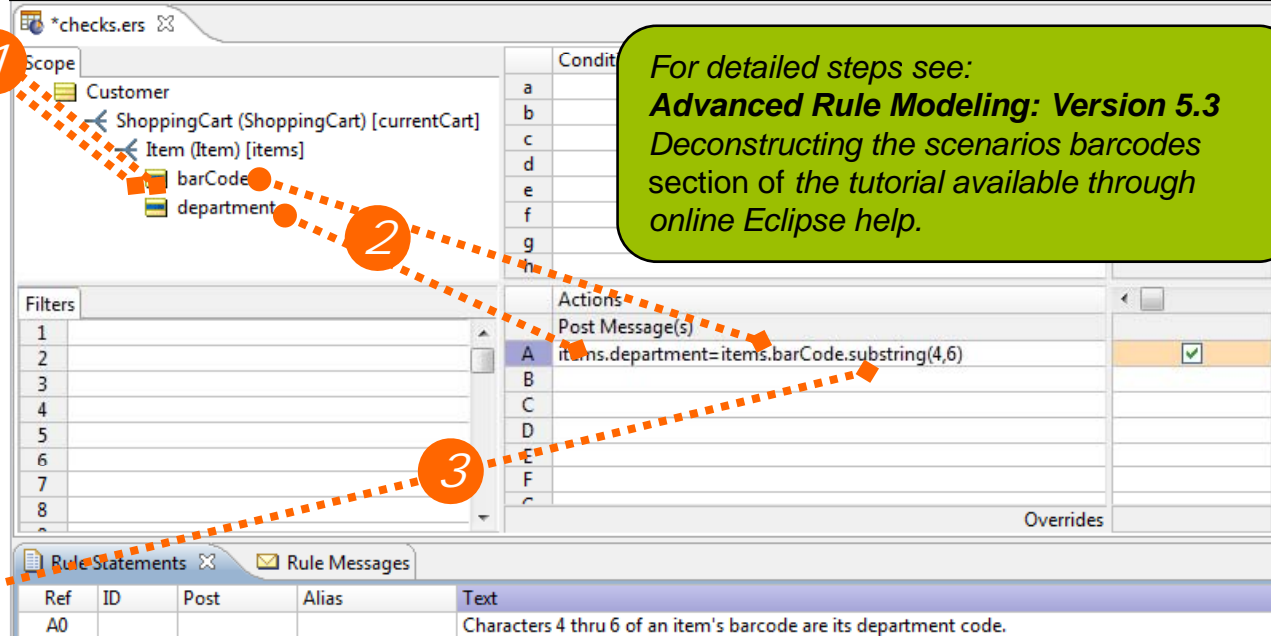
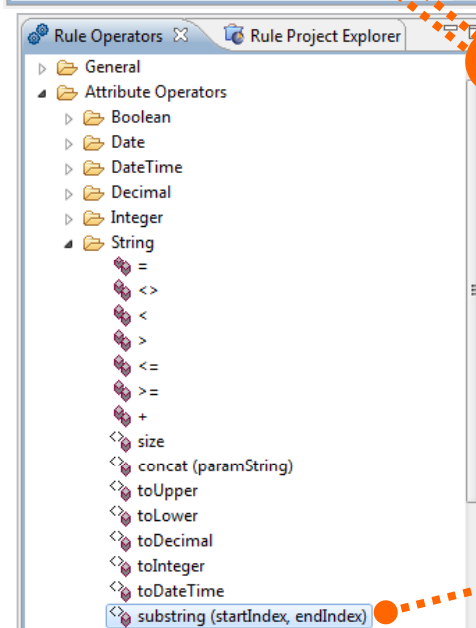
In order to model the 1st business rule, we need to be able to identify items in a customer's shopping cart that come from the Liquor department. We know an item's department is identified by the 4th thru 6th characters in the item's **barCode**.

Using the items alias, we've added a rule in an **Actions** row of Column 0 using the **.substring** operator to determine the department code for an item.

Remember that the alias **items** represents the collection of all items associated with a customer's shopping cart. So this rule will evaluate and process every 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 rule engine: it will automatically process all data that matches the rule's scope.

As terms are dragged from the Vocabulary, they are automatically added to the Scope window. Over time, the Scope window becomes a reduced version of our Vocabulary, containing only those terms used by the rules in that Rulesheet.



For detailed steps see:
Advanced Rule Modeling: Version 5.3
Deconstructing the scenarios barcodes
section of the tutorial available through
online Eclipse help.

Testing the 1st Rule

Test

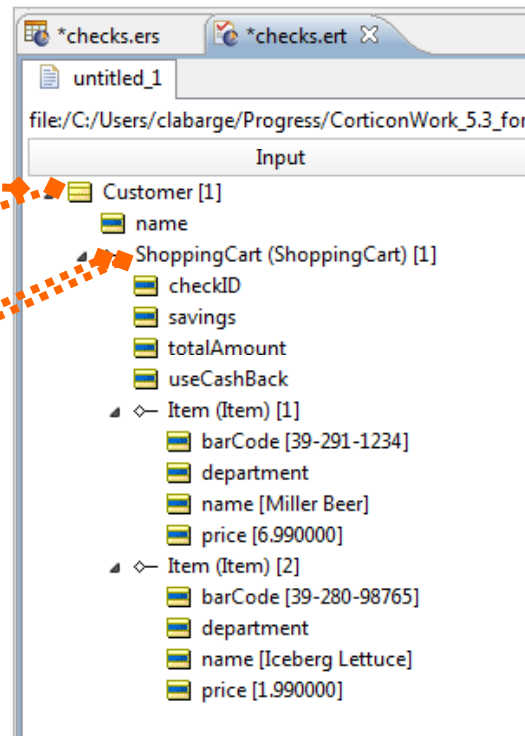
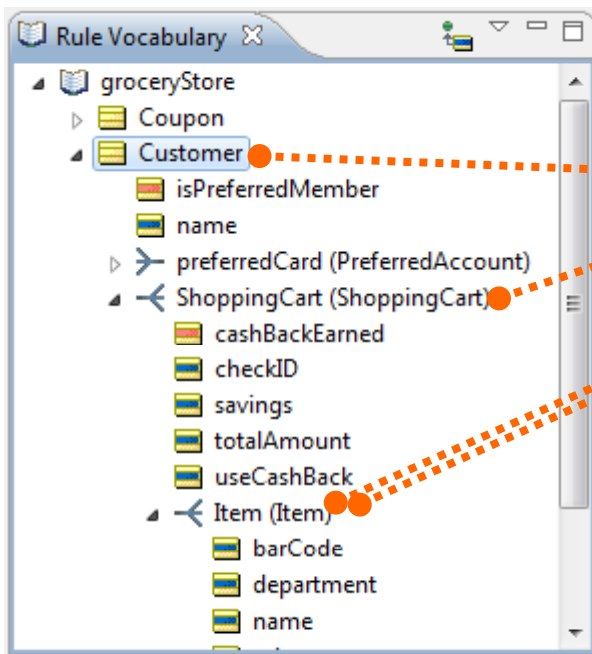
Action

Let's test our first rule. In the **Input** column of a new **Ruletest** as shown here, we have a customer with an associated shopping cart containing two items. One of them is from the Liquor department.

Logical Analysis

Ordinarily, we'd 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's 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 repeated in this guide.

For detailed steps see:
Advanced Rule Modeling: Version 5.3
Testing the first rule
section of the tutorial
available through online
Eclipse help.



Action

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.

First, drag and drop the **Customer** entity into the **Input** panel. Then, drop the **shoppingCart** entity onto the **Customer** entity. Finally, drag and drop the **item** entity onto the **shoppingCart** entity *twice*.

When finished, enter test data as shown.

Finally, execute the Ruletest.

1st Rule Test Results

Test

Test Results

Our first rule has worked as expected!
Characters 4-6 have been successfully parsed from each item's **barCode** and assigned to its **department** attribute.

Testing as you go

By modeling a rule and then immediately testing it, we've demonstrated good Studio modeling practice.

Testing right away will help expose flaws in our rules before we build too many.

Input	Output
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">nameShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkIDsavingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]departmentname [Iceberg Lettuce]price [1.990000]	<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">nameShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkIDsavingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]department [291]name [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]department [280]name [Iceberg Lettuce]price [1.990000]

Modeling the 1st Business Rule – Continued

Model



Our 1st Business Rule - Continued

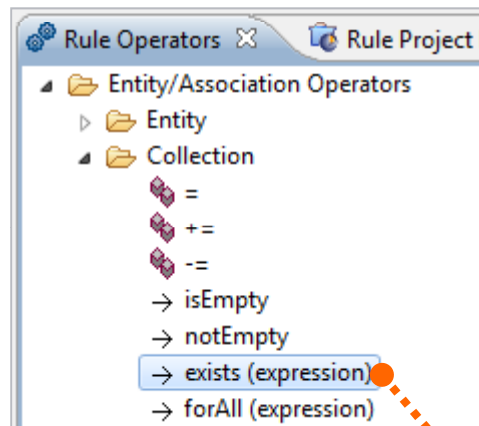
Now that department codes are readily available for every item in a customer's shopping cart, we need to determine if any came from the Liquor department. This type of question requires us to “look inside” our collection of **items** and see if there exists an item with **department = '291'** (always use plain single-quote marks to specify a text string.). Since we only need one “**check ID**” alert per checkout transaction, this is a job for a collection operator.

A collection operator, because it “acts on” collections, will evaluate once per collection and not once per item as the previous rule did. In other words, we want one “**check ID**” alert if the shopping cart contains any liquor. But we don't need, say, 5 alerts if the shopping cart contains 5 liquor items. Once is enough.

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.

For detailed steps see:

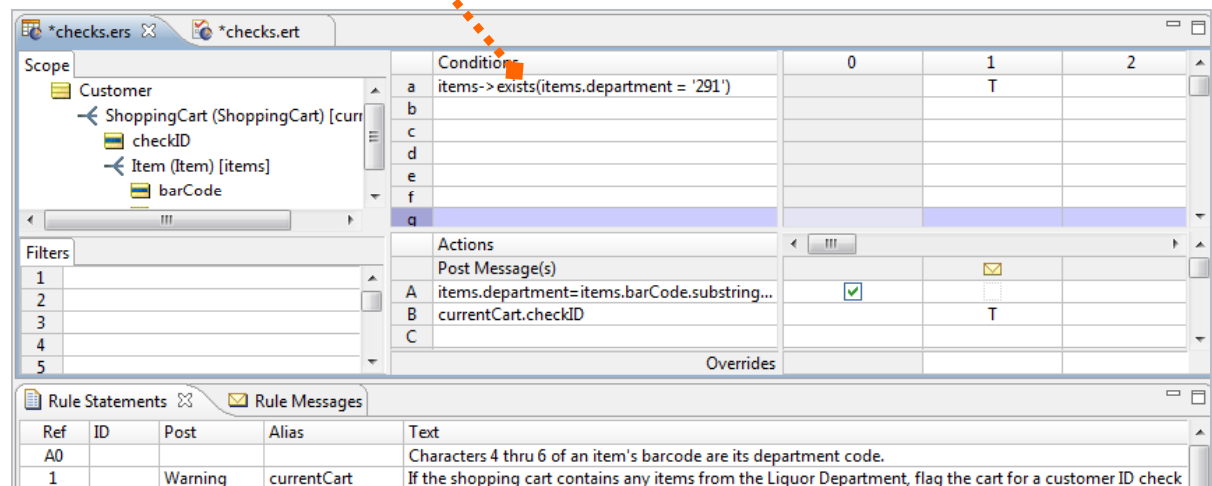
Advanced Rule Modeling: Version 5.3 Using the collections operator section of the tutorial available through online Eclipse help to create the collection.



Aliases with Collections

Using aliases to represent collections is **mandatory** when collection operators (like **→exists**) are used.

Much more information on collections and collection operators is contained in the **Rule Modeling Guide** and the **Rule Language Guide**.



1st Business Rule Test Results – Continued

Test



Testing the Complete 1st Business Rule

Re-running the same Ruletest as before, we see that our Condition/Action rule has worked as expected!

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.

Notice that the business rule statement has also been posted in the **Message Box**. Often, a simple message is all we need to raise an alert or warning.

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

Input	Output
Customer [1] <ul style="list-style-type: none">nameShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkIDsavingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]departmentname [Iceberg Lettuce]price [1.990000]	Customer [1] <ul style="list-style-type: none">nameShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkID [true]savingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]department [291]name [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]department [280]name [Iceberg Lettuce]price [1.990000]

Rule Statements | Rule Messages

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]



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

Model

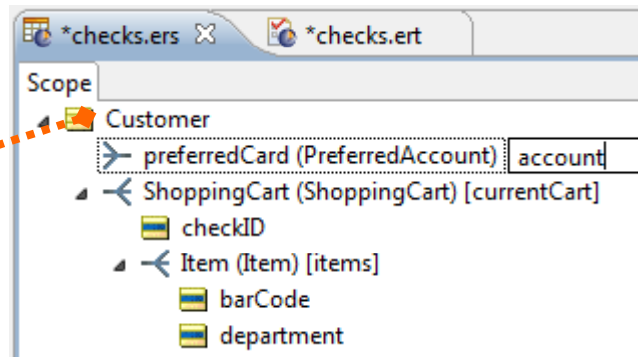
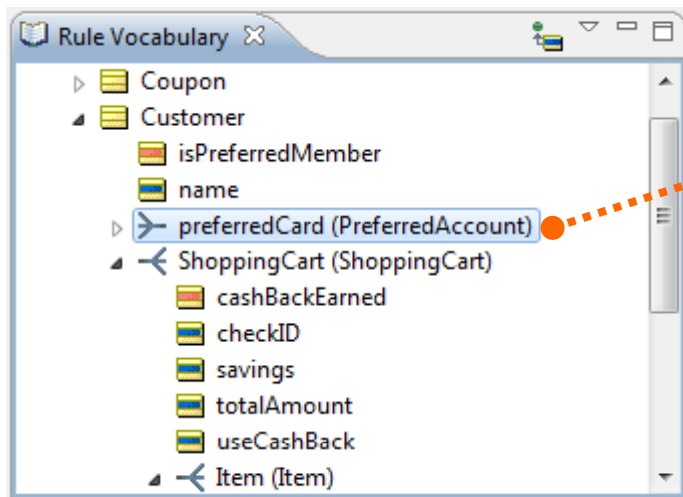


Action

Next, let's add an alias to represent a customer's **Preferred Account**. Not all customers will have **Preferred Accounts**, but those who do will have an associated **preferredCard**.

Remember from our initial scenario, 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.



For detailed steps see:
Advanced Rule Modeling: Version 5.3 Adding to scope section of the tutorial available through online Eclipse help to add preferred customer test to the rulesheet.



Note

The **account** 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) may share the same **Preferred Account**.

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

Modeling another Condition/Action Rule

Model

Flagging our Data

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 her **isPreferredMember** attribute. This method of “flagging” an entity with a boolean attribute (also known as a “flag”) 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.

Action

Now we’ll add a second Condition/Action rule that checks if the customer has a Preferred Card account. We’ve modeled a 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.

For detailed steps see:
**Advanced Rule Modeling:
Version 5.3 Modeling another
condition/action rule section of
the tutorial available through
online Eclipse help to add a rule
that checks if the customer has
a Preferred Card account.**

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

Filters	Actions	0	1	2
1	Post Message(s)			
2	A items.department=items.barCode.substring(4,6)	✓		
3	B currentCart.checkID		T	
4	C Customer.isPreferredMember			T
5				
6				
7				

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

Testing the 2nd Condition/Action Rule

Test

Action

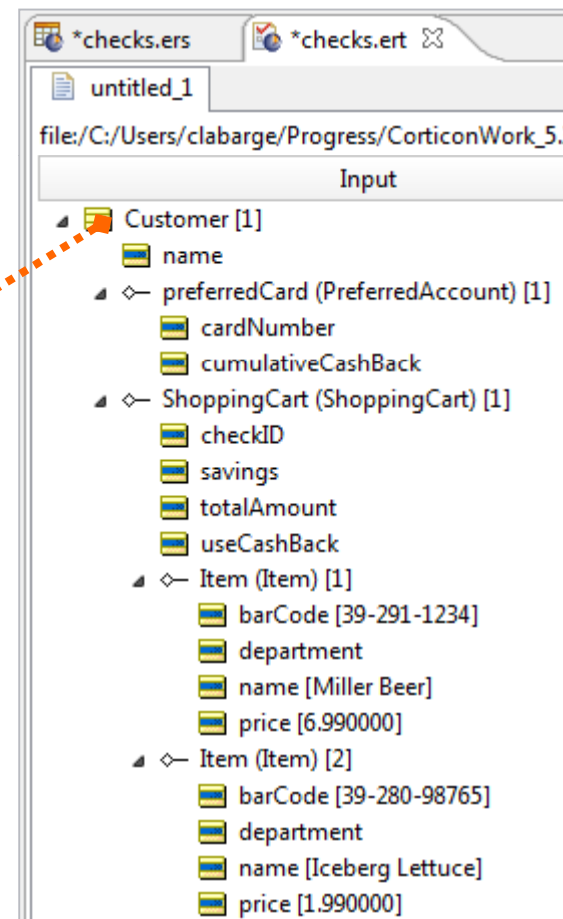
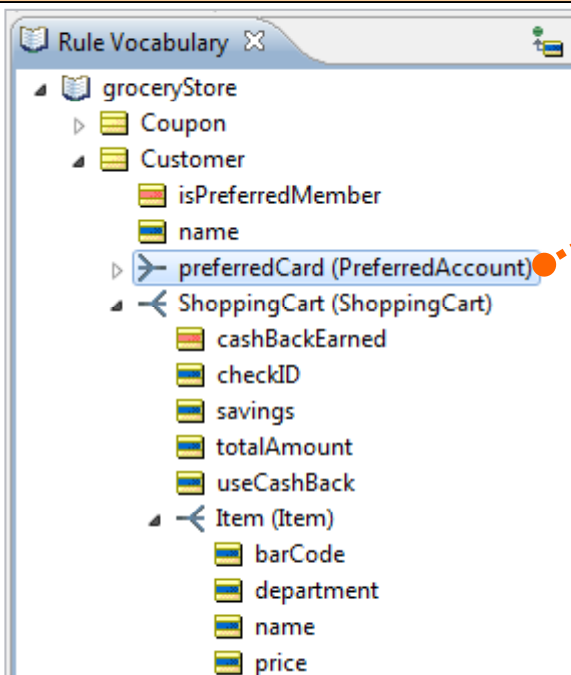
Now let's test our second Condition/Action rule. For our rule to detect the presence of a **Preferred Card** account associated with this customer, we need to provide the appropriate test data.

Drag and drop the **preferredCard** entity onto the **Customer** entity in the **Ruletest Input**, as shown to the right. If you don't get the identical indented structure as shown, delete the entity and try again.

Now execute the Ruletest.

For detailed steps see:

Advanced Rule Modeling: Version 5.3 Testing the second condition/action rule section of the tutorial available through online Eclipse help to test the second condition/action rule.



2nd Condition/Action Rule Test Results

Test

Results

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!

Input

Output

Customer [1]

- isPreferredMember [true]
- name
- preferredCard (PreferredAccount) [1]
 - cardNumber
 - cumulativeCashBack
- ShoppingCart (ShoppingCart) [1]
 - checkID [true]
 - 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]

Rule Statements

Rule Messages

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]

Modeling the Price Summation Rule


Model

Action

Finally, we'll add one more **Action** to Column 0 that will calculate the **totalAmount** of all items in a customer's shopping cart. 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.

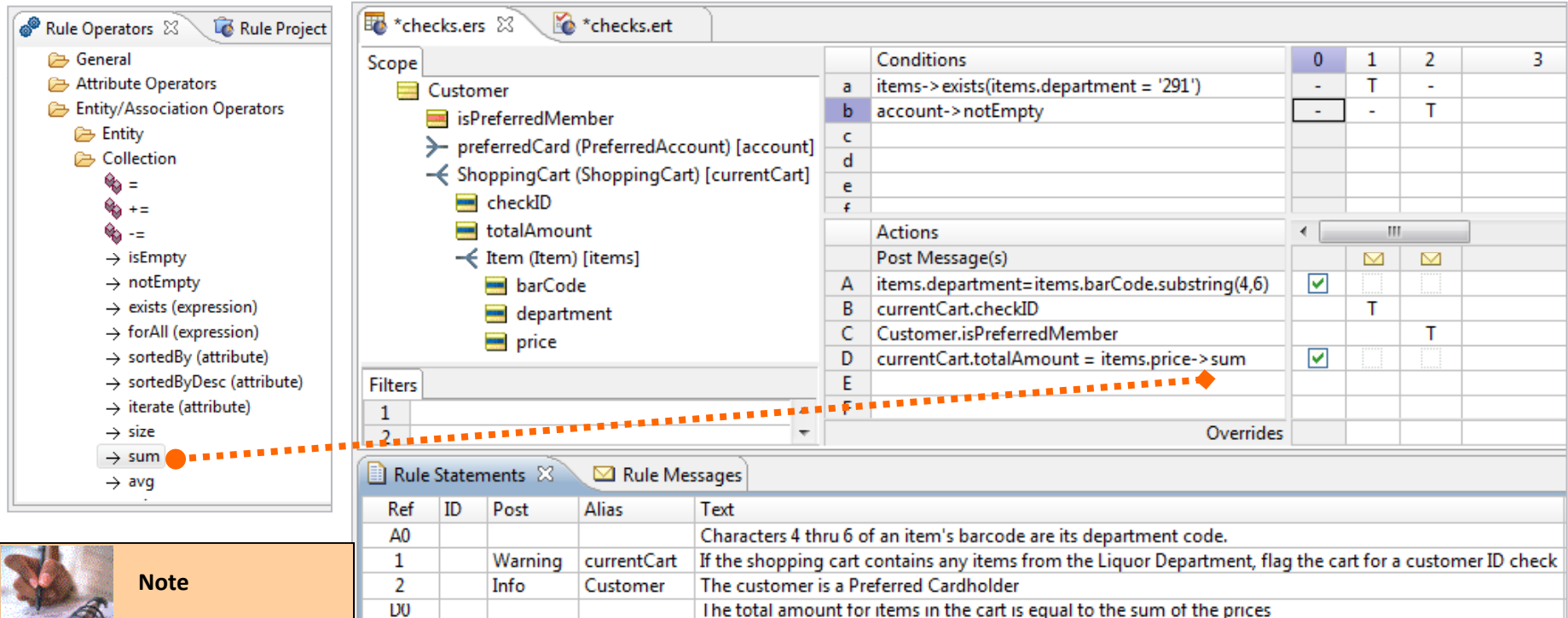
For detailed steps see:

Advanced Rule Modeling: Version 5.3 Modeling the price summation rule section of the tutorial available through online Eclipse help to calculate the total price of the items in the current shopping cart.



Note

Adding **Rule Statements** is good practice, even if you don't post them as messages.



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
UU				The total amount for items in the cart is equal to the sum of the prices

Testing the Price Summation Rule

Test



Action

Let's test our third, and final, rule on this Rulesheet. In the Input Testsheet shown here, we have a customer with two items in his shopping cart. Does our third rule provide us with the **totalAmount** for the items in the Customer's shopping cart?

The screenshot shows a software window with two tabs: '*checks.ers' and '*checks.ert'. The active tab is '*checks.ers', which contains a document titled 'untitled_1'. The file path is 'file:/C:/Users/clabarge/Progress/CorticonWork_5.3_for_Analysts'. The main area is titled 'Input' and shows a tree structure of data. The tree is expanded to show the 'totalAmount' field, which is highlighted. The tree structure is as follows:

- 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]



Action

Let's run our test and see what happens...

Price Summation Test Results

Test

Results

A lot has happened here...

First, note that our rules to determine if a) an ID check is required and b) if the customer is a **Preferred Card** holder still work as before. It's always good to double-check cumulative test results to make sure nothing has broken along the way.

Also, notice that the **totalAmount** attribute has returned a value of **8.98**, which is the correct sum of the prices of items 1 and 2.

Our third rule has also worked as we expected!

untitled_1

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

Input	Output
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBackShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkIDsavingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]departmentname [Iceberg Lettuce]price [1.990000]	<ul style="list-style-type: none">isPreferredMember [true]namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBackShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">checkID [true]savingstotalAmount [8.980000]useCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]department [291]name [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-280-98765]department [280]name [Iceberg Lettuce]price [1.990000]

Rule Statements Rule Messages

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

Test



Summary

An item has been purchased from department **291**, the Liquor department (identified by the **barCode**). A **checkID** alert is issued and a warning message is 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.

Finally, the **totalAmount** shows that the two item prices have been added together correctly.

So the rules we've built on this Rulesheet have all functioned as we expected them to!

- name [Miller Beer]
- price [6.990000]
- Item (Item) [2]
 - barCode [39-280-98765]
 - department
 - name [Iceberg Lettuce]
 - price [1.990000]

Work_5.3_for_Analysts/workspace/MyAdvancedTutorial/checks.ers

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]

Rule Statements Rule Messages

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]

'checks' Rulesheet Completed

Model



'checks' Rulesheet completed

Here's our first completed Rulesheet!

*checks.ers ✕ *checks.ert

Scope

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

Filters

1	
2	

Conditions

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

Actions

Post Message(s)

		0	1	2	3
A	items.department=items.barCode.substring(4,6)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
B	currentCart.checkID		T		
C	Customer.isPreferredMember			T	
D	currentCart.totalAmount = items.price-> sum	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
E					
F					
G					
H					
I					

Overrides

Rule Statements ✕ Rule Messages

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

Model the 'coupon' Rulesheet

Model

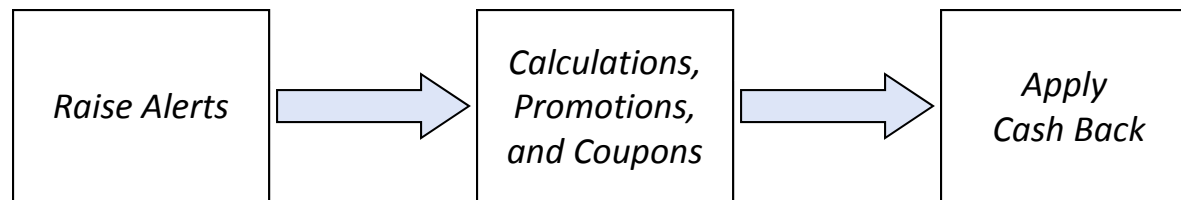


Action

Now it's time to apply some promotions to our **Preferred Account** holders when they spend a pre-defined amount of money or buy items from specific departments at our store. According to the Scenario's Business Rules, these promotions vary, but include discounts, rebates, or even free gifts when items are purchased in specific amounts or from specific departments. The promotions will change frequently – modeling them in Corticon will make future changes much easier.

Let's create a second Rulesheet (we'll call it **coupons**) and model our rules to reflect these promotions for our Preferred **Account** holder customers.

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 **Studio Quick Reference Guide**.



- 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

Scope Revisited

Model

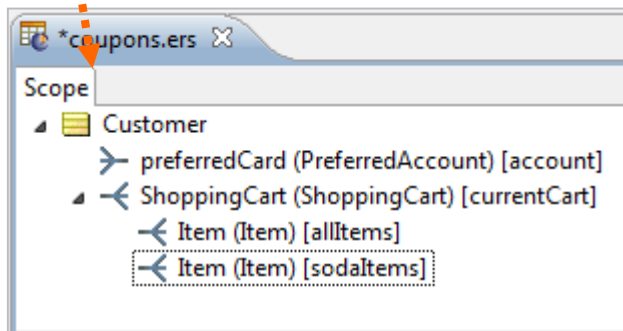
Scope Revisited

Create the **Scope** for the new coupons Rulesheet as shown below.

First, a customer's shopping cart is still assigned an alias of **currentCart**, just like on the **checks** Rulesheet. But on the coupons Rulesheet we have created two new aliases to define the **currentCart.item** perspective of our data. For now, we'll simply define the two aliases **allItems** and **sodaItems** to represent the same perspective, but we'll differentiate between them shortly.

As before, the **account** alias still represents the **preferredCard** account associated with our customer.

For detailed steps see:
**Advanced Rule Modeling:
Version 5.3 Scope revisited** section of the tutorial available through online Eclipse help to create the new Rulesheet's scope.

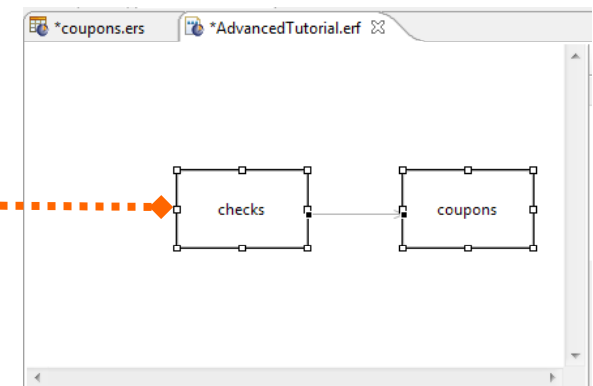
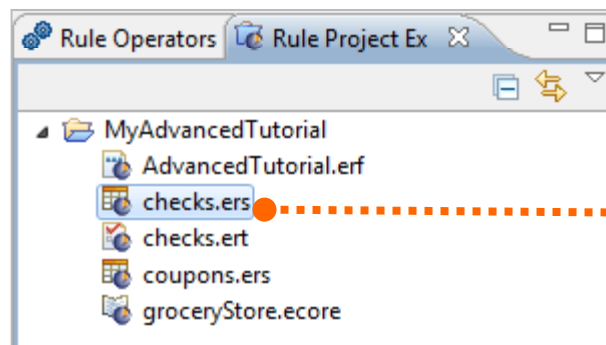


Note

When creating Ruletests that need to process multiple Rulesheets in sequence, be sure to choose your **Ruleflow** as the **Test Subject** during the **Ruletest Creation Wizard** process. That will ensure that all Rulesheets are processed in the correct sequence and allow values derived on prior Rulesheets to be used in subsequent Rulesheets.

Ruleflow

The Rulesheet processing sequence is visible here in the **Ruleflow** diagram at the right. For a complete discussion of the Ruleflow Editor's purpose and functionality, please refer to the **Quick Reference Guide**.



Filter Expressions

Model



Filters

A **Filter** expression acts to 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.

We often say that data satisfying the **Filter** expression “survives” the filter. Data that does not satisfy the expression is said to be “filtered out.” Data that has been filtered out is ignored by other rules in the same Rulesheet.

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

For detailed steps see:

Advanced Rule Modeling: Version 5.3
Filter expressions section of the tutorial available through online Eclipse help to create the filter shown here

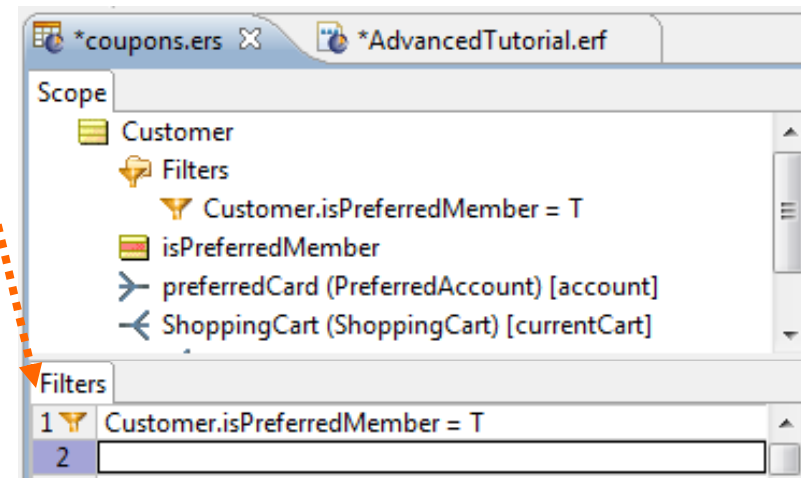


Filters

Customers who are not **Preferred Card** holders are not eligible for the promotions defined in the original business rules. So we want to exclude non-preferred customers from evaluation by the Rulesheet.

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

Those customers whose **isPreferredMember** attribute value is not true are filtered out and not evaluated by other rules on this Rulesheet.



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.

Testing CashBackEarned Calculation

Test

Test the Rule

Here's a simple test for **Action** row A in column 0. We've added a few items to the **shoppingCart** and entered prices for each of them. According to the rule we are testing, the shopping cart of a preferred cardholder should earn cash back equal to **2%** of the **totalAmount** in the shopping cart.

*For detailed steps see:
Advanced Rule Modeling: Version 5.3
Testing
cashBackEarned
calculation section of
the tutorial available
through online Eclipse
help to use the ruleflow
for testing.*

untitled_1

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

Input	Output	Expected
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBackShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">savingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-290-12345]departmentname [Tulips - 1 dozen]price [30.000000]Item (Item) [3]<ul style="list-style-type: none">barCode [39-285-12345]departmentname [Tomato Juice (case)]price [62.000000]		


Use the ruleflow created earlier

Action

Let's run the Ruletest and see what happens...

CashBackEarned Calculation Test Results

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!

Input	Output
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBackShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">savingstotalAmountuseCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-290-12345]departmentname [Tulips - 1 dozen]price [30.000000]Item (Item) [3]<ul style="list-style-type: none">barCode [39-285-12345]departmentname [Tomato Juice (case)]price [62.000000]	<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">isPreferredMember [true]namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBackShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">cashBackEarned [1.979800]checkID [true]savingstotalAmount [98.990000]useCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]department [291]name [Miller Beer]price [6.990000]Item (Item) [2]<ul style="list-style-type: none">barCode [39-290-12345]department [290]name [Tulips - 1 dozen]price [30.000000]Item (Item) [3]<ul style="list-style-type: none">barCode [39-285-12345]department [285]name [Tomato Juice (case)]price [62.000000]

Modeling Cumulative CashBackEarned

Model

2nd Action in Column 0

Action row B in Column 0 calculates the **cumulativeCashBack** amount in a customer's account by incrementing its value (using +=) by the **cashBackEarned** in the current shopping cart.

We'll also want to post a message displaying that amount. The choices are made from the drop-down menus in the **Post** and **Alias** columns.

For detailed steps see:
Advanced Rule Modeling: Version 5.3
Modeling cumulative cashBackEarned section of the tutorial available through online Eclipse help to add the current cashBack amount to the cumulative amount.

Scope

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

Filters

Ref	ID	Post	Alias	Text
1				Customer.isPreferredMember = T
2				
3				
4				
5				

Conditions

Ref	ID	Post	Alias	Text
a				
b				
c				
d				
e				
f				
g				
h				
i				
j				
k				

Actions

Ref	ID	Post	Alias	Text
				Post Message(s)
A				currentCart.cashBackEarned = currentCart.totalAmount * 0.02
B				account.cumulativeCashBack += currentCart.cashBackEarned
C				

Overrides

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}

Rule Operators

- Date
- DateTime
- Decimal
 - =
 - <>
 - <
 - >
 - <=
 - >=
 - +
 -
 - *
 - /
 - **
 - +=

Testing the Cumulative CashBackEarned Model

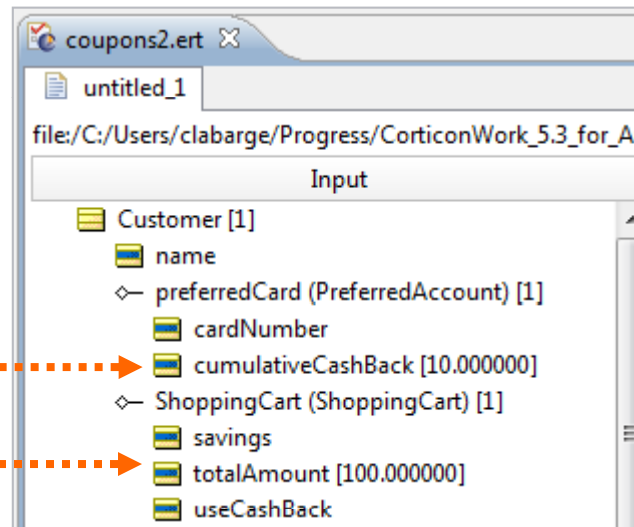
Test



Test the Rule Model

For this test, we've entered a **totalAmount** of **\$100** for the **shoppingCart** and a **cumulativeCashBack** amount of **\$10**.

We've already tested this Ruleflow's ability to sum up the prices of each individual item to calculate a **totalAmount**, so we won't test that Rulesheet now.



Test the Rule

When building Ruletests, it's easy to forget that a Rulesheet's **Filters**, if not satisfied, *may 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

Test



Test Results

As you can see in the Results Testsheet below, 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

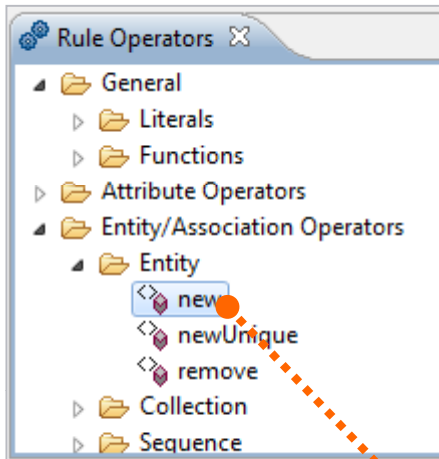
untitled_1

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

Input	Output
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBack [10.000000]ShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">savingstotalAmount [100.000000]useCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]departmentname [Miller Beer]price [8.000000]	<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">isPreferredMember [true]namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBack [12.000000]ShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">cashBackEarned [2.000000]checkID [true]savingstotalAmount [100.000000]useCashBackItem (Item) [1]<ul style="list-style-type: none">barCode [39-291-1234]

Modeling Condition/Action Rule 1

Model



Model C/A Rule 1

The first Condition on our Rulesheet is used to identify any items purchased from department **290**, the **Floral Department**. For each item identified, we want to give the customer a coupon (using the **.new** operator) for a free balloon.

Notice the assignment of the value 12/31/9999 to the **expirationDate** attribute, which is a common way to indicate that the expiration date is essentially indefinite. There are other ways to accomplish this. For example, the entity **Coupon** might have a boolean attribute named **expires**, to which a **true** or **false** value could be assigned inside the **.new** expression.

For detailed steps see:

**Advanced Rule Modeling:
Version 5.3 Modeling
condition/action rule 1**
section of the tutorial
available through online
Eclipse help for details on
how to generate the floral
department coupon.

Scope

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

Filters

Ref	ID	Post	Alias	Text	Rule Name
1		Info	currentCart	One free balloon for every item purchased {allItems.name} from the floral department	

Conditions

Ref	ID	Post	Alias	Text	Rule Name
a				allItems.department	

Actions

Ref	ID	Post	Alias	Text	Rule Name
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']	

Rule Statements

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

Testing Condition/Action Rule 1

Test



Test for the Floral Department

In this Ruletest, we want to make sure that when an item has been purchased from the **Floral Department** (department **290** according to the code table) that a new **Coupon** is created entitling the customer to one free balloon. The coupon's expiration date should also be created, though here we use a date that makes this coupon essentially non-expiring.

*coupons.ers *coupons3.ert X

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]



Test Results

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**.

Condition/Action Rule 1 Test Results

Test



Results of our Test

Department **290** has been recognized and the informational message has been posted. Also, our new **Coupon** entity has been created, displaying a value of **One Free Balloon** in the **description** attribute and **12/31/9999** in the **expirationDate** attribute, indicating that the coupon will not expire (practically speaking).

Also, note the new Message posted by our new rule: it contains the value of **allItems.name** "embedded" inside. The syntax to embed attributes is shown in the Rule Statement and is discussed in more detail in the *Rule Language Guide*, "Special Syntax" chapter.

The screenshot displays a rule engine interface with two shopping cart objects and a coupon object. The left cart contains three items: Tulips (department 290), Tomato Juice (department 285), and another item. The right cart contains the same three items, but the department for the first item is updated to 290. A new coupon is created with the description 'One free balloon for every item purchased[Tulips - 1 dozen]from the floral department' and an expiration date of 12/31/9999. Dashed orange arrows indicate the flow of data from the left cart's items to the right cart's items and to the coupon's description. The bottom panel shows a table of rule messages.

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 Condition/Action Rule 2 – Filter

Model



Filters – Continued

The next Business Rule to be modeled creates a “\$2 off” coupon when a customer buys 3 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 3 or more items were purchased from the Soda department, we won't count all items in the shopping cart, just those from the Soda Department. To help us, we'll use the **SodaItems** alias we defined earlier in the **Scope** section.

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.

For detailed steps see:

Advanced Rule Modeling: Version 5.3 Modeling condition/action rule 2 using filters section of the tutorial available through online Eclipse help to generate the soda department coupon.

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'

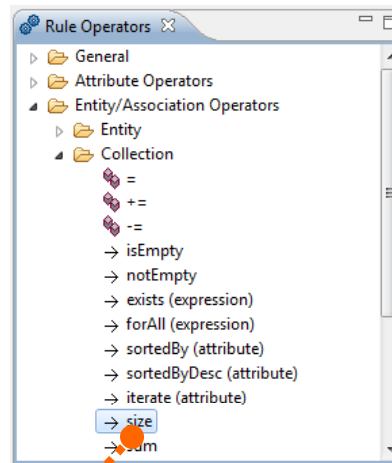
Modeling Condition/Action Rule 2

Model

Condition Row 2

The **→size** operator counts the number of elements in the **SodaItems** collection of data. If 3 or more, then the **\$2** off coupon is issued to the customer.

Please refer to the **Rule Language** Guide for more details about the **→size** operator, and all other operators available within Studio.



Action Row 2

The **expirationDate** attribute derives 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 issuance.

The screenshot shows the Studio interface with the rule configuration for 'coupons.ers'. The 'Conditions' section shows two conditions: 'a allItems.department' and 'b sodaItems->size >= 3'. The 'Actions' section shows five actions: '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)]', and 'E'. The 'Rule Statements' section shows a table with columns: Ref, ID, Post, Alias, and Text.

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

Testing Condition/Action Rule 2

Test



Test for the Soda Department

To test this rule, our shopping cart must contain 3 or more items from the **Soda Department** (department **285**).

When the condition determines that 3 or more Soda items are present (counted using the **→size** operator) then the action will fire and generate the appropriate coupon.

coupons.ers *coupons4.ert

untitled_1

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]

For detailed steps see:
Advanced Rule Modeling: Version 5.3
Testing the condition/action rule 2 section of the tutorial available through online Eclipse help to align the test data to get the illustrated results.

Condition/Action Rule 2 Test Results

Test

Results of the Ruletest

The items from the **Soda Department** have been identified and counted, the **Coupon** has been added with a “\$2 off next purchase” description and an **expirationDate** of **04/02/2009** (which is 1 year from the date this test was run), and the informational message has been posted.

Coupon rule 2 works as expected!

coupons.ers *coupons4.ert

untitled_1

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

Input	Output	Expected
<ul style="list-style-type: none"> Customer [1] <ul style="list-style-type: none"> name preferredCard (PreferredAccount) [1] <ul style="list-style-type: none"> cardNumber cumulativeCashBack [0.000000] ShoppingCart (ShoppingCart) [1] <ul style="list-style-type: none"> savings totalAmount useCashBack Item (Item) [1] <ul style="list-style-type: none"> barCode [32-285-12345] department name [Coke(12 pack)] price [3.990000] Item (Item) [2] <ul style="list-style-type: none"> barCode [32-285-23456] department name [Pepsi (12 pack)] price [3.990000] Item (Item) [3] <ul style="list-style-type: none"> barCode [32-285-34567] department name [Sprite (12 pack)] price [3.990000] 	<ul style="list-style-type: none"> Customer [1] <ul style="list-style-type: none"> isPreferredMember [true] name preferredCard (PreferredAccount) [1] <ul style="list-style-type: none"> cardNumber cumulativeCashBack [0.239400] ShoppingCart (ShoppingCart) [1] <ul style="list-style-type: none"> cashBackEarned [0.239400] savings totalAmount [11.970000] useCashBack Item (Item) [1] <ul style="list-style-type: none"> barCode [32-285-12345] department [285] name [Coke(12 pack)] price [3.990000] Item (Item) [2] <ul style="list-style-type: none"> barCode [32-285-23456] department [285] name [Pepsi (12 pack)] price [3.990000] Item (Item) [3] <ul style="list-style-type: none"> barCode [32-285-34567] department [285] name [Sprite (12 pack)] price [3.990000] Coupon [1] <ul style="list-style-type: none"> description [\$2 off next purchase] expirationDate [11/28/13] 	

Rule Statements Rule Messages

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]

Modeling Condition/Action Rule 3

Model

The third condition on our Rulesheet is used to identify when a customer's **totalAmount** exceeds the **\$75** threshold prescribed in the scenario and award 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.

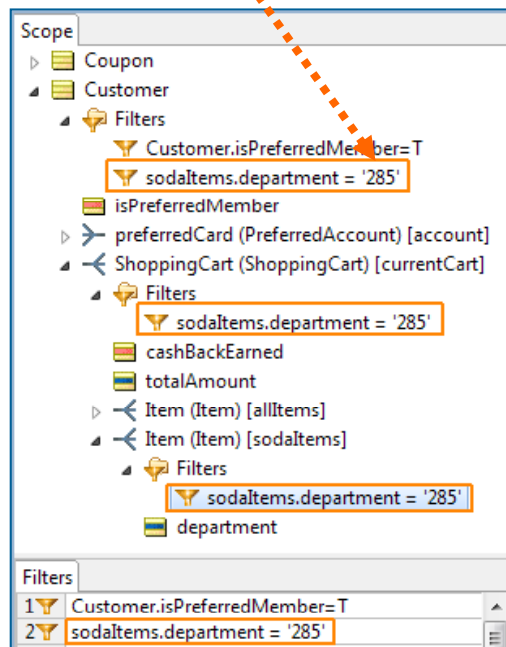
For detailed steps see:
Advanced Rule Modeling: Version 5.3
 Testing the condition/action rule 2 section
 of the tutorial available through online
 Eclipse help to align the test data to get
 the illustrated results.

Filters

Model

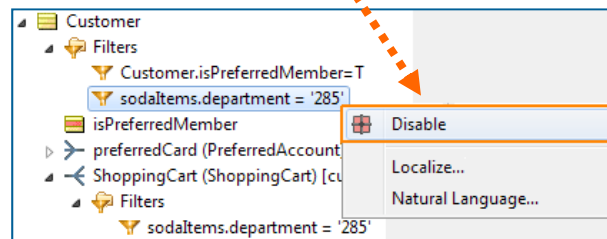
Full Filters

The filter we created is applied to every relevant level in the scope. It is a **full filter**, applying to the Customer, the currentCart, and -- the level we want to filter -- the Items.



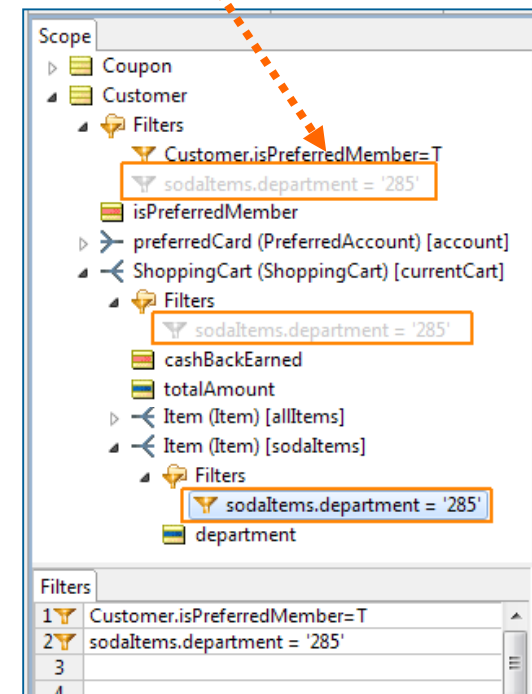
Disabling Filters

We can disable the filter at selected levels to make it a **limiting filter** by right-clicking on a filter level and then selecting **Disable**.



Limiting Filters

When we disable the filter on the Customer and currentCart, their values are greyed out.



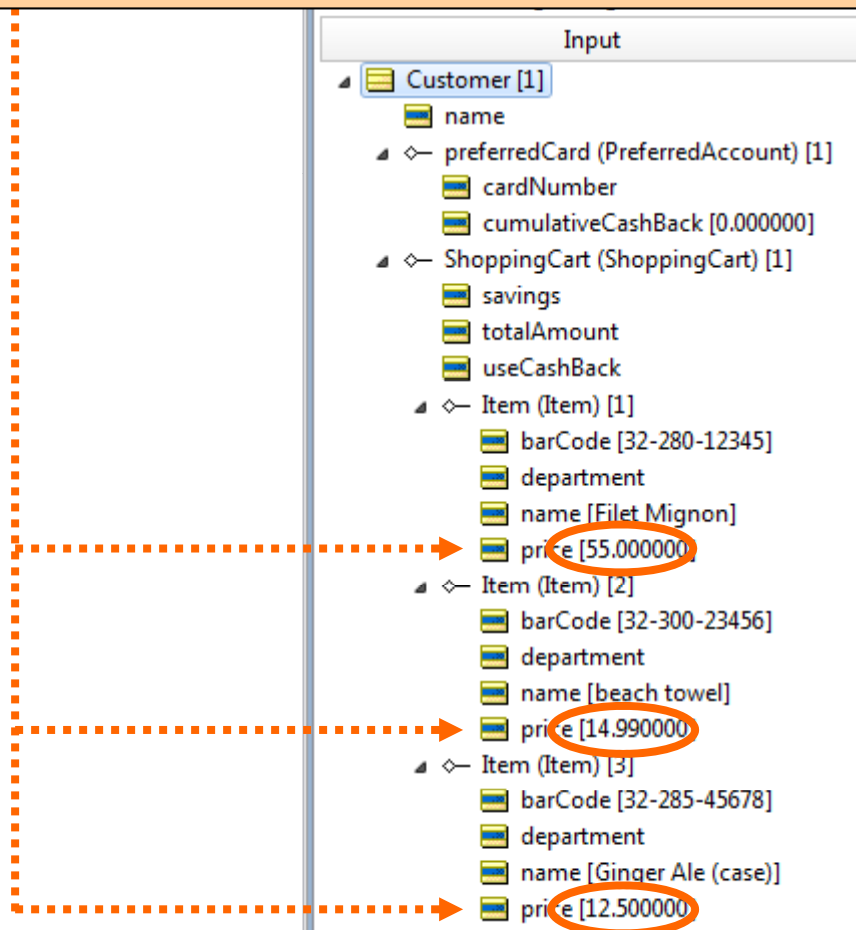
Testing Condition/Action Rule 3

Test



Total Amount Test

For our 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.



Summary - Condition/Action Rule 3 Test Results

Test

Results of the Test

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

Our 3rd Condition/Action rule works!

The screenshot displays a software interface for testing a Condition/Action Rule. The interface is divided into two main sections: 'Input' and 'Output'.

Input Section:

- 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 Section:

- 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 Messages Panel:

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

'coupons' Rulesheet Completed

Model



'coupons' Rulesheet

Here's our second completed Rulesheet!

*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

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

Rule Statements

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

Rule Messages

Modeling the 'use_cashBack' Rulesheet

Model



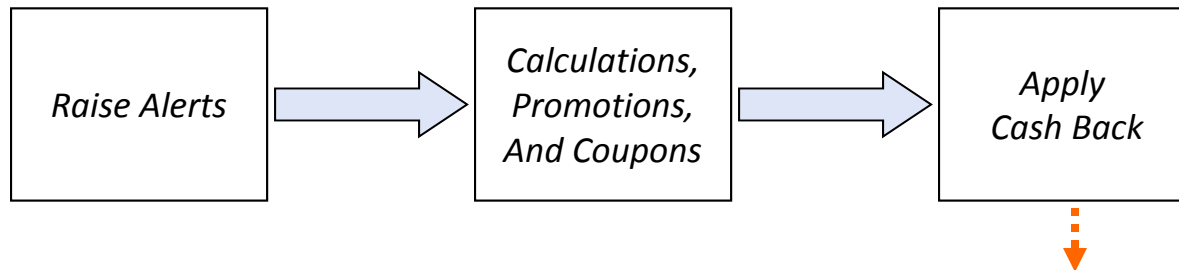
Action

The previous 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 his **cumulativeCashBack** account to reduce his total amount at checkout time. We'll assume that at time of checkout, the cashier asks the shopper if he wants to apply his **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.



- 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.

'use_cashBack' Rulesheet Scope

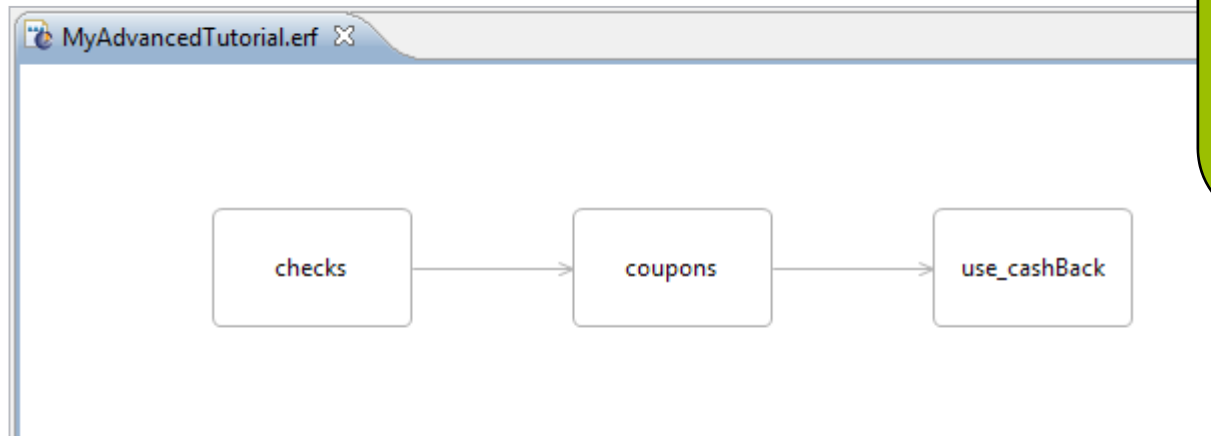
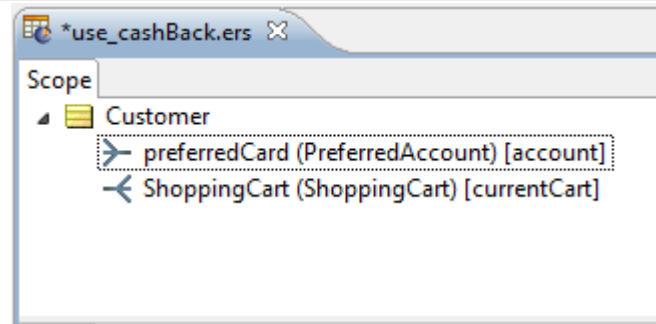
Model



Rulesheet Scope

Now we will build our third and final Rulesheet and call it **use_cashBack**. Notice that with the addition of this third Rulesheet, we can complete our **Ruleflow Diagram** indicating execution sequence of the 3 Rulesheets.

Because the rules on this Rulesheet deal with a preferred shopper's cart, we only need a few aliases to represent these perspectives of our data.



For detailed steps see:
Advanced Rule Modeling: Version 5.3
Scope of the third Rulesheet section of the tutorial available through online Eclipse help to create the new Rulesheet's scope. Also, Extending the Ruleflow section for extending the Ruleflow.

Filters

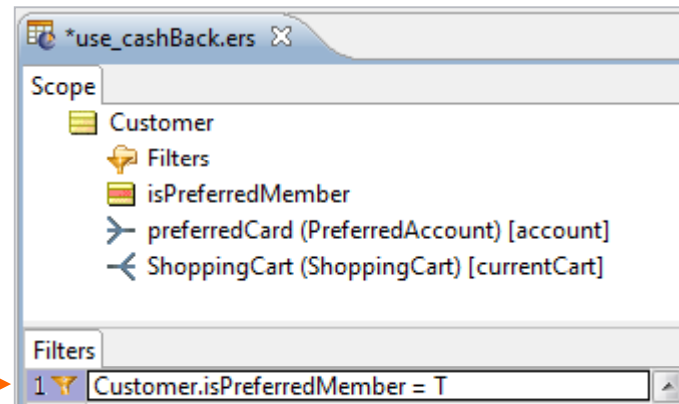
Model



Filters

The first thing we want to accomplish with this Rulesheet is to make sure we only evaluate preferred customers since they are only customers 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 don’t have a Preferred Card).



For detailed steps see:
**Advanced Rule
Modeling: Version 5.3**
Using filters section of
the tutorial available
through online Eclipse
help to create a filter.

Modeling Condition/Action Rule 1

Model



Condition/Action Rule 1

We only need to create one Condition/Action rule here, with one **Condition** and a few **Actions**. As you can see in the illustration below, we only want to process the **currentCart** when the shopper has chosen to apply his or her **cashBack** balance to the current purchase, in other words, when **useCashBack = true**.

Then, we'll deduct the **cumulativeCashBack** balance from **totalAmount**, as shown below.

In keeping with our model/test approach, we'll test the rule before adding more to it. Also, we'll postpone adding a **Rule Statement** until we have completed the rule model.

For detailed steps see:
Advanced Rule Modeling: Version 5.3
Modeling and testing the first business rule, Modeling condition/action rule 1 subsection of the tutorial available through online Eclipse help to define the Condition/Action rule one.

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

Actions			
Post Message(s)			
A	currentCart.totalAmount -= account.cumulativeCashBa...	<input type="checkbox"/>	<input checked="" type="checkbox"/>
B			
C			
D			
E			

Filters	
1	Customer.isPreferredMember = T
2	
3	

Testing Condition/Action Rule 1

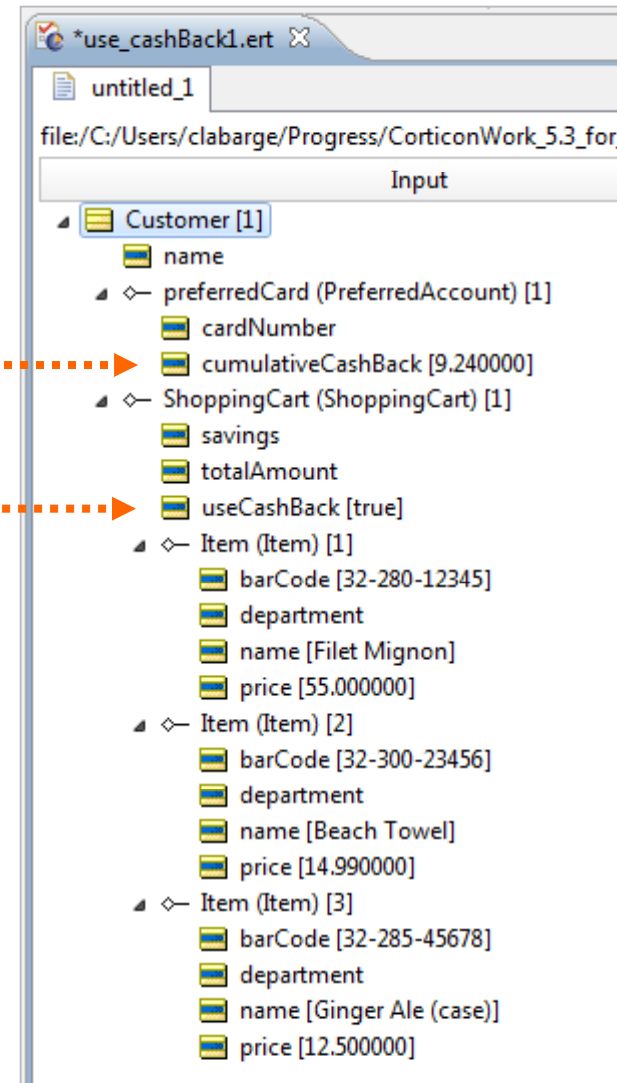
Test



Use Cash Back Test

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

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 final price.



Condition/Action Rule 1 Test Results

Test



Test Results

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

We also see some of those values embedded in the 3rd Message displayed at the bottom.

But we're not done. We still need to reset the **cumulativeCashBack** attribute to **0**. Let's modify the rule to add the necessary logic.

untitled_1

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

Input	Output
Customer [1]	Customer [1]
name	isPreferredMember [true]
preferredCard (PreferredAccount) [1]	name
cardNumber	preferredCard (PreferredAccount) [1]
cumulativeCashBack [9.240000]	cardNumber
ShoppingCart (ShoppingCart) [1]	cumulativeCashBack [10.889800]
savings	ShoppingCart (ShoppingCart) [1]
totalAmount	cashBackEarned [1.649800]
useCashBack [true]	savings
Item (Item) [1]	totalAmount [71.600200]
barCode [32-280-12345]	useCashBack [true]
department	Item (Item) [1]
name [Filet Mignon]	barCode [32-280-12345]
price [55.000000]	department [280]
Item (Item) [2]	name [Filet Mignon]
barCode [32-300-23456]	price [55.000000]
department	Item (Item) [2]
name [Beach Towel]	barCode [32-300-23456]
price [14.990000]	department [300]
Item (Item) [3]	name [Beach Towel]
barCode [32-285-45678]	price [14.990000]
department	Item (Item) [3]
name [Ginger Ale (case)]	barCode [32-285-45678]
price [12.500000]	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

Adding Action Rows B & C

Model



Adding Actions Rows B & C

Before we reset **cumulativeCashBack** to B, let's ensure our preferred customer is aware of her savings today. Let's assign the value of **cumulativeCashBack** to the attribute named **savings**. We'll assume that this **savings** value will be printed on a receipt, displayed on a screen, or by some other mechanism made visible to the shopper.

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

Adding a Rule Statement completes this business rule model.

The screenshot displays the IBM Business Rule Editor interface for a rule named `*use_cashBack.ers`. The interface is divided into several panes:

- Scope:** A tree view showing the rule's scope. It includes a `Customer` object with a `Filters` property (containing `Customer.isPreferredMember = T`), an `isPreferredMember` property, a `preferredCard (PreferredAccount) [account]` object with a `cumulativeCashBack` property, and a `ShoppingCart (ShoppingCart) [currentCart]` object with `savings`, `totalAmount`, and `useCashBack` properties.
- Filters:** A list of filter conditions. The first filter is `1 Customer.isPreferredMember = T`.
- Conditions:** A table with 9 rows (a-i) and 3 columns. The first row (a) contains the condition `currentCart.useCashBack`. The second column has values `-` and `T` in rows a and b respectively. The third column has a value `T` in row b.
- Actions:** A table with 8 rows (A-H) and 3 columns. The first row (A) is labeled `Post Message(s)`. The second row (B) contains the action `currentCart.totalAmount -= account.cumulativeCashBack`. The third row (C) contains the action `currentCart.savings = account.cumulativeCashBack`. The fourth row (D) contains the action `account.cumulativeCashBack = 0`. The second column has checkboxes for rows B, C, and D. The third column has checkboxes for rows B, C, and D, all of which are checked.
- Rule Statements:** A table with 5 columns: `Ref`, `ID`, `Post`, `Alias`, and `Text`. The first row (1) has `Info` in the `Post` column, `currentCart` in the `Alias` column, and the text `cashback bonus has been deducted from the total. New total = ${currentCart.totalAmount}. Today's savings = ${currentCart.savings}` in the `Text` column.

Summary - Final Condition/Action Rule 1 Test Results

Test

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 below explaining what has happened.

Our final rule works as expected!

Since this was a cumulative test, we also can verify that the entire Ruleflow (all 3 Rulesheets) work as expected. The Scenario has now been fully modeled and tested.

Input	Output
<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBack [9.240000]ShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">savingstotalAmountuseCashBack [true]Item (Item) [1]<ul style="list-style-type: none">barCode [32-280-12345]departmentname [Filet Mignon]price [55.000000]Item (Item) [2]<ul style="list-style-type: none">barCode [32-300-23456]departmentname [Beach Towel]price [14.990000]Item (Item) [3]<ul style="list-style-type: none">barCode [32-285-45678]departmentname [Ginger Ale (case)]price [12.500000]	<ul style="list-style-type: none">Customer [1]<ul style="list-style-type: none">isPreferredMember [true]namepreferredCard (PreferredAccount) [1]<ul style="list-style-type: none">cardNumbercumulativeCashBack [0.000000]ShoppingCart (ShoppingCart) [1]<ul style="list-style-type: none">cashBackEarned [1.649800]savings [10.889800]totalAmount [71.600200]useCashBack [true]Item (Item) [1]<ul style="list-style-type: none">barCode [32-280-12345]department [280]name [Filet Mignon]price [55.000000]Item (Item) [2]<ul style="list-style-type: none">barCode [32-300-23456]department [300]name [Beach Towel]price [14.990000]Item (Item) [3]<ul style="list-style-type: none">barCode [32-285-45678]department [285]name [Ginger Ale (case)]price [12.500000]Coupon [1]<ul style="list-style-type: none">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

'use_cashBack' Rulesheet Completed

Model



Completed 'use_cashBack' Rulesheet

Here's our third, and final, completed Rulesheet!



Final Note 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 incompleteness in each of the 3 Rulesheets.

The Completeness Check and the other Studio Logical Analysis and Validation tools are covered in more detail in the **Basic Rule Modeling Tutorial** and in a special chapter in the **Rule Modeling Guide**. Identifying and resolving incompleteness or conflicts in these rules are left to you.

The screenshot displays the Studio interface for the 'use_cashBack.ers' rulesheet. The Scope tree on the left shows the hierarchy: Customer (Filters, isPreferredMember, preferredCard (PreferredAccount) [account], cumulativeCashBack, ShoppingCart (ShoppingCart) [currentCart], savings, totalAmount, useCashBack). The Conditions table has columns for Conditions, 0, and 1. The Actions table has columns for Actions, Post Message(s), and a checkbox column. The Rule Messages table at the bottom shows a single message with Ref 1, ID, Post Info, Alias currentCart, and Text 'cashback bonus has been deducted from the total. New total = \${currentCart.totalAmount}. Today's savings = \${currentCart.savings}'.

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

Actions	Post Message(s)	
A	currentCart.totalAmount -= account.cumulativeCashBack	<input checked="" type="checkbox"/>
B	currentCart.savings = account.cumulativeCashBack	<input checked="" type="checkbox"/>
C	account.cumulativeCashBack = 0	<input checked="" type="checkbox"/>
D		
E		
F		
G		

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}

Summary/What You've Learned



Summary – What You've Learned

Congratulations on completing the Corticon Advanced Rule Modeling Tutorial! We hope you have found this tutorial useful in your quest to get the most out of Corticon Studio, the best business rule modeling system in the business. You have learned to incorporate some of Studio's more powerful functionality into your rule modeling process, including:

Diagramming a Vocabulary – Based on the necessary items identified during analysis of a Business Problem, we've shown you how to diagram a Vocabulary using an ER Diagram from which you can confidently create a valid Studio Vocabulary for use in rules modeling and testing.

Scope and Aliases – Scope helps us tell the Corticon rules engine which data to use when evaluating and executing rules. Using Scope to incorporate associations between entities, and defining Aliases to represent it in your rules, establishes the context in which your data is analyzed.

Collections and Collection Operators – You now know that a Collection is often comprised of one entity associated with one or more other entities, which we call elements of the collection, and that Collection Operators are used to analyze groups of entities rather than individuals. Studio contains a number of Collection Operators which operate on the Collections you create and it's mandatory to use them with Aliases that represent those collections.

Condition/Action Column 0 – We've shown you how to use this portion of a Rulesheet to perform mathematical calculations which can contribute data to other rules in the Rulesheet, or in downstream Rulesheets in the same Ruleflow.

Filters – You've learned that a Filter expression acts to 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.

Sequencing Rulesheets using Ruleflows – If a natural sequence or "flow" of logical steps can be identified within a single decision step, it often makes sense to organize the flow using separate Rulesheets for each logical step. Rulesheets will execute in a sequence determined by their order in the Ruleflow. Using multiple Rulesheets helps us both visualize our logic and maintain and reuse it more easily.

Extended Transient Attributes – You are now aware that some attributes are little more than "intermediate" or "temporary" value holders. We don't need to return these values in a response, or save them in a database. In Studio, a special type of attribute called Extended Transient fills this purpose.

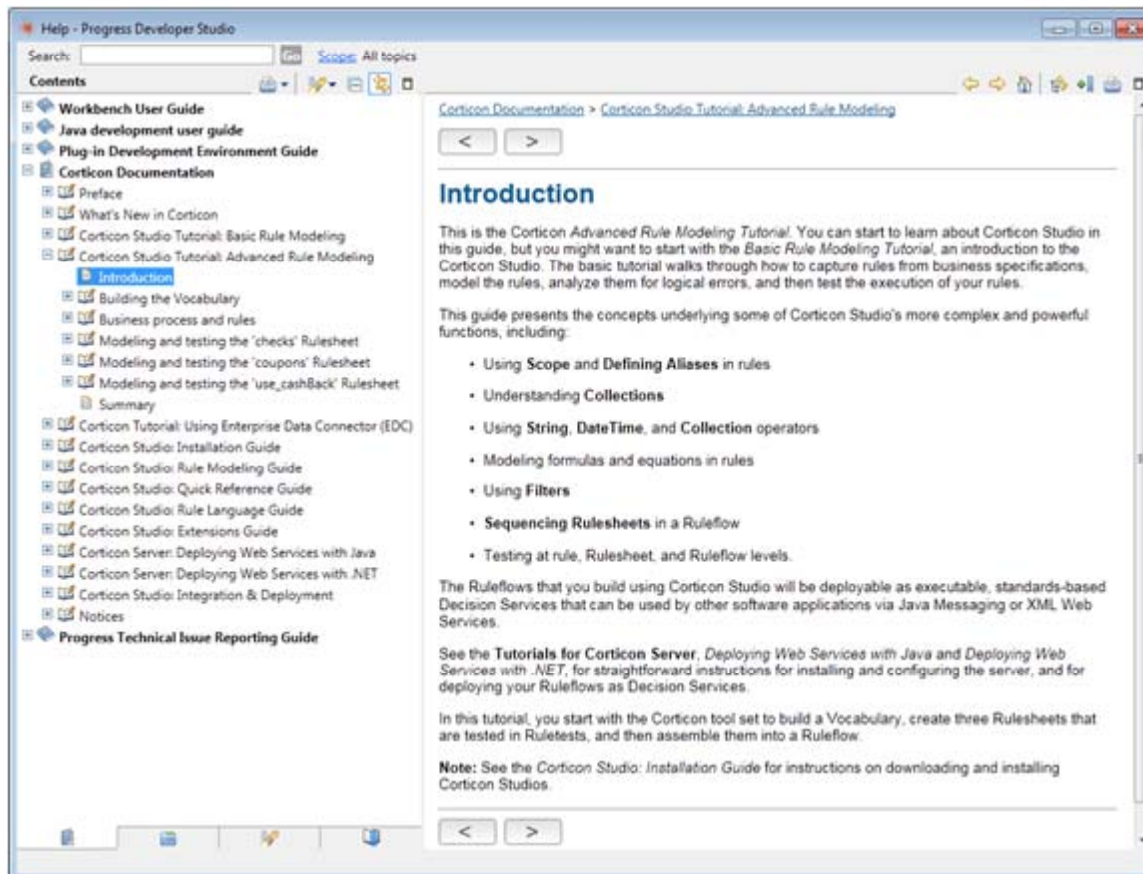
Embedding Attributes within Rule Statements – We've shown you how you can embed attributes within Rule Statements and the proper syntax to use when doing so.

Rule Statement Messaging – We've explained the value of Studio's Rule Statement Messaging for use in posting 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.

Appendix A: Corticon Technical Publications

PROGRESS
software

Corticon Studio and Corticon Studio for Analysts both provide the whole doc set in online help.



The whole doc set is also available as a package of books in PDF format, available on the Progress Software download site.

