Progress®

Corticon Tutorial

Advanced Rule Modeling in Corticon Studio

Progress® Corticon®

Copyright

Last updated with new content: Corticon 6.1.1

Updated: 2020/06/25

Table of Contents

1

Tutorial - Advanced Rule Modeling in Corticon Studio

In the Basic Rule Modeling Tutorial you learned how to capture rules from business specifications, model the rules, analyze them for logical errors, and test the execution of your rules in Corticon Studio, all without programming.

The Advanced Rule Modeling Tutorial builds on what you learned in the Basic Tutorial. In the Advanced Tutorial, you will learn how to use some of Corticon Studio's more complex and powerful functions, including:

- Building a Vocabulary with associations—Associations enable you to define relationships between entities. For example, many items can be associated with one shopping cart.
- Using Scope and Aliases in rules—Scope and Aliases enable you to define rules that apply to an entity in relation to another entity. For example, if the total price of items in a customer's shopping cart exceeds $100, give the customer a coupon.
- Creating action-only rules—Rules in column 0 of a Rulesheet. They are non-conditional rules: the rules always fire so the action always applies. For example, calculate the total price of items in a shopping cart.
- Using equations in rules
- Using Collections and Collection operators—Collections enable you to define rules that apply to a group of entity instances. For example, check to see if any items in a Shopping Cart are from the Liquor department.
- Using Filters in rules—Filters enable you to filter out data, so only entities that pass the filter criteria are evaluated by rules in a Rulesheet.
- Using a variety of attribute and entity operators in rules
- Sequencing Rulesheets in a Ruleflow

- Embedding attributes within rule statements—This feature enables you to retrieve the value of an attribute instead of hard-coding it in a rule message. For example, "${ShoppingCart.cashBackEarned} bonus earned today".
- Testing at the Ruleflow level

Just like the Basic Tutorial, you will follow the rule development lifecycle. You will discover, model, and test rules. Since the focus of this tutorial is to teach you how to build complex rule models, we will skip the Analyze phase to save time.

This tutorial is designed for hands-on use. We recommend that you follow along in Corticon Studio, using the instructions and illustrations that are provided. If you haven't installed Corticon Studio yet, install it now. Click here for instructions on installing Corticon Studio.

The business problem

Like the flight planning example in the Basic Rule Modeling Tutorial, the Advanced Tutorial uses a business case which you will use to build a rule model in Corticon Studio. The business case is as follows:

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

In addition to the minimum cash register functionality (adding up the price of items in a customer's shopping cart, for example), the new system should also be able to apply:

- Promotional rules
- Loyalty program rules
- Coupon generation rules
- Special warning rules that alert the cashier to take certain actions

Because every item in every store has a bar-coded label, the system's scanner should be able to determine complete information about each item, such as which department an 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 the total amount at any visit. The cashier will ask a Preferred Shopper if they would like to apply their cash back balance to their current purchase.
  - Once a Preferred Shopper chooses to apply their cash back balance, the cumulative cash back total maintained by the system will be reset to zero, and the accumulation of cash back will begin 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. This coupon has no expiration date.
  - Preferred Shoppers receive a coupon for $2 off on their next purchase when 3 or more Soda/Juice items are purchased in a single visit. This coupon has an expiration date of one year from the 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. The expiration date for this coupon is 3 months from the date of issue.

Additionally, 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. The new system should display an alert or warning on the cashier's screen, prompting them to check the customer's ID.

2

Discover the business rules

Discovering business rules involves two things—identifying what terms need to be included in the Vocabulary and identifying the business rules themselves. Let's start with the Vocabulary.

Identify Vocabulary terms and associations

To get started, we review the business problem and start compiling terms that need to be included in the Vocabulary. Based on this, we can identify the key entities and the assumptions we can make about each entity:

- Customers:
  - A Customer has a Name.
  - A Customer uses a Shopping Cart to carry items.
  - 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.
- Items being purchased:
  - An Item has a Name.
  - An Item has a Price.
  - An Item has a Bar-coded label.
  - An Item has a Department embedded in the Bar-coded label.

- Shopping Carts:
  - A Shopping Cart contains the Items a Customer purchases during each visit.
  - A Shopping Cart has a Total Amount.
  - If the Customer has a Preferred Shopper account, a Cash-Back Bonus is calculated using the Shopping Cart's Total Amount and is deducted from the Total Amount upon a Customer request.
- Coupons:
  - Coupons are issued to shoppers based on promotions.
  - A Coupon has a Description.
  - A Coupon has an Issue Date.
  - A Coupon has an Expiration Date.

Based on these assumptions, we can derive the attributes for each entity in our Vocabulary. Attributes are properties or characteristics that distinguish one instance of an entity from another. For example, each Item has attributes like name, a price, and bar code. The attribute values for each item make it unique.

This table lists the attributes for each entity along with their data type and attribute mode:

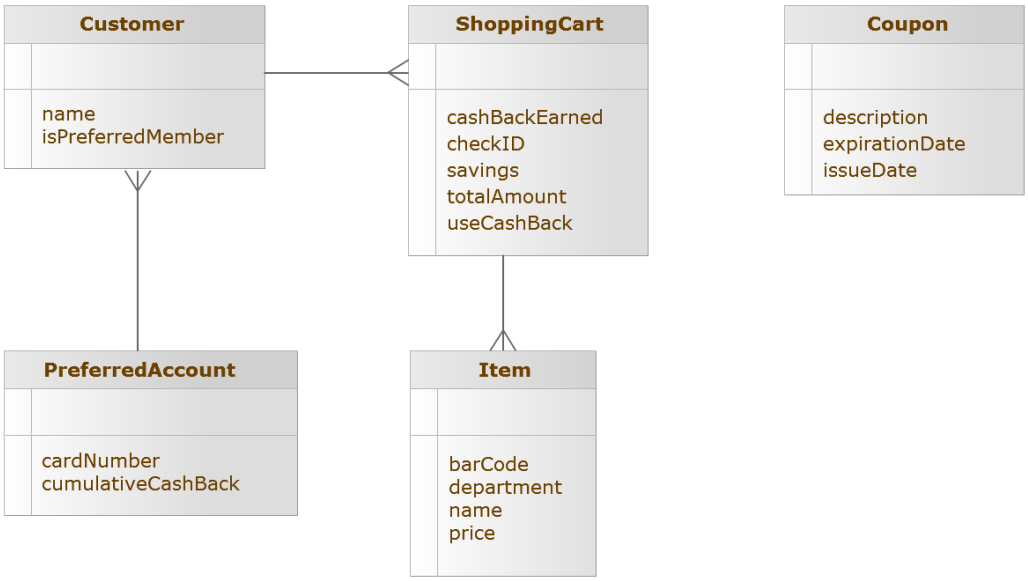
Entity	Attributes	Data Type	Attribute Mode
Customer			
	name	String	Base
	isPreferredMember	Boolean	Transient
Item			
	name	String	Base
	price	Decimal	Base
	department	String	Transient
	barCode	String	Base
ShoppingCart			
	totalAmount	Decimal	Base
	cashBackEarned	Decimal	Transient
	savings	Decimal	Base
	useCashBack	Boolean	Base
	checkID	Boolean	Base
PreferredAccount			
	cardNumber	String	Base
	cumulativeCashBack	Decimal	Base
Coupon			
	issueDate	Date	Base
	description	String	Base
	expirationDate	Date	Base

The mode of an attribute can be Base or Transient. Attributes that have a Base mode are attributes whose values will be sent to the rule model from a client application and/or returned to a client application from the rule model. Transient attributes are only used within the rule model. Their values are assigned or derived by rules, but not sent to a client application. For example, the cashBackEarned attribute is a Transient attribute that is used to update the value of the cumulativeCashBack attribute, which is a Base attribute.

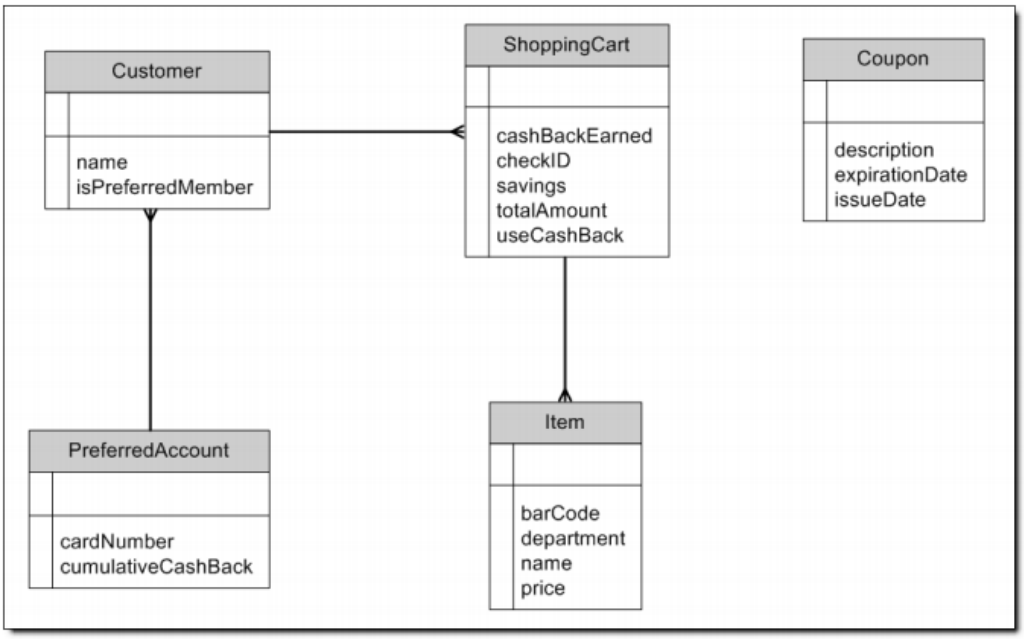
Next, let's identify the associations for our Vocabulary. An association defines the relationship between two entities. An association can be one-to-one, one-to-many, many-to-one, or many-to-many. In our grocery store business problem, we have the following associations:

- Many Customers (members of a family) can be associated with one PreferredAccount (many-to-one).
- One Customer can be associated with many ShoppingCarts over multiple visits (one-to-many).
- One ShoppingCart can be associated with many Items (one-to-many).

To make these relationships clear, we create a diagram of the associations. Creating a diagram is especially useful when you have a large or complex Vocabulary with many associations. Here is the diagram of entities and associations for our business problem:



In our diagram, the connectors between entities show the kind of relationship. For example, Customer has a one-to-many association with ShoppingCart.

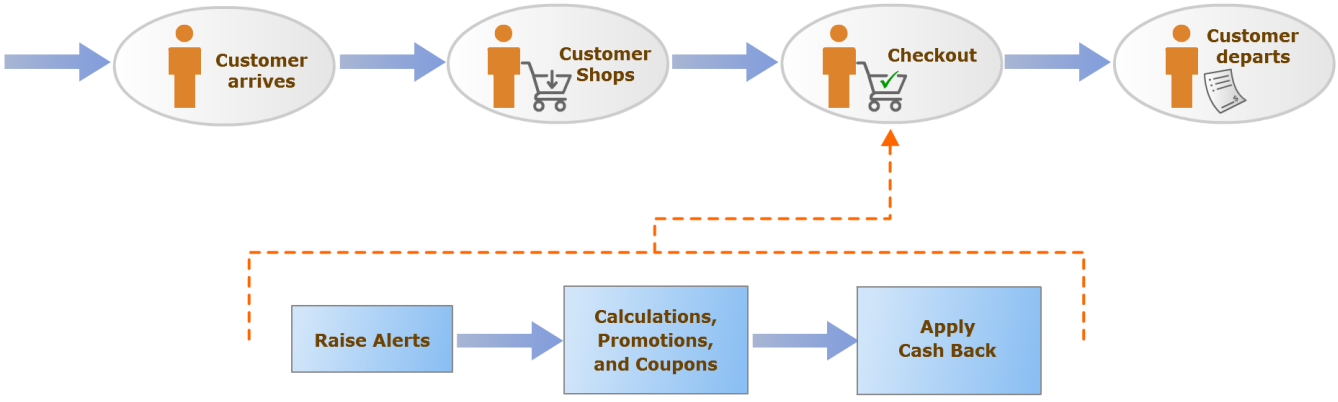


Identify the business rules

Next, let's identify specific business rules. 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.

Let's drill down into the Checkout step and define more detail about the rules inside. If a natural sequence or "flow" of logical substeps can be identified within a single decision step, it often makes sense to organize the substeps using separate Rulesheets, and then combining them into a Ruleflow. For the Checkout step, we identify three substeps as shown below. We will create a Rulesheet for each of these substeps and combine them into a Ruleflow.



Next, let's look at the business rules that we need to model for each substep:

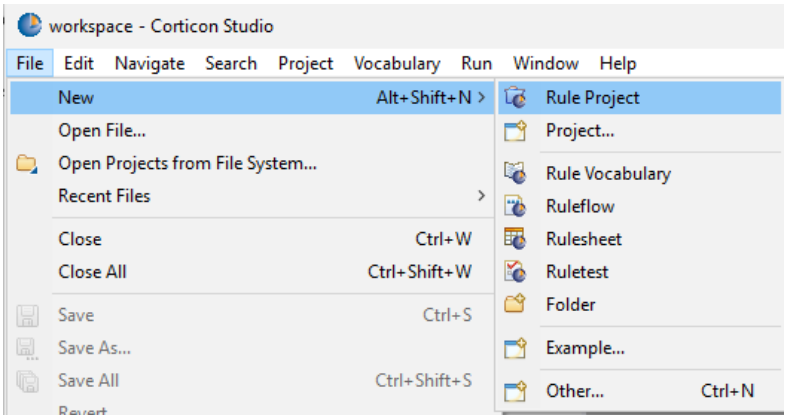
Substep	Rules
Raise Alerts	• Liquor purchases (any items from the Liquor department) can only be made by shoppers 21 or older
Calculations, Promotions, and Coupons	• Preferred Shoppers earn 2% cash back on all purchases at any branch. • Cash back earned by preferred shoppers should be added to the cumulative cash back in their preferred shopper account. • 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.
Apply Cash Back	• A Preferred Shopper account will track the accumulated cash back and allow the customer to apply it to reduce 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.

3

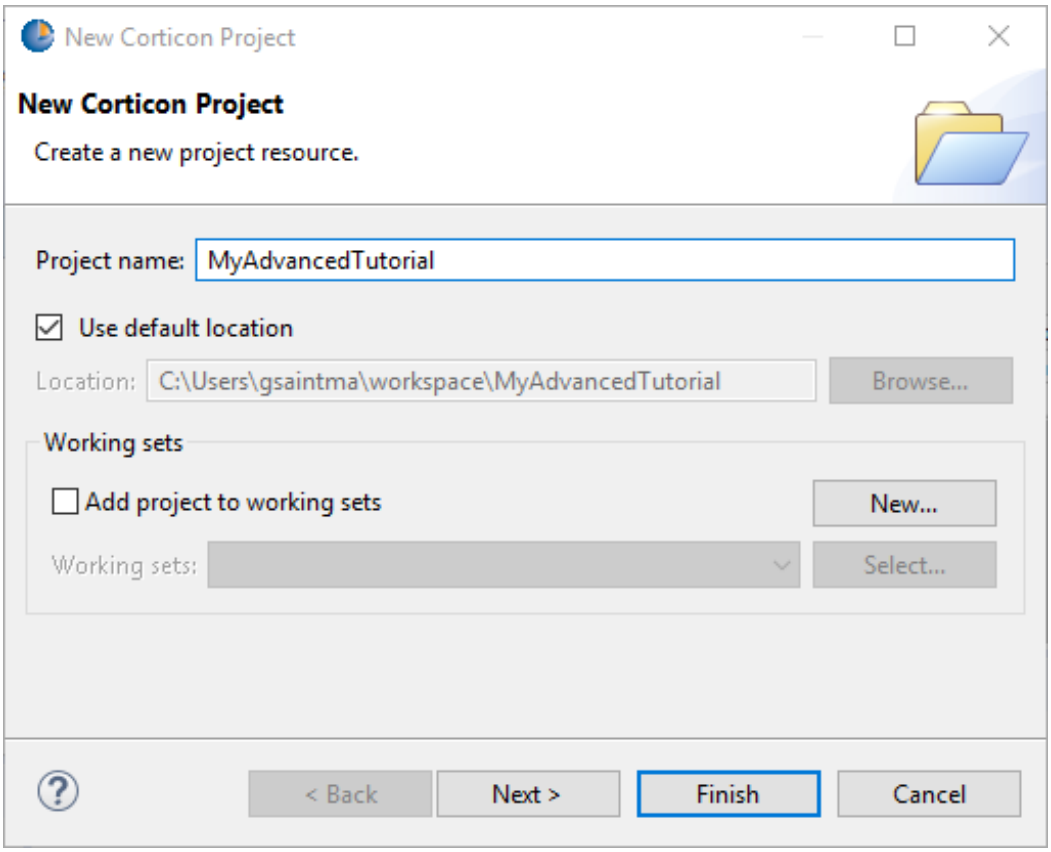
Build the Vocabulary

Now, let's implement the Vocabulary in Corticon Studio. To begin, launch Corticon Studio and create a Rule Project:

1. On the Start menu, select Progress > Corticon Studio. In the Workspace Launcher dialog box that opens, retain the default workspace and click OK. Corticon Studio opens.
2. Select File > New > Rule Project.



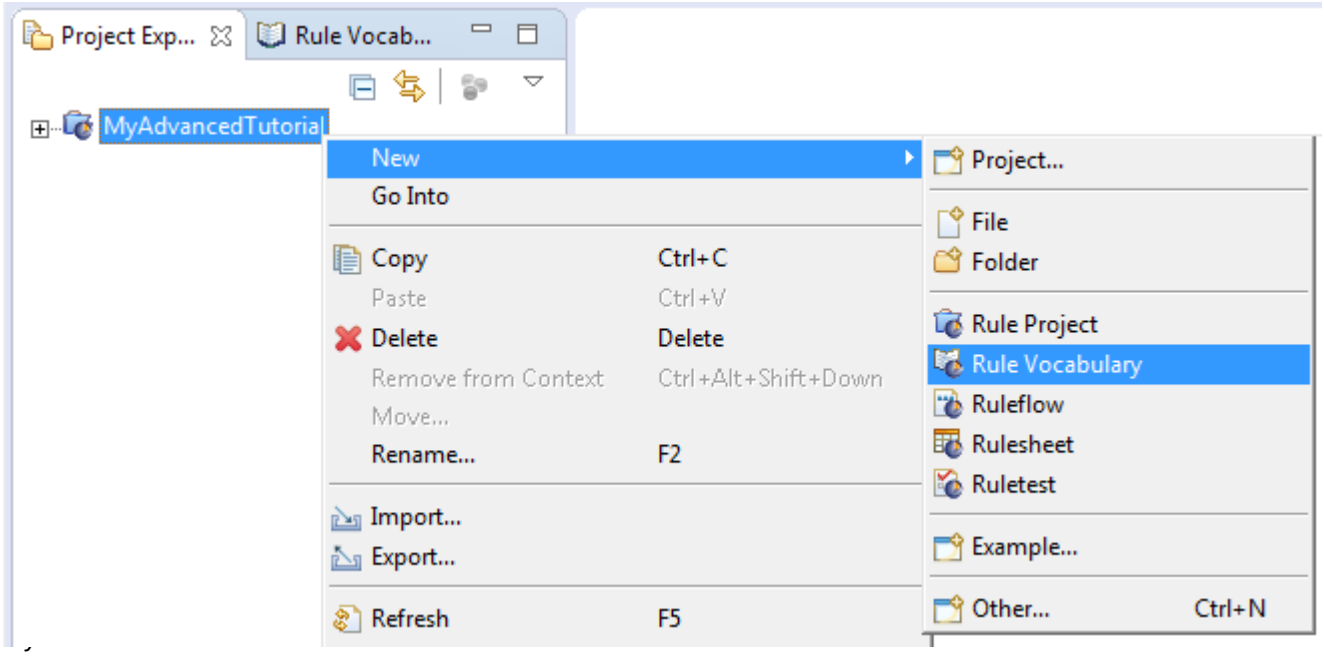
3. In the New Corticon Project window, enter MyAdvancedTutorial as the project name and click Finish.



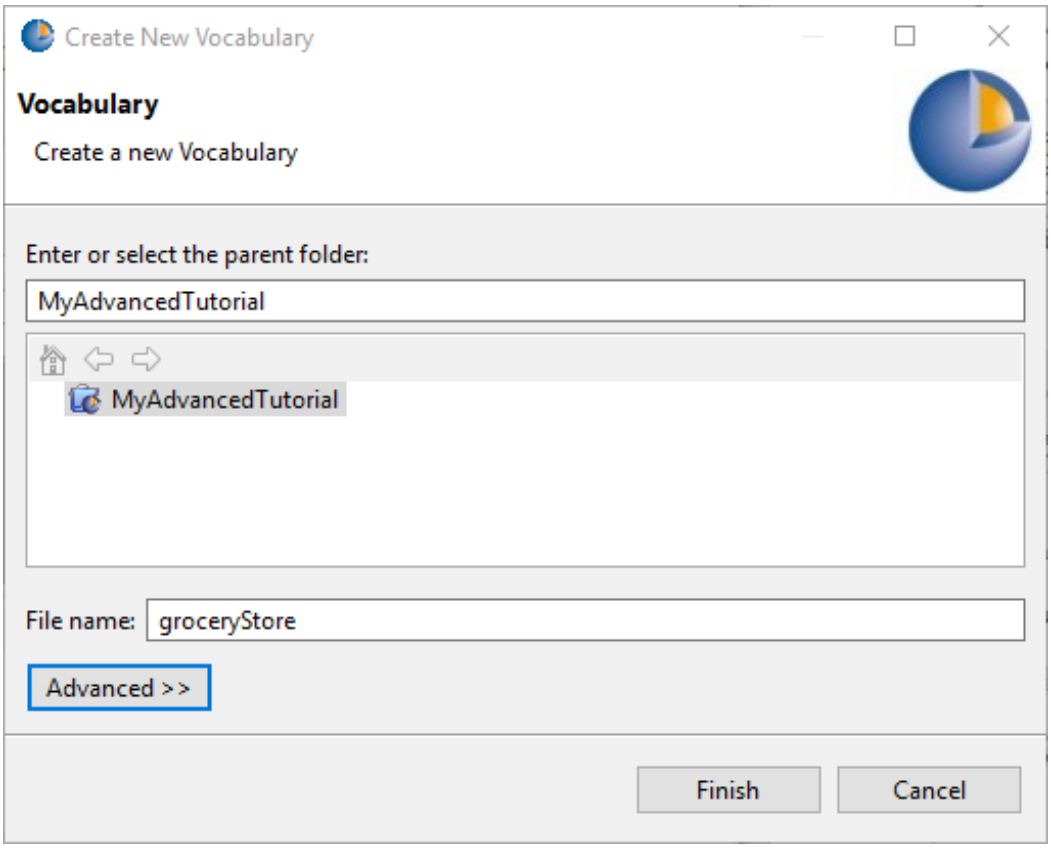
Create the Vocabulary

Next, create a Vocabulary file:

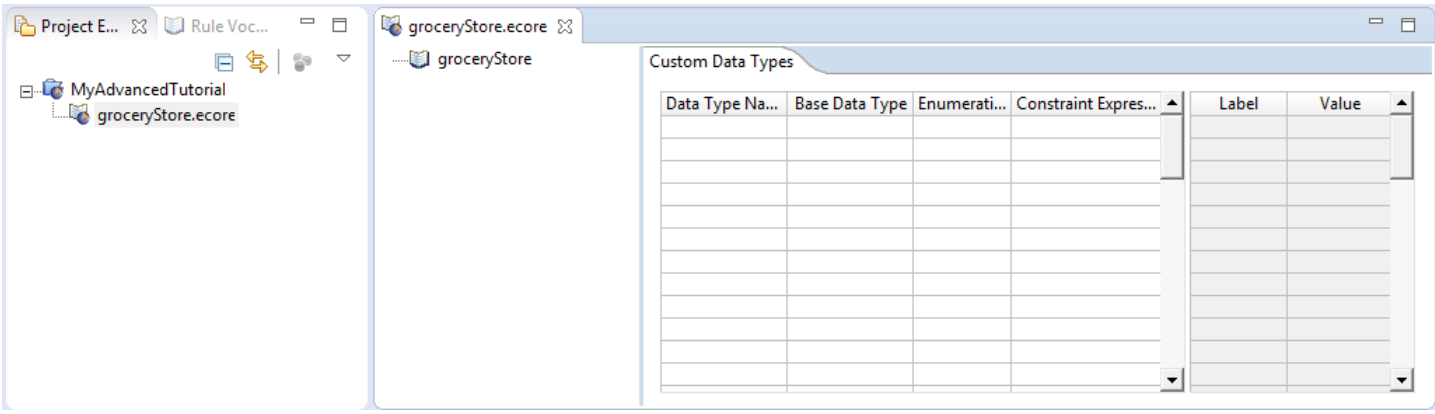
1. Right-click MyAdvancedTutorial and select New > Rule



2. In the Create a New Vocabulary window, enter groceryStore as the Vocabulary file name and click Finish.



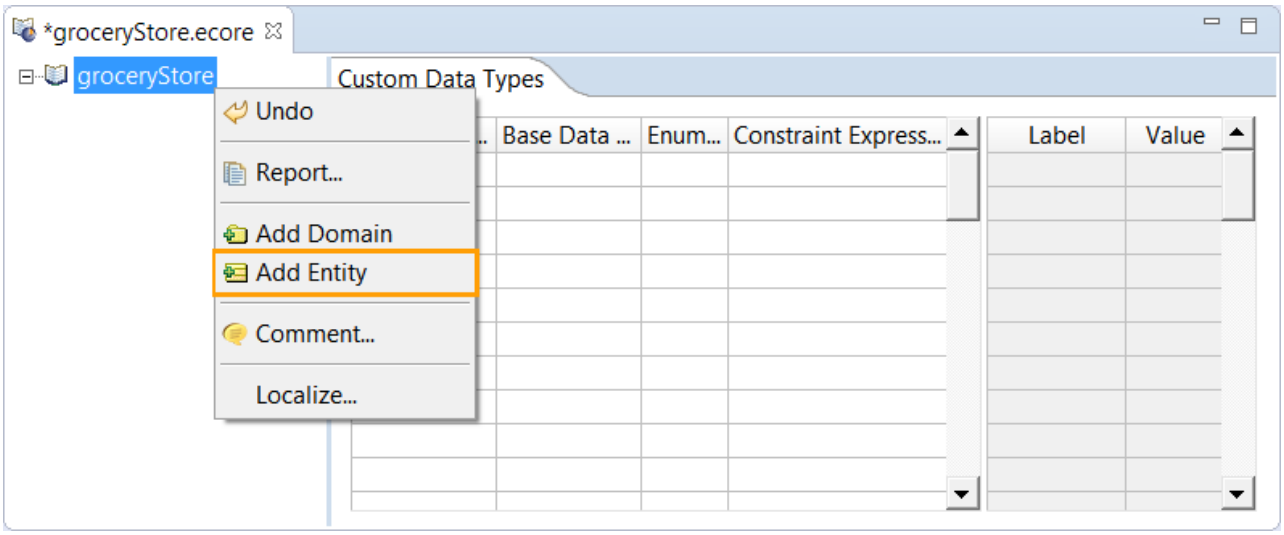
The Vocabulary opens under the rule project MyAdvancedTutorial as shown:



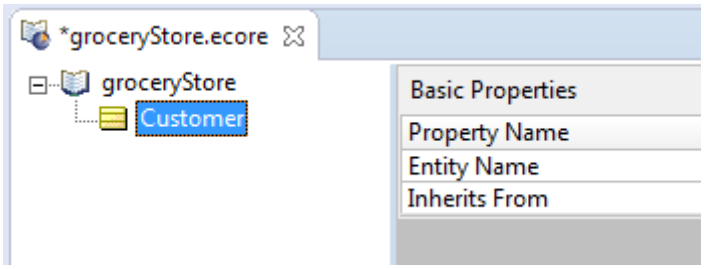
Add Entities

Now, let's add the entities (Customer, PreferredAccount, Item, ShoppingCart, Coupon):

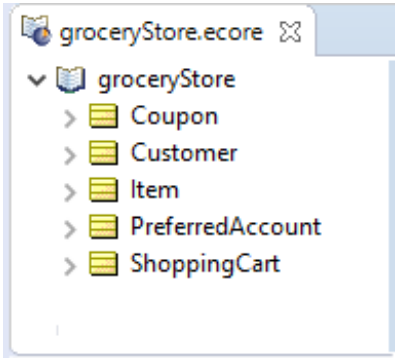
1. In the Vocabulary editor on the right, right-click groceryStore and select Add Entity.



2. An entity with the default name Entity_1 is created. Type over this and enter Customer.



3. Repeat these steps to add the remaining entities. The result will look like this:

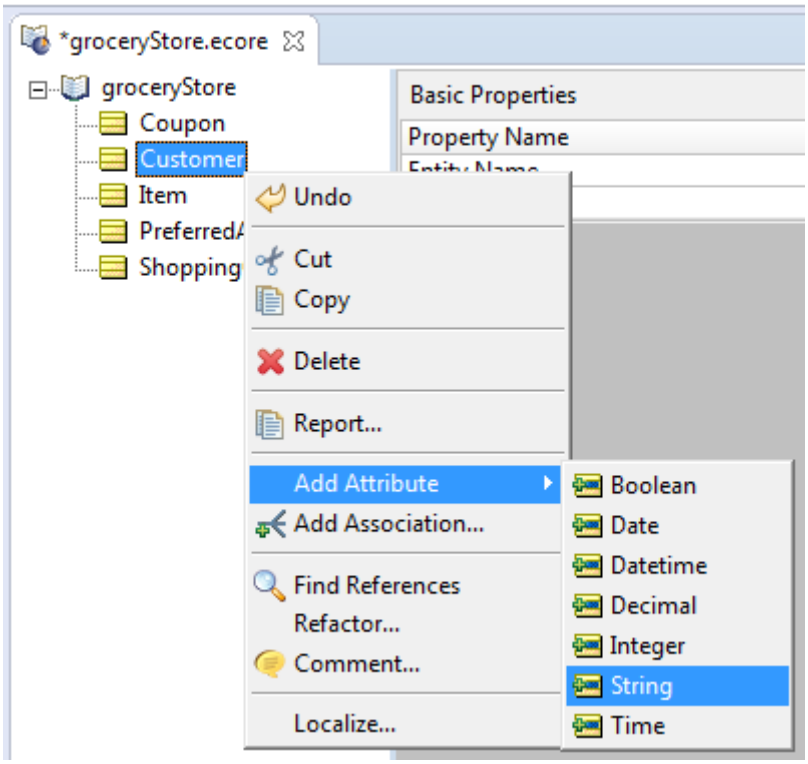


Add Attributes

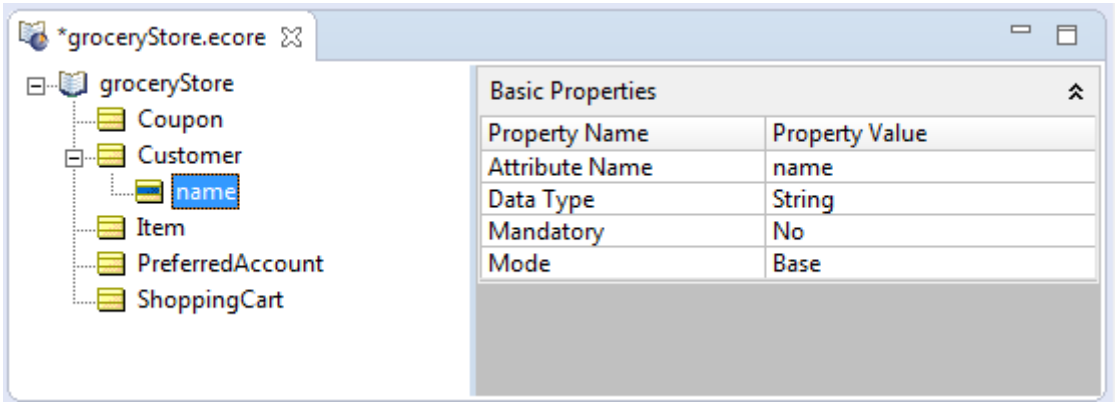
Next, let's add the attributes. Start by adding attributes for the Customer entity based on this table:

Entity	Attributes	Data Type	Attribute Mode
Customer			
	name	String	Base
	isPreferredMember	Boolean	Transient

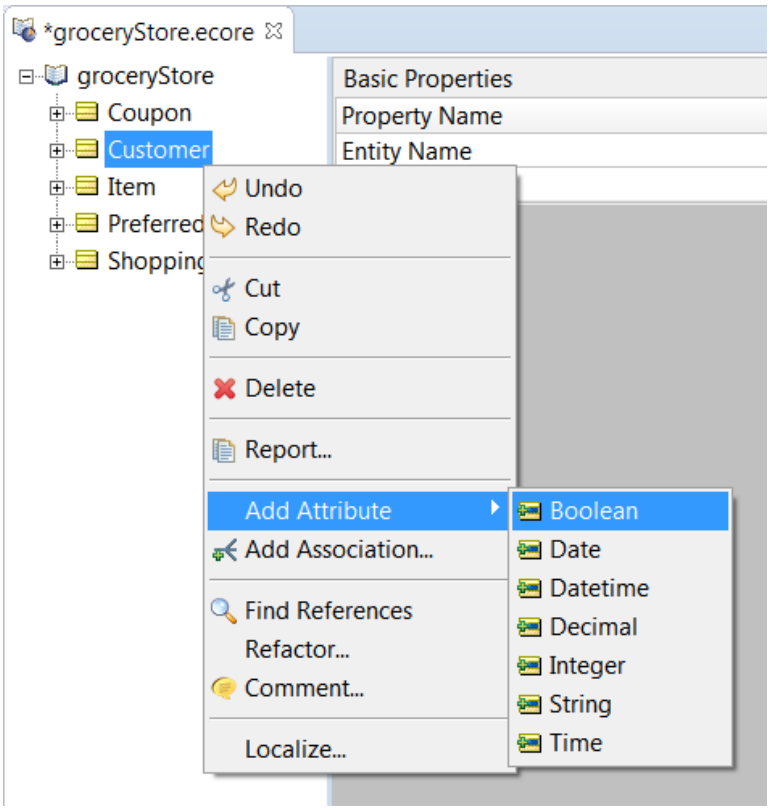
1. Right-click Customer and select Add Attribute, and then choose String.



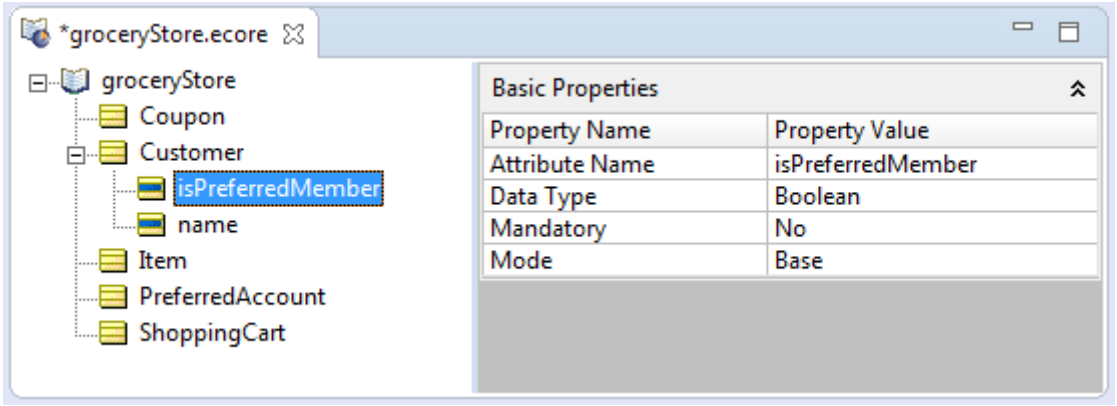
2. An attribute named attribute_1 is created under Customer. Type over this and enter name.



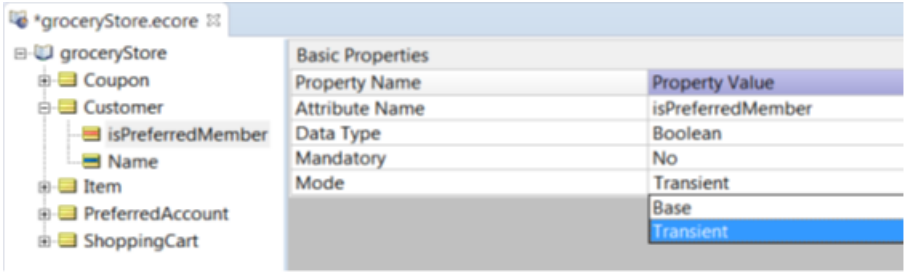
3. Right-click Customer, select Add Attribute, and then choose Boolean



4. Type over the default attribute name and enter isPreferredMember.



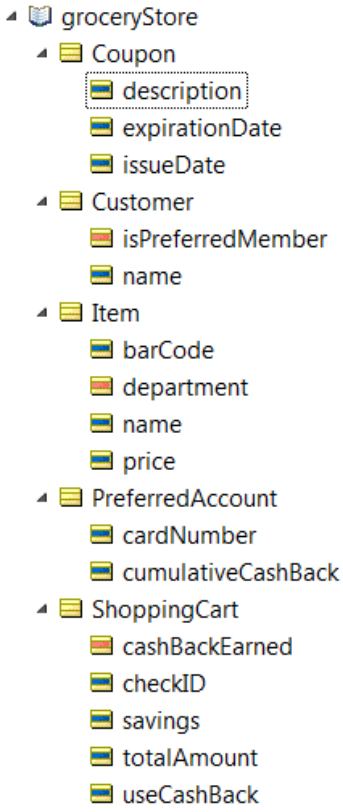
5. Click the Mode drop-down and select Transient.



Now, add attributes for the rest of the entities based on this table:

Entity	Attributes	Data Type	Attribute Mode
Item			
	name	String	Base
	price	Decimal	Base
	department	String	Transient
	barCode	String	Base
ShoppingCart			
	totalAmount	Decimal	Base
	cashBackEarned	Decimal	Transient
	savings	Decimal	Base
	useCashBack	Boolean	Base
	checkID	Boolean	Base
PreferredAccount			
	cardNumber	String	Base
	cumulativeCashBack	Decimal	Base
Coupon			
	issueDate	Date	Base
	description	String	Base
	expirationDate	Date	Base

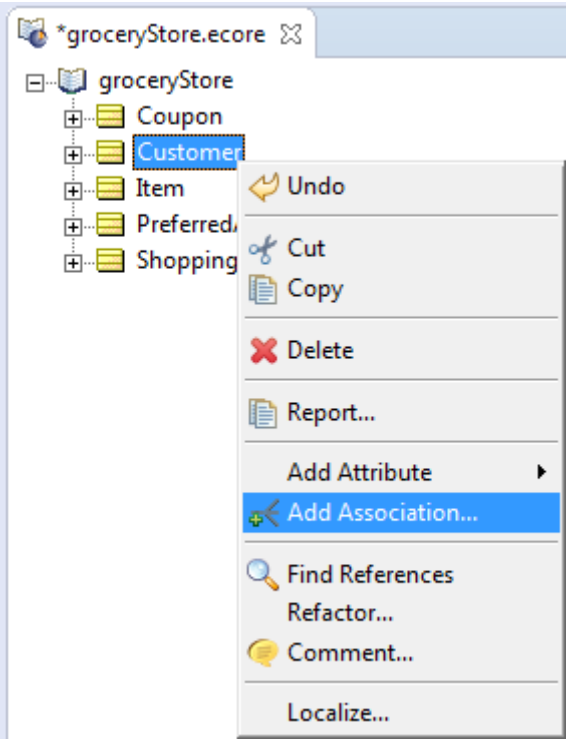
After adding all the attributes, the Vocabulary looks like this:



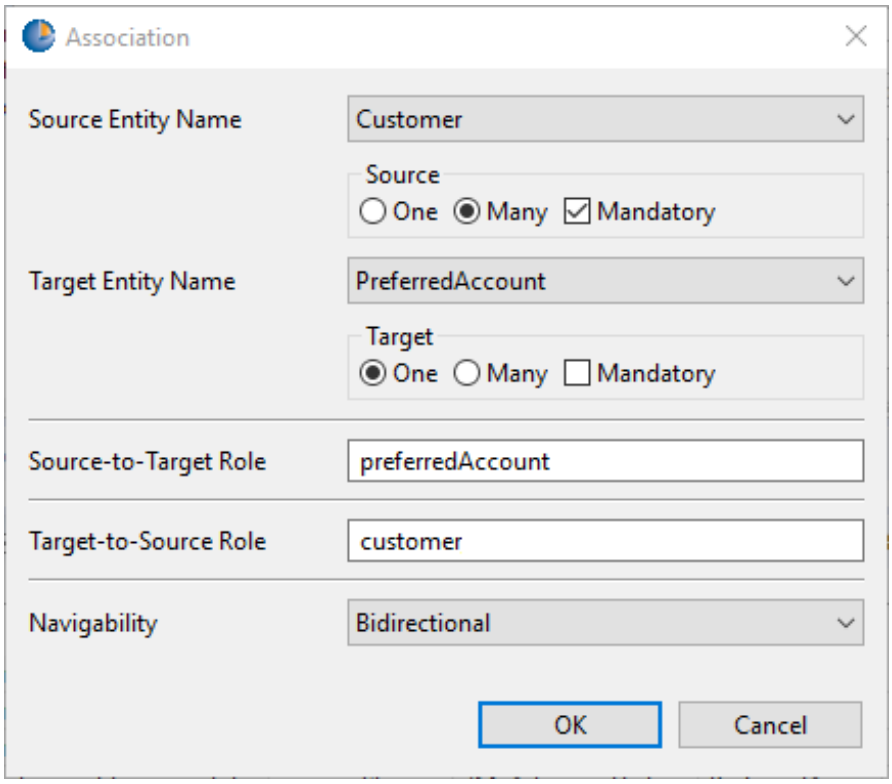
Add Associations

Next, let's create associations between the entities. Let's start with the association between Customer and PreferredAccount. This is a many-to-one association.

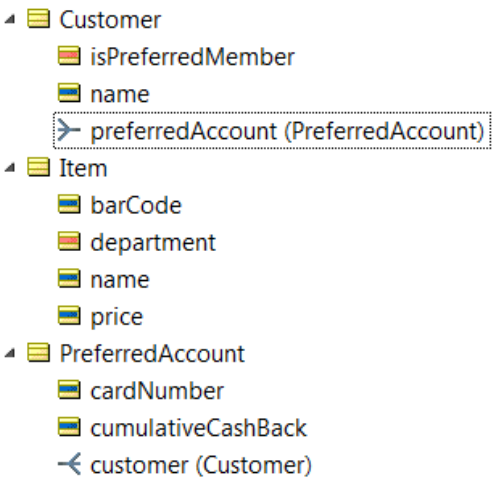
1. Right-click Customer and select Add Association.



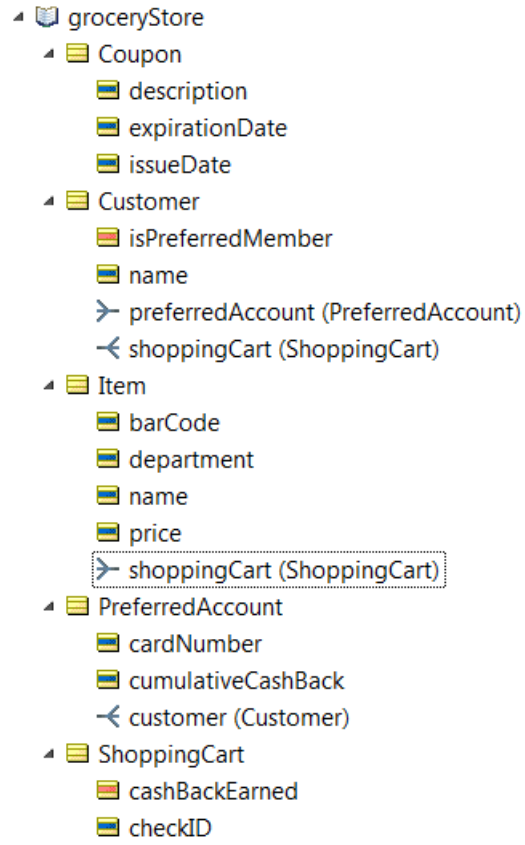
2. In the Association dialog box:
   a. Select Many in the Source section
   b. Select PreferredAccount as the Target Entity Name
   c. Select One in the Target section
   d. Click OK



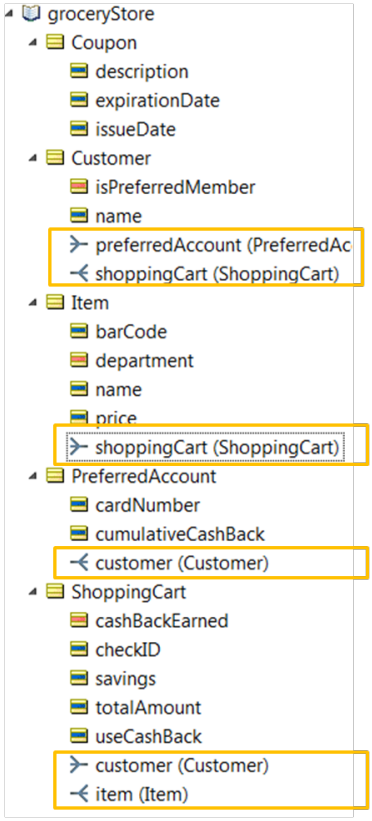
The association appears as shown here.



Notice that the association shows up as many-to-one (⊢) under Customer and one-to-many (⊣) under PreferredAccount.

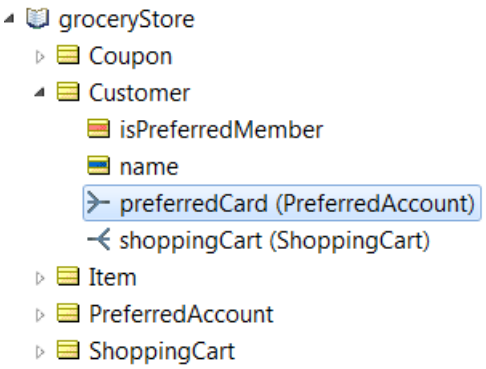


Similarly add associations between Customer and ShoppingCart (one-to-many) and between Item and ShoppingCart (many-to-one). The final output will look like this.



Each association has an association role name. For example, the association between Customer and PreferredAccount is given the name preferredAccount. Note that the opposite association between PreferredAccount and Customer is given the role name customer. A role name helps describe or clarify the relationship of an entity with another entity.

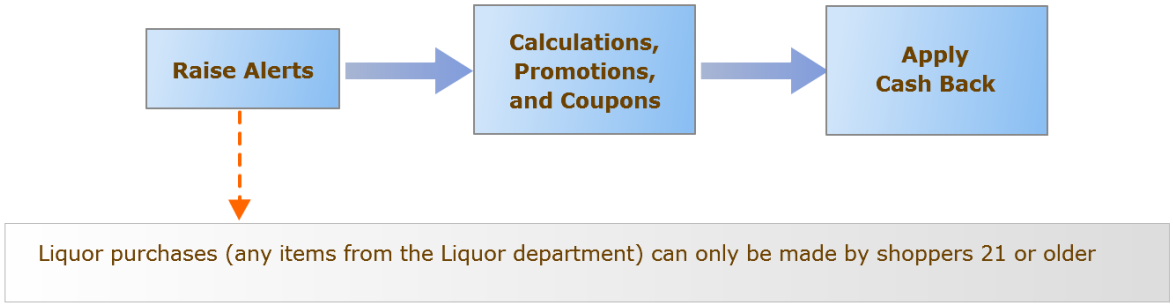
You can change the role name for an association to make it more meaningful. In our example, let's change the role name for the association between Customer and PreferredAccount to preferredCard. To do this, double-click the association under Customer, type over the default value, and enter preferredCard.



4

Model the first Rulesheet

With our Vocabulary ready we can now focus on modeling the rules. Let's begin with the first Rulesheet, which will model the rules in the Raise Alerts substep.



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

The business rule we identified for the Raise Alerts substep examines all items in a customer's shopping cart and determines which items (if any) come from the Liquor department.

As you may remember, each Item has a barcode. The department code occupies the 4th thru 6th characters of the barcode. Let's assume that the department code for liquor is 291. So if an item has 291 occupying the 4th through 6th characters of the barcode, the cashier must be alerted to check the customer's identification.

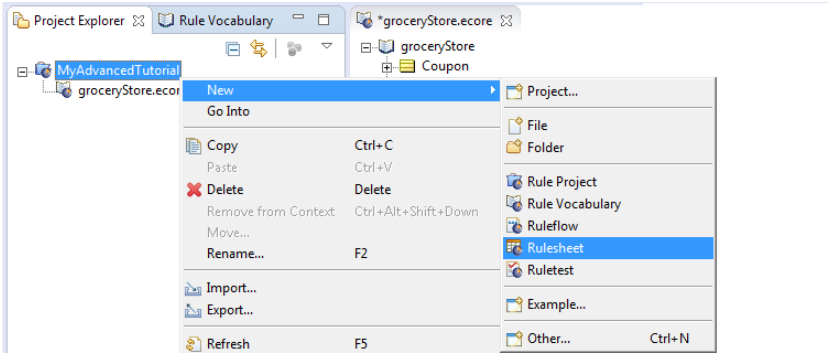
Keeping this in mind, we will define two rules in Corticon Studio:

- The first will determine the department code for every item in the shopping cart
- The second will determine if any of the items come from the Liquor department and if so, the rule will raise an alert of some kind.

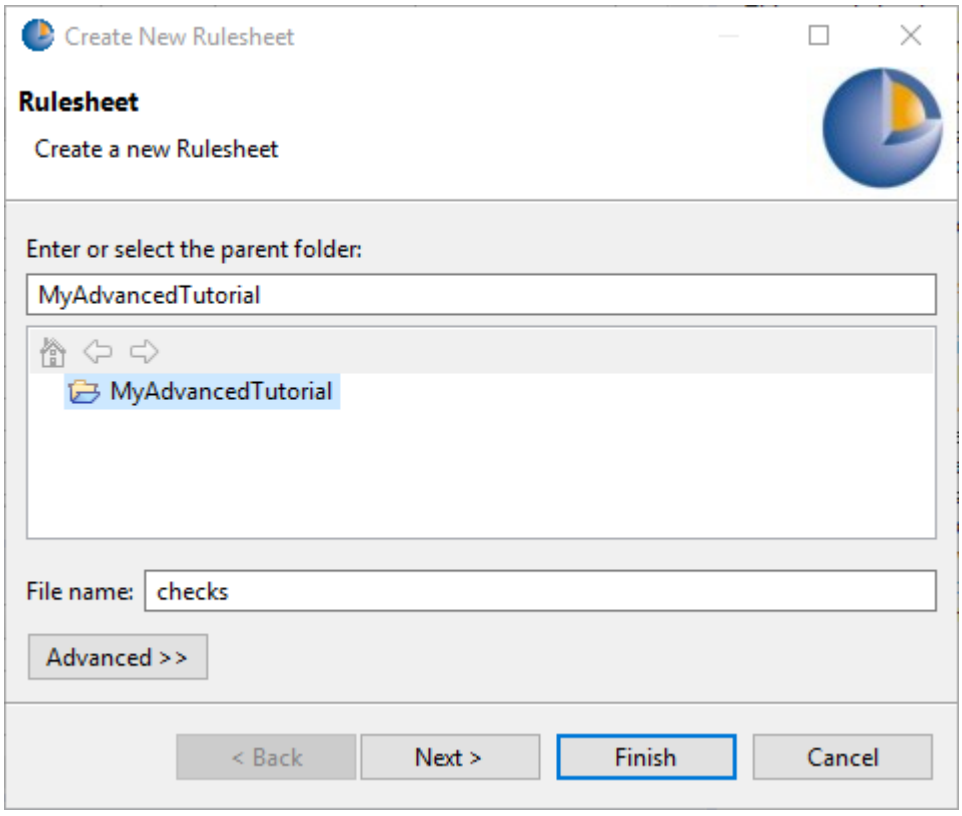
As you can see, there isn't always a one-to-one correlation between the business rule defined in a business scenario and the corresponding rules modeled in Corticon Studio. This is normal. A good guideline is to keep your individual rules relatively simple and let them work together to perform more complex logic defined by the business rules.

Create the Rulesheet

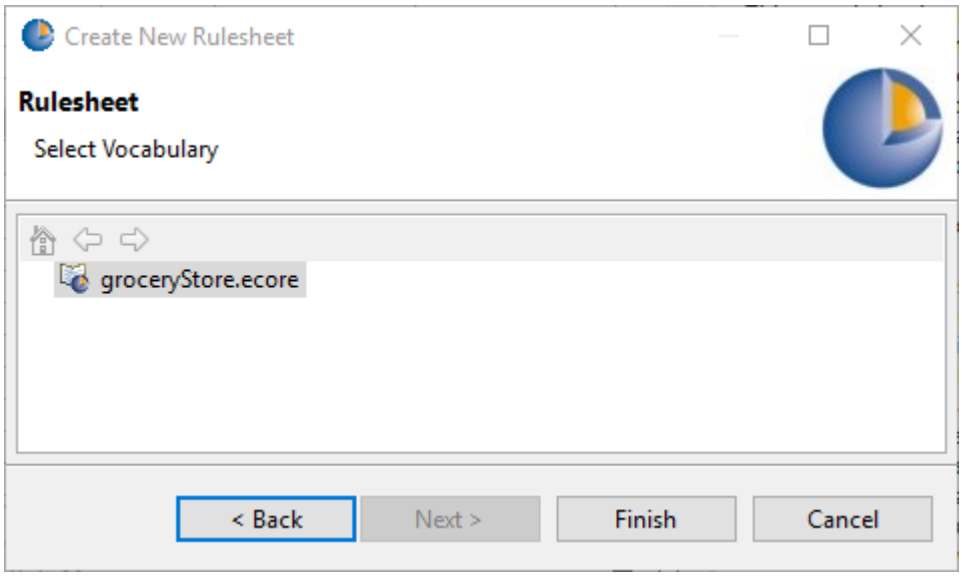
Before we start defining the rules, we need to create a Rulesheet.



Name the Rulesheet checks, and then click Next.



Click groceryStore.ecore as the Vocabulary.

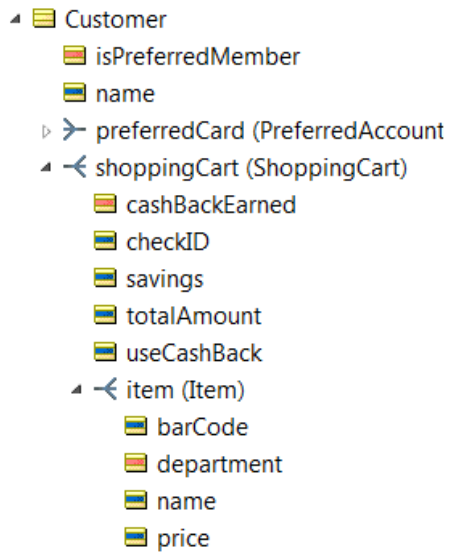


We've named this Rulesheet checks as a way of reminding ourselves of the overall organization—this Rulesheet will perform any necessary checks and raise alerts as required. Click Finish.

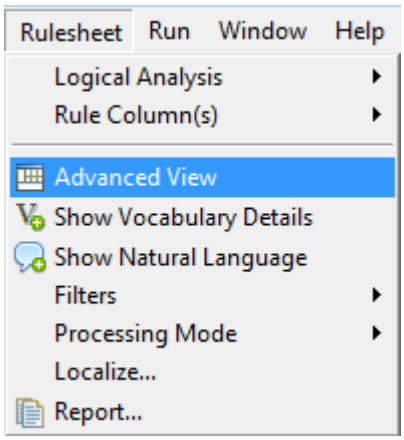
Define rule scope

We need to choose the "point of view" in the Vocabulary that best represents the terms required by the rules themselves. This is called the scope of the rule. The scope changes from Rulesheet to Rulesheet.

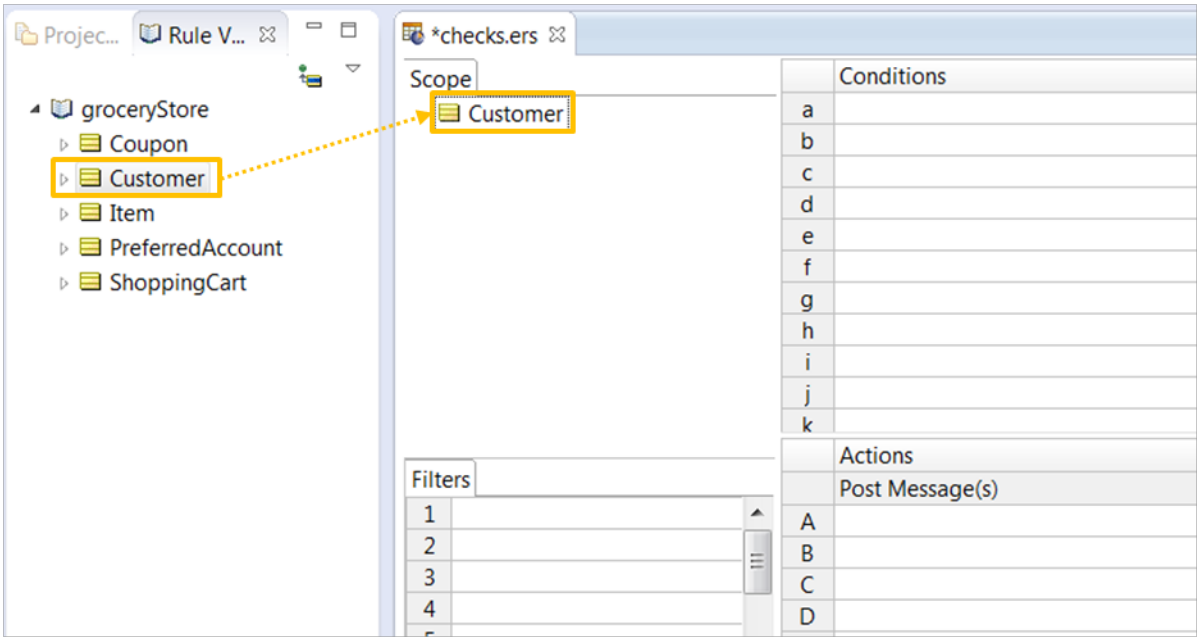
Scope is a powerful and important concept. Scope determines which entity instances and attributes are evaluated and acted upon by a rule. For the first Rulesheet, we want to define rules that act only on Items that are associated with a ShoppingCart, which in turn is associated with a Customer. Using 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 step. This will form the scope of the rules in the Rulesheet.



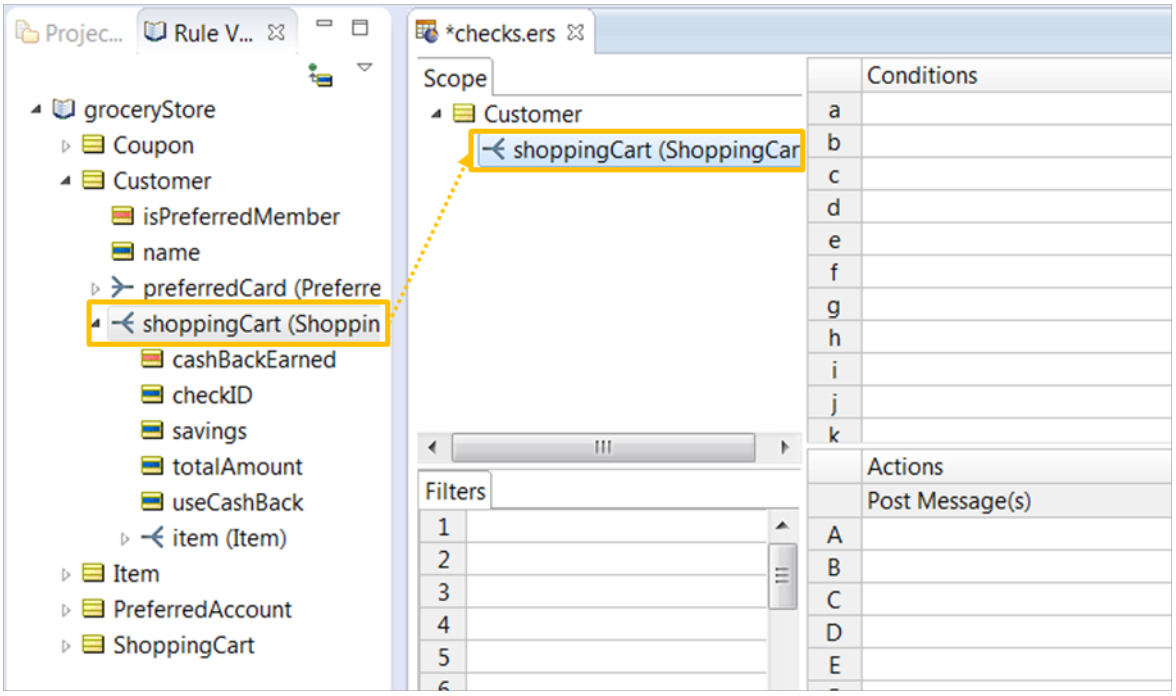
Let's define the scope. Display the Scope pane of the Rulesheet by ensuring that checks.ers is open and active and selecting Rulesheet > Advanced View.



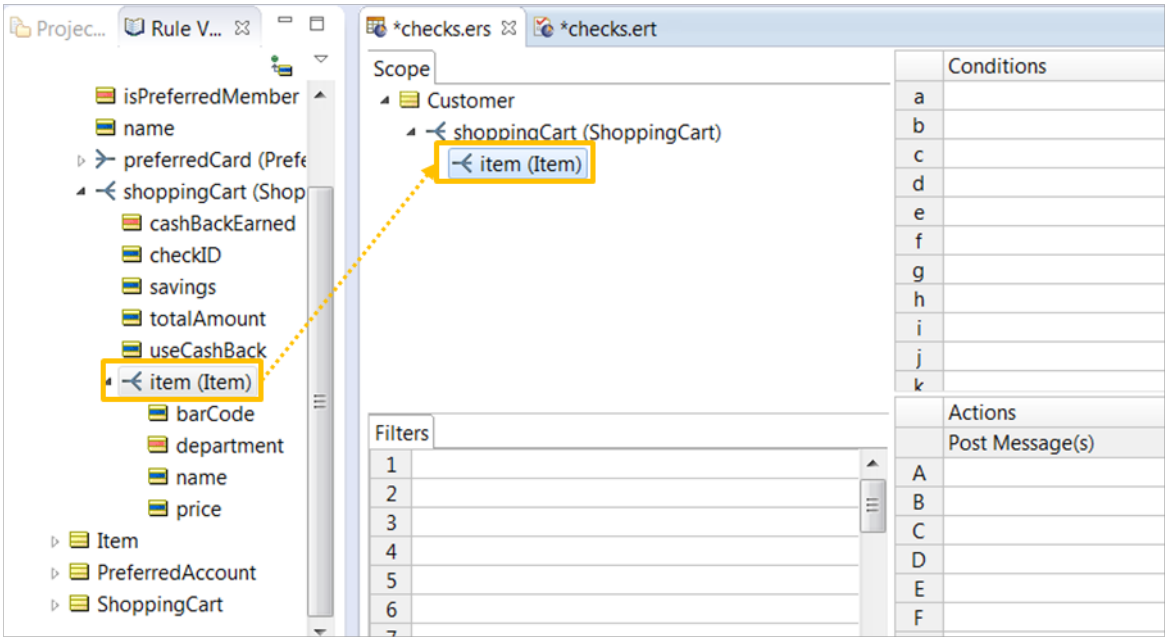
The Scope pane opens in the Rulesheet. Drag and drop the Customer entity from the Rule Vocabulary view into the Scope pane.



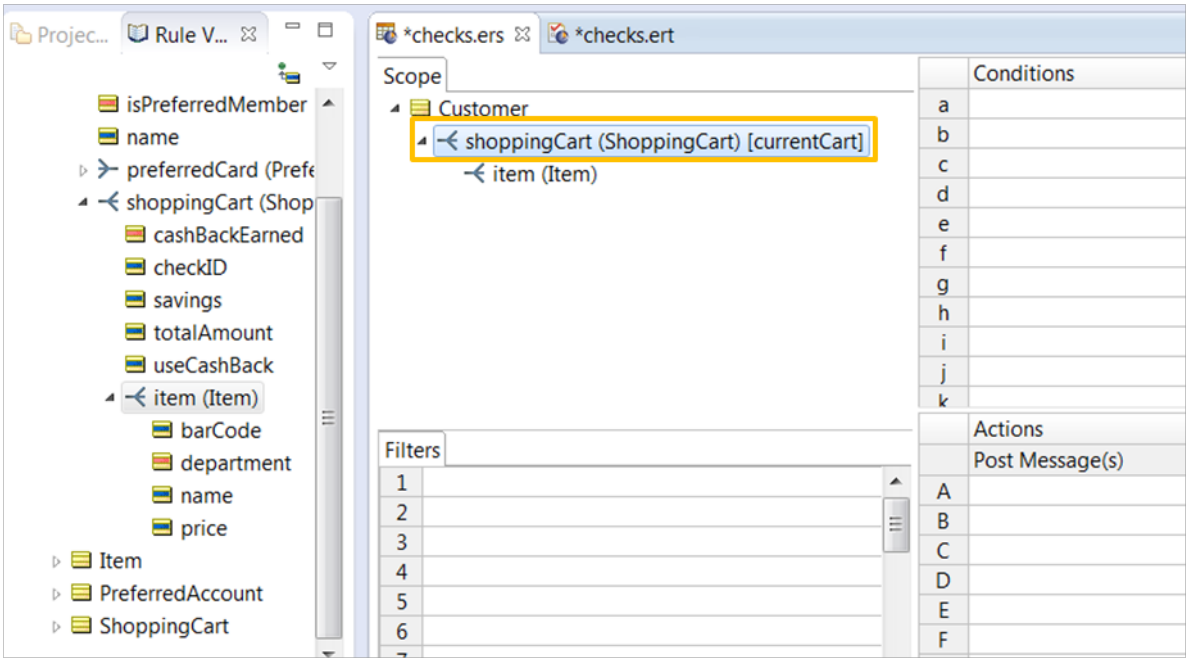
Drag and drop shoppingCart (under Customer) into the Scope pane.



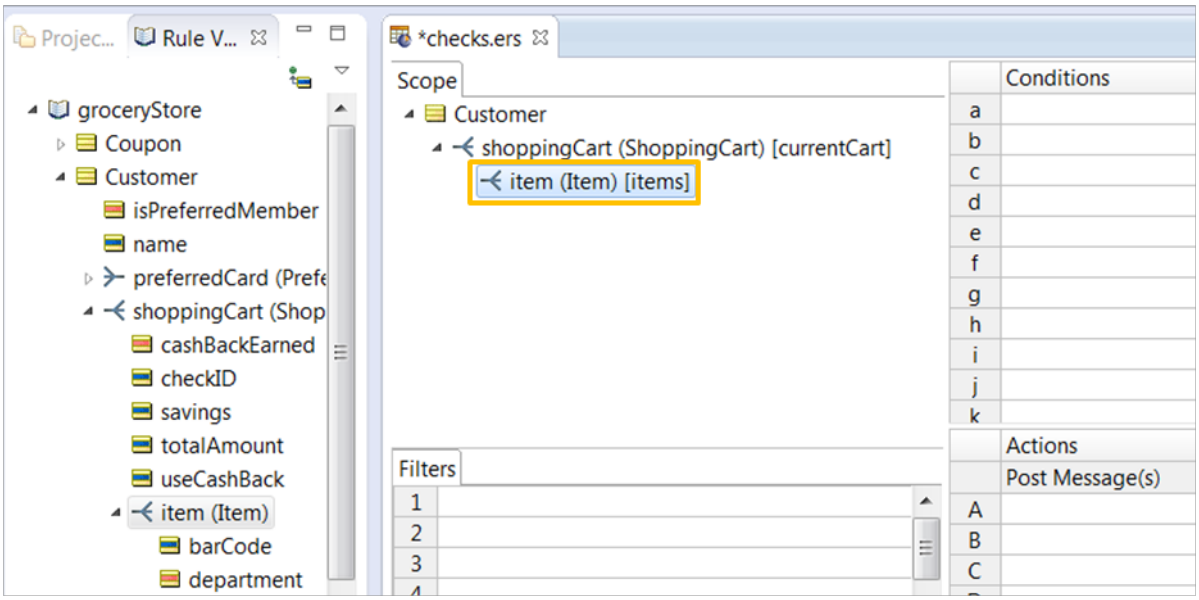
To complete the Scope, drag and drop item (under shoppingCart) into the Scope pane.



Next, let's enter an Alias for a customer's shopping cart and call it currentCart. To do this, double-click shoppingCart in the Scope pane and enter currentCart. From now on, when we model rules involving a customer's shopping cart, we'll use the Alias currentCart to represent the perspective of a customer's shopping cart.



We know that a shopping cart can contain many items. Let's define another Alias that represents all the items in a customer's shopping cart. Give it the Alias items.



Assigning meaningful alias names is a good practice and using the plural form of item reminds us that the Alias represents all 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. For example, 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.

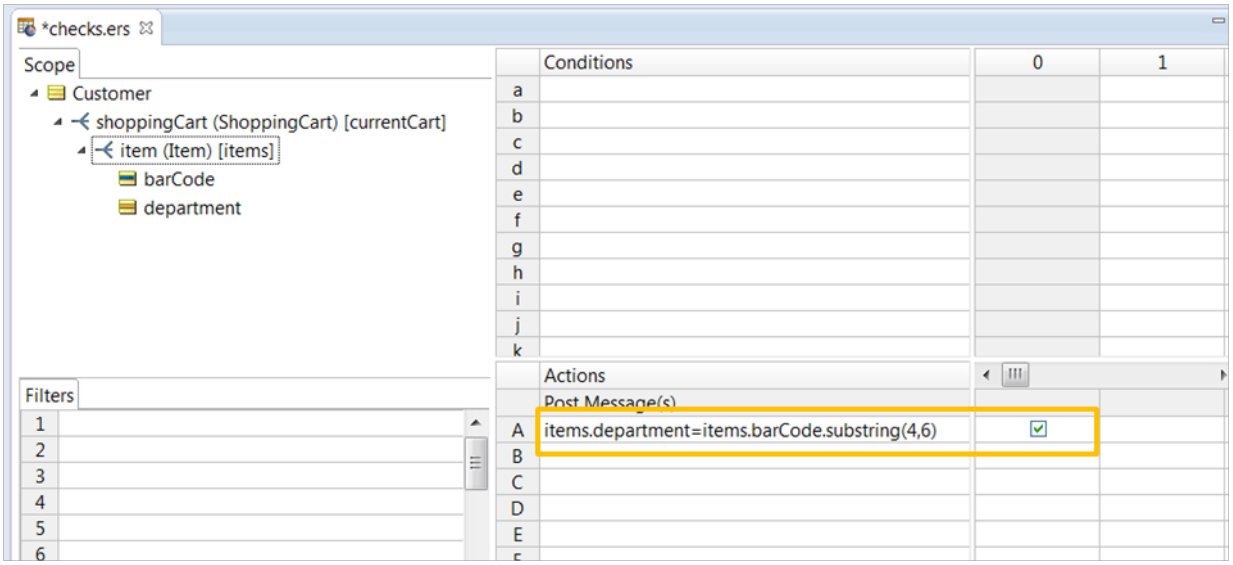
Aliases will always insert themselves automatically when terms are dragged and dropped from the Scope section or Vocabulary window to the Rulesheet. Since all Studio expressions are case-sensitive, it's better to drag and drop terms instead of typing them manually, which can cause errors.

Model the first rule

We decided earlier that in order to model the business rule for the Raise Alerts substep, we need to create two rules in Corticon Studio in the checks Rulesheet:

- Rule 1: Determine the department code for every item in the shopping cart.
- Rule 2: Use the department code to determine if any of the items come from the Liquor department and if so, raise an alert.

Let's start modeling the first rule—determining the department codes. We know an item's department is identified by the 4th through 6th characters in the item's barCode. So using the items alias, let's add an action-only rule in an Actions row of Column 0 using the String operator .substring as shown.



The expression items.department=items.barCode.substring(4,6) extracts the characters from positions 4 to 6 in the barcode string and assigns the substring to items.department. The checkbox in the corresponding cell of Column 0 indicates that this is an action-only rule. The rule fires whenever any data is received by the Rulesheet. Action-only rules fire first in the Rulesheet. In our example, this is useful because we need to extract the department code first before we can identify if any items come from the Liquor department.

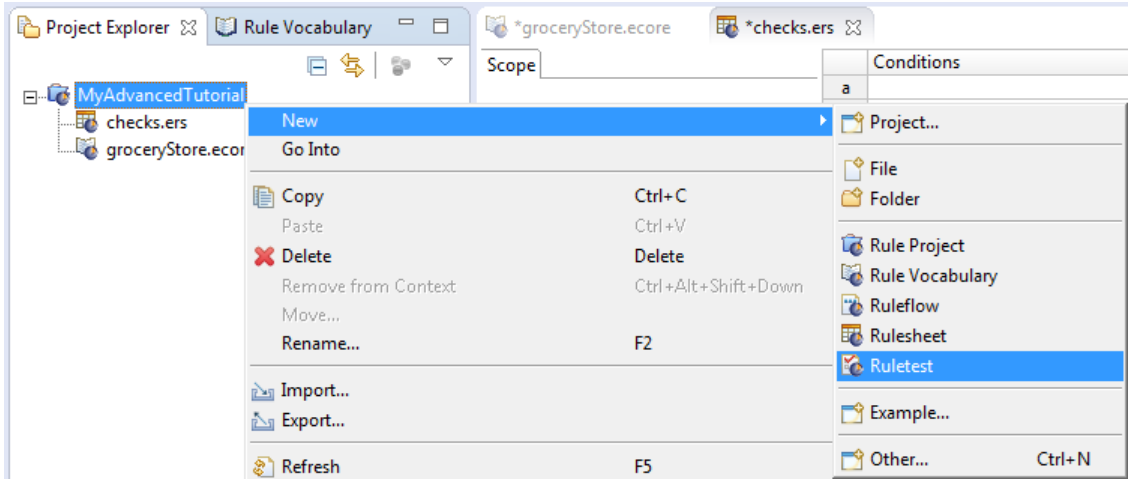
Remember that the alias items represents the collection of all items associated with a customer's shopping cart. So this rule will evaluate and process every item in a customer's shopping cart, extract the department code for each, and then assign that code to the item's department attribute. This iteration is a natural behavior of the rule engine: it will automatically process all data that matches the rule's scope.

Notice that when you dragged the terms barcode and department from the Vocabulary to the Action row, they were automatically added to the Scope pane. Over time, the Scope pane becomes a reduced version of our Vocabulary, containing only those terms used by the rules in that Rulesheet.

Save the Rulesheet.

Test the first rule

Next, let's test our first rule. Create a Ruletest

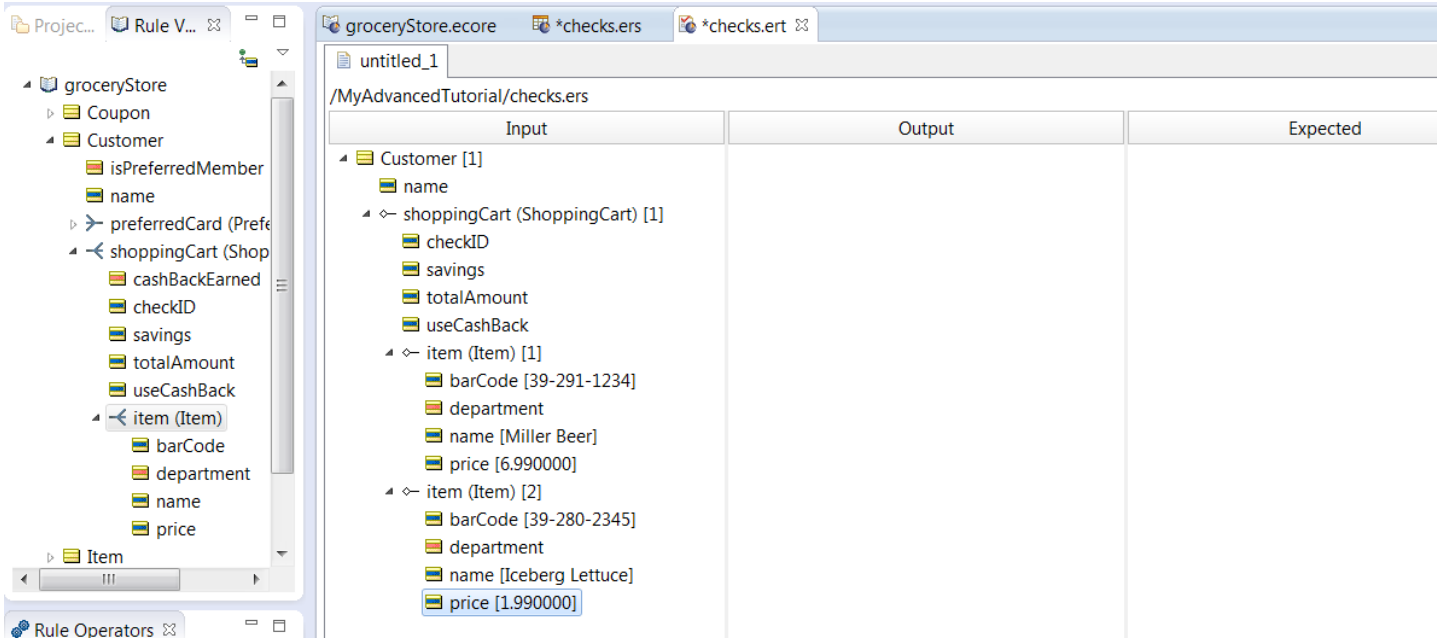


Name the Ruletest checks, and then choose the test subject checks.ers.

In the Input pane of the Ruletest, define a customer with an associated shopping cart containing two items—one of which is from the Liquor department.

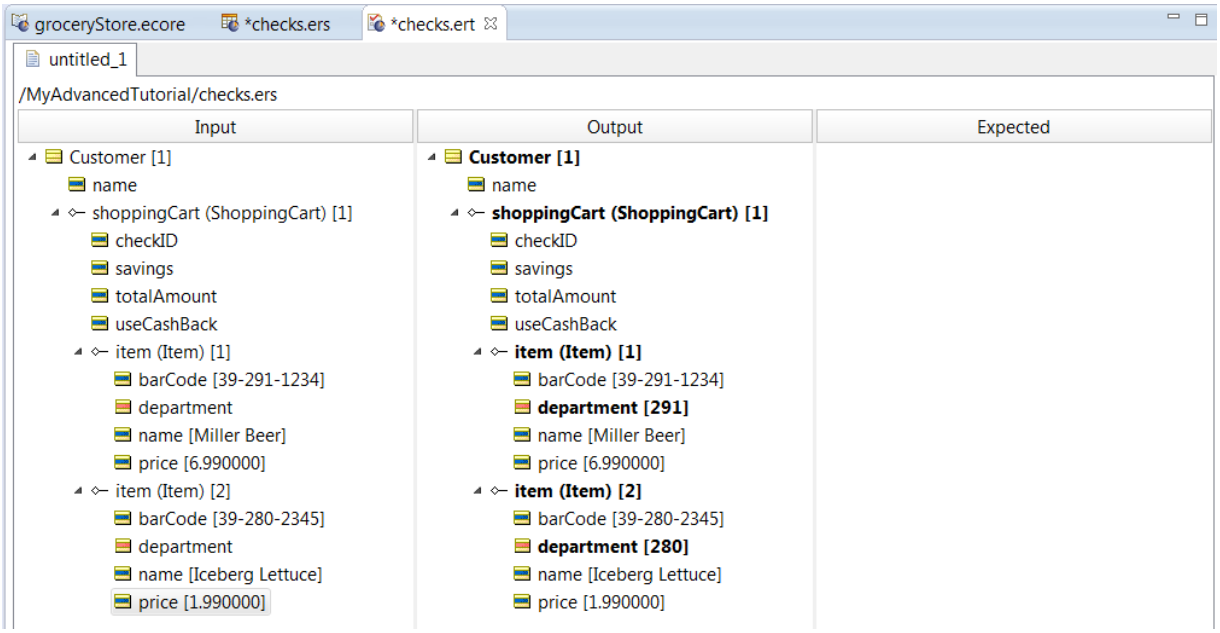
First, drag and drop the Customer entity into the Input panel. Then, drop shoppingCart (under Customer) onto the Customer entity. Finally, drag and drop item (under shoppingCart) onto the shoppingCart entity twice. Note: It is critical to drop the items from the Vocabulary into the Input panel of the Ruletest in the order indicated so that we duplicate the Scope of the rule which will be processing this data.

When finished, enter test data as shown.



As you can see, one of the items is from the Liquor department (remember that the department code for Liquor is 291 and this occupies the 4th to 6th characters in the barcode).

Finally, execute the Ruletest. The output should look like this.



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.

By modeling a rule and then immediately testing it, we have demonstrated a good Studio modeling practice. Testing right away will help expose flaws in our rules as we go along.

Model the second rule

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

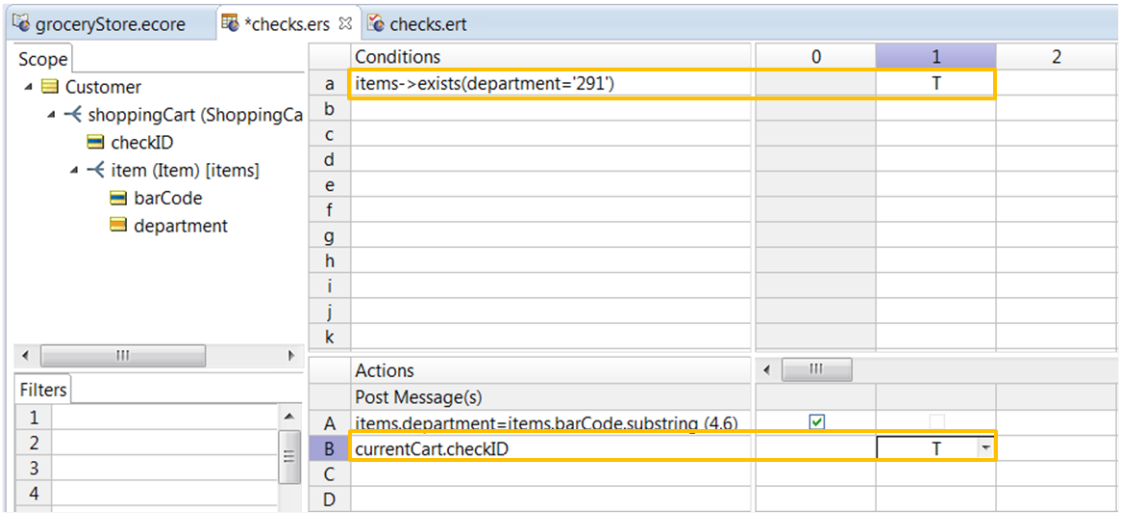
This requires us to "look inside" our Collection of Items to see if there exists an item with department = '291'. Since we only need one "check ID" alert per checkout transaction, this is a job for a collection operator.

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 only need one "check ID" alert if the shopping cart contains any liquor. We don't need, say, 5 alerts if the shopping cart contains 5 liquor items.

Making use of the items alias, let's add a Condition for the second rule that determines if any Liquor items exist in the customer's shopping cart. To do this, we use the Collection operator exists. The exists operator checks if a specific value exists for an attribute in the entity instances in the collection. In this case the collection is items. So the condition expression is items->exists(department='291'). Note: Always use plain single-quote marks to specify a text string.

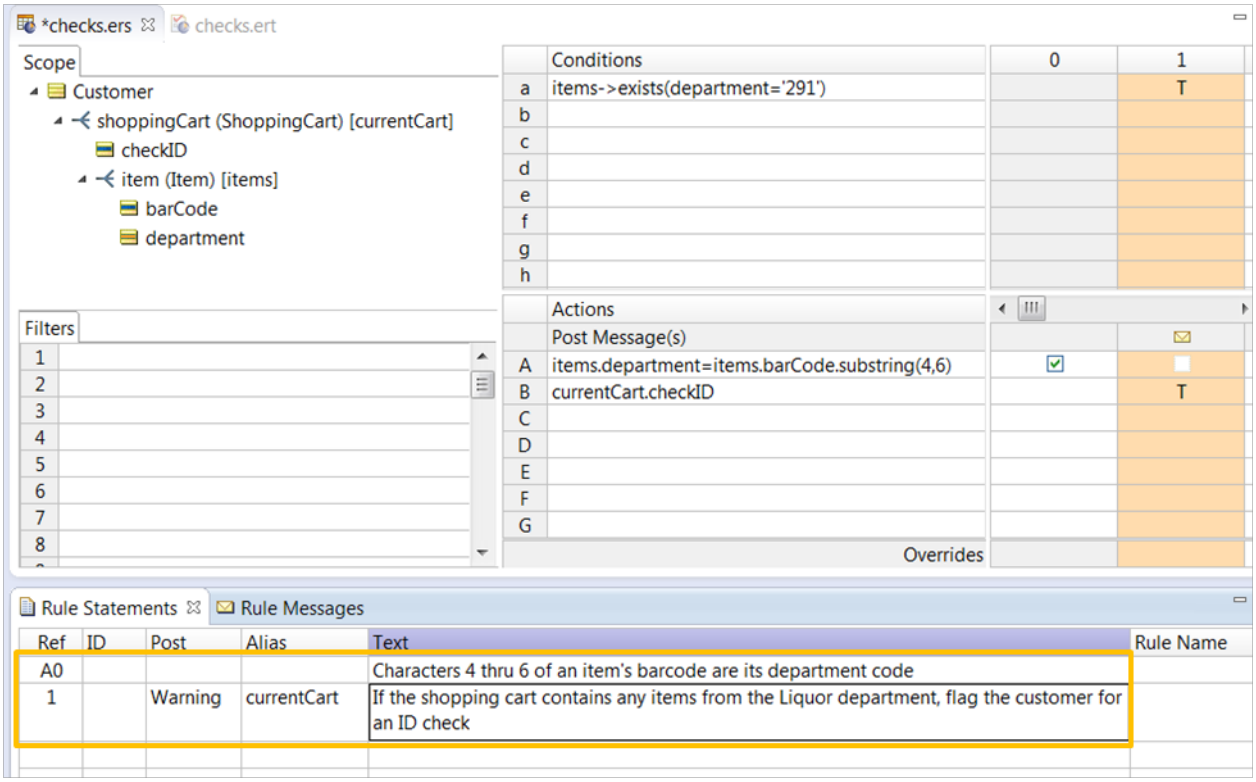
Then define an Action for the second rule to assign a value of true to the shopping cart's checkID attribute if any are found. (We're assuming that the checkID term will act as the alerting mechanism to signal the cashier that an ID check is required during this checkout transaction.)

The second rule will look like this:



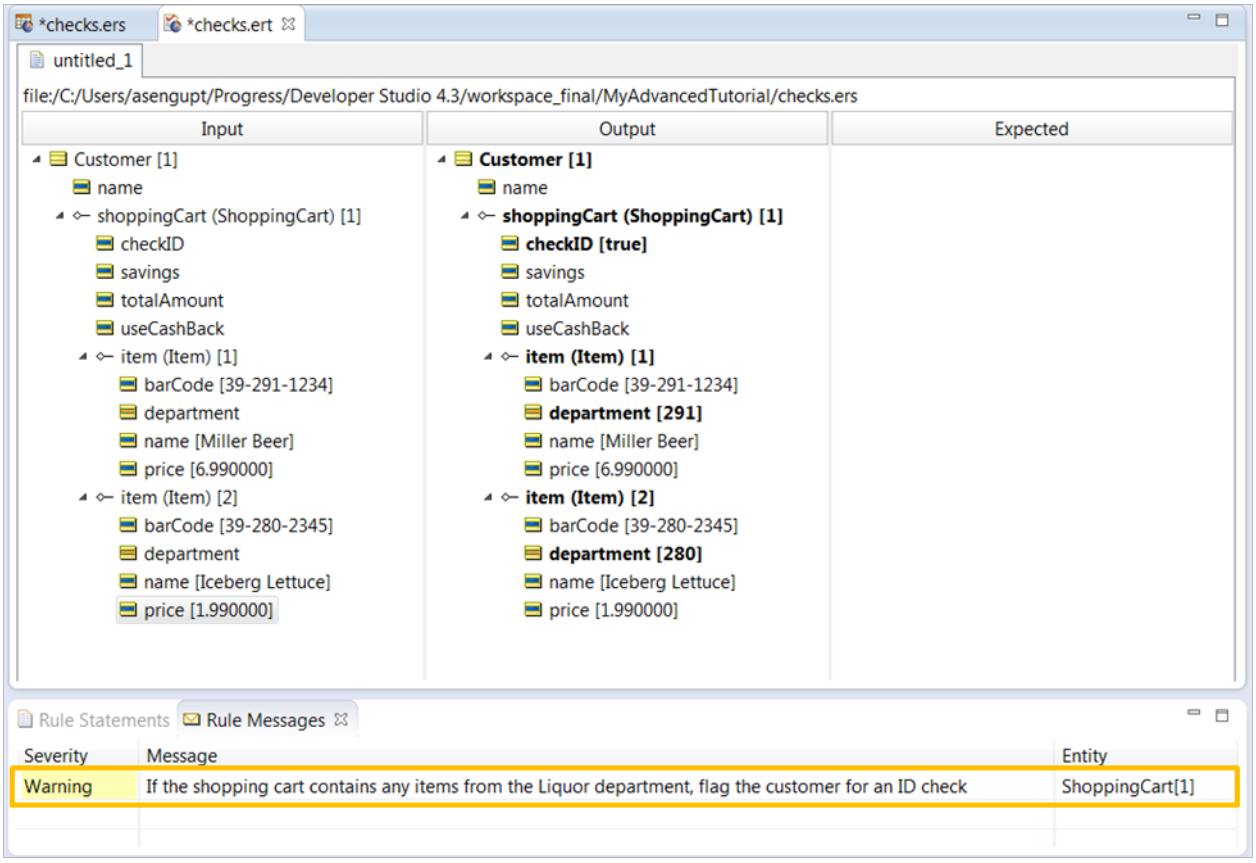
As mentioned earlier, using aliases to represent collections is mandatory when collection operators (like exists) are used.

Add rule statements for each rule as shown:



Test the second rule

Now, let's re-run the same Ruletest as before:



As you can see, 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.

Note: Ordinarily, we'd check for Conflicts and Completeness before testing with data. But since we are focusing on advanced rule modeling features in this Tutorial we are skipping the Analyze phase of the rule development lifecycle.

Add more rules to the checks Rulesheet

We have implemented two rules representing the first business rule in the checks Rulesheet. We will use this Rulesheet to model two more rules:

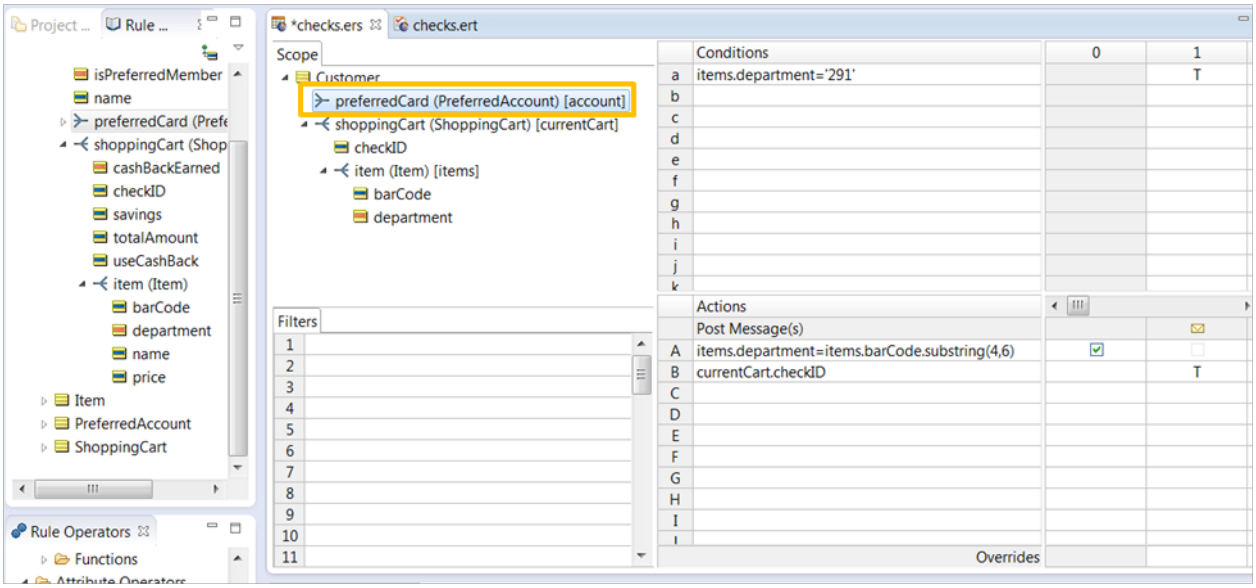
- Check if a customer has a preferred account
- Add the total price of items in a shopping cart

We will use the output of the first rule in the next Rulesheet to filter out customers who do not have preferred accounts, since it contains promotional rules that apply only to preferred account holders.

The second rule, that calculates total price, must be included in the first Rulesheet, since the second Rulesheet filters out customers who don't have preferred accounts, and we want to calculate the total price for every customer.

Check if a customer has a preferred account

Any customer who has a Preferred Account will have an associated preferredCard. So, let's begin by defining the scope. Drag and drop preferredCard under Customer in the Rule Vocabulary to the Scope pane. Give it the Alias account.



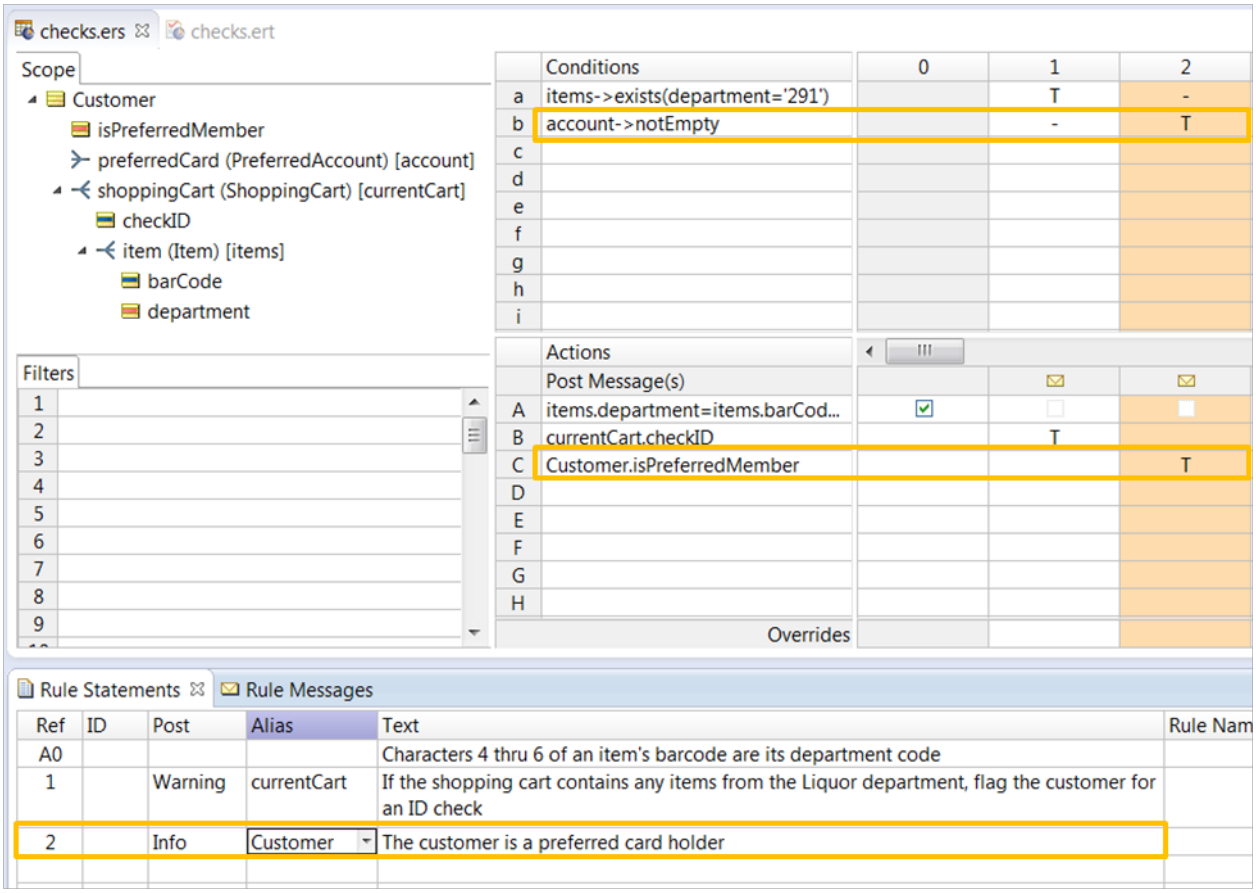
The account alias represents a “potential collection”, that is, a customer will have a Preferred Card only if they have a Preferred Account. And the “many-to-one” nature of the association means a customer will have at most one account. Other customers (as with a family) may share the same Preferred Account. For Customers who don’t have Preferred Accounts, the alias account represents an empty collection (the collection contains no elements).

The notEmpty collection operator checks a collection for the existence of at least one element in the set. We can use this operator to check if a customer has a preferred card. Because notEmpty is “acting on” a collection, the account alias must be used with it.

Model a Boolean condition in row b in the Conditions pane that uses the notEmpty collection operator as shown. If the account alias is not empty, we know the customer has a preferred account.

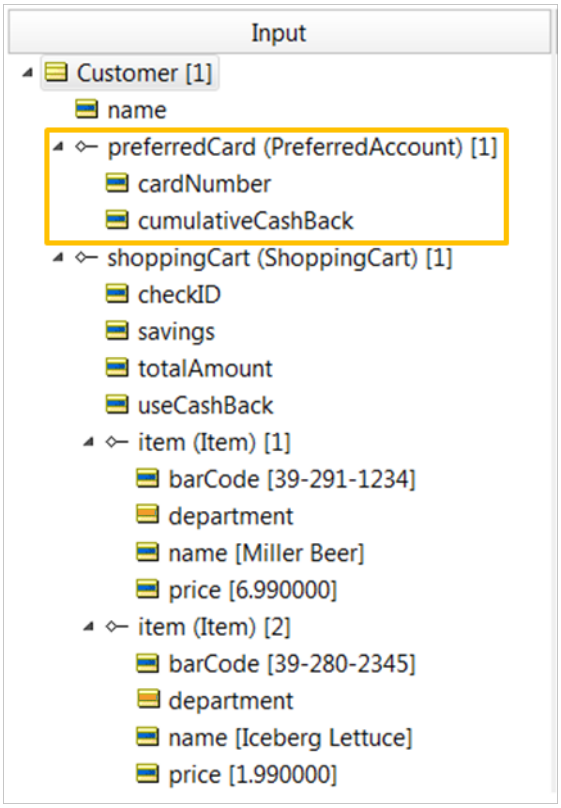
Conditions	0	1	2
items->exists(department='291')		T	
account->notEmpty			T

Let’s add an action to this rule that assigns the value of true to the isPreferredMember attribute (from the Customer entity) and posts an informational message as shown. Recall that isPreferredMember is a transient attribute. Its value will only be used in the rules.

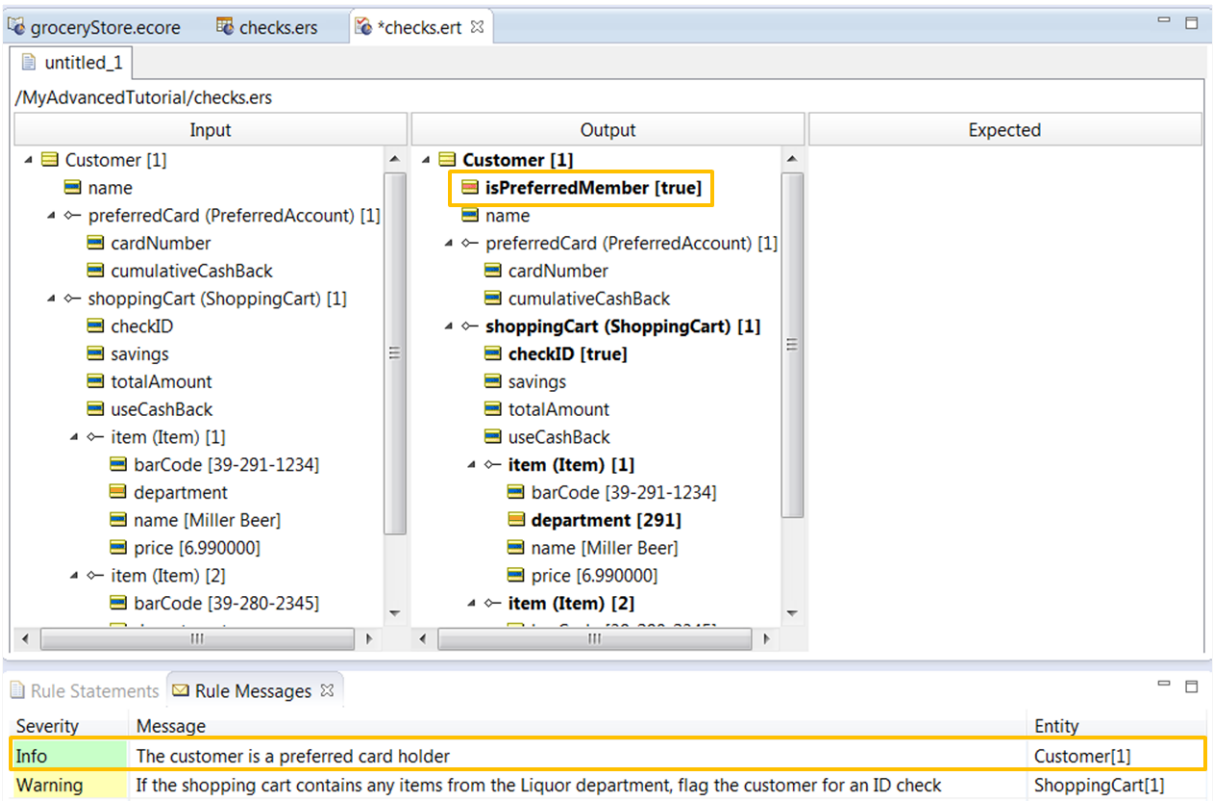


Now, whenever we need to know if a customer is a preferred customer, we simply refer to the value of the isPreferredMember attribute. This method of “flagging” an entity with a Boolean attribute (also known as a “flag”) is convenient when modeling larger Ruleflows. The value of the flag, like all attributes, will carry over to other rules in this and other Rulesheets in the same Ruleflow.

Now, let’s test this rule. For our rule to detect the presence of a Preferred Card account associated with a customer, we need to provide the appropriate test data. Drag and drop the preferredCard entity onto the Customer entity in the Ruletest Input. Note: If you don’t get the identical indented structure as shown, delete the entity and try again.



Run the Ruletest.

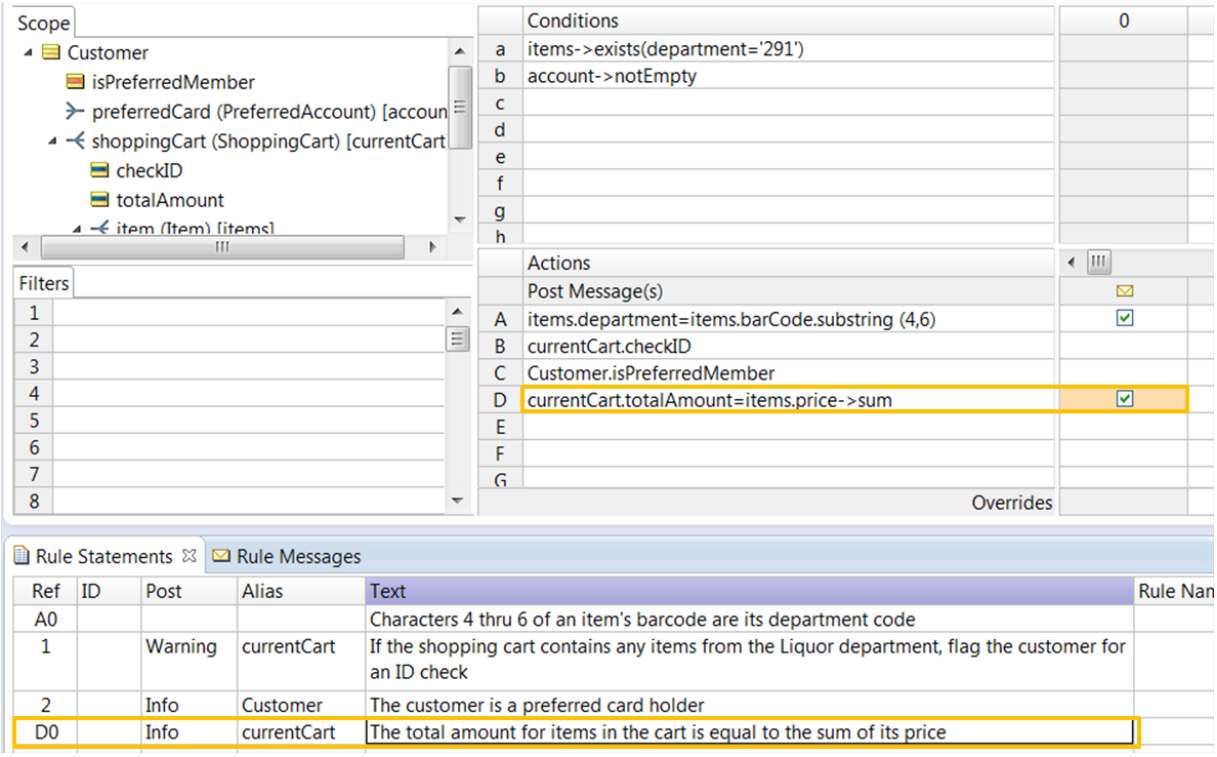


Notice that the isPreferredMember transient attribute has been inserted and assigned a value of true, and that an informational message has been posted. Our rule has worked as expected.

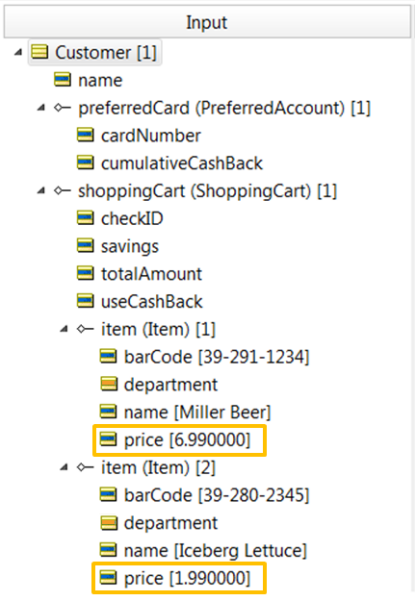
Calculate the total price of items in a shopping cart

Finally, we'll add one more action-only rule to calculate the totalAmount of all items in a customer's shopping cart. To do this, use the collection operator sum as shown. The sum operator will add up the price attributes of all elements in the items alias, and then assign that value to the totalAmount attribute.

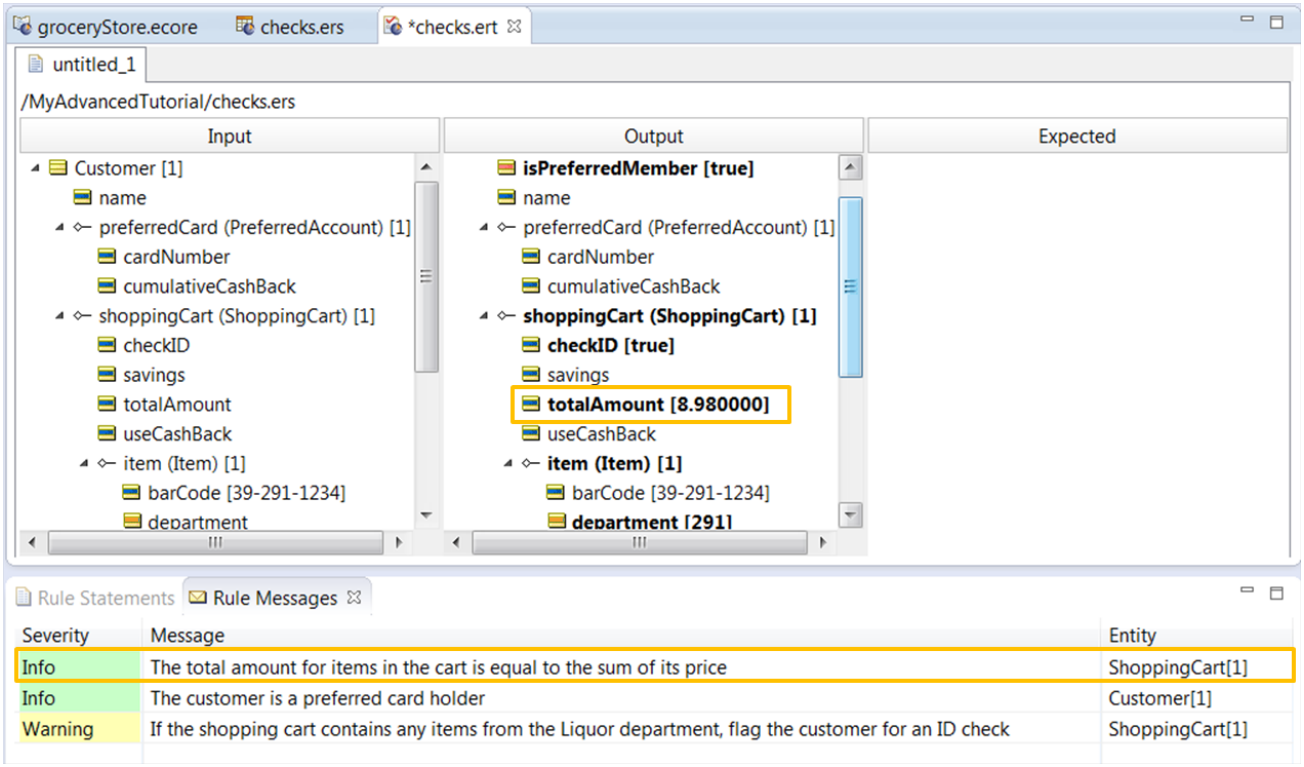
Now, add a rule statement for this rule. Adding rule statements is good practice, even if you don't post them as messages.



Finally, let's test this rule. In the Input pane of the Ruletest shown here, we have a customer with two items in their shopping cart. Let's see if our last rule calculates the totalAmount for the items in the Customer's shopping cart.



Run the Ruletest.



A lot has happened in this Ruletest. First, note that our rules to determine if an ID check is required and 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. This shows that our latest rule also works as expected.

We have now completed modeling and testing our first Rulesheet.

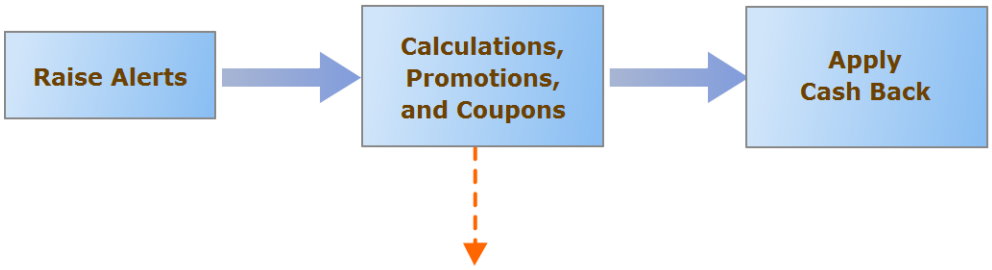
5

Model the second Rulesheet

Let’s take a quick look at what we did in the first Rulesheet (checks.ers):

- We defined an action-only rule to extract the department code from the barcode of each item.
- We defined a rule to identify if there are any items from the Liquor department, and if so, to raise an alert.
- We defined a rule to check if a customer is a preferred card holder.
- Finally, we defined another action-only rule to calculate the total price of items in the shopping cart.

Now, we are ready to model the second Rulesheet. Recall that the second Rulesheet corresponds to the second substep in the Checkout process.



- Preferred Shoppers earn 2% cash back on all purchases at any branch.
- Cash back earned by preferred shoppers should be added to the cumulative cash back in their preferred shopper account.
- 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.

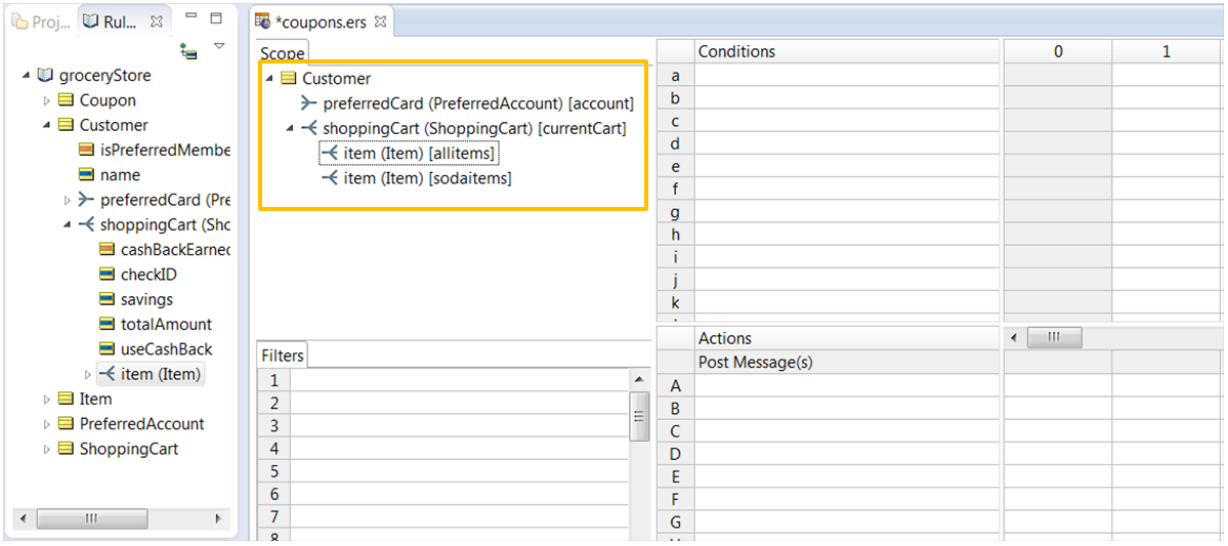
In the second Rulesheet, we apply some promotional rules to our Preferred Account holders when they spend a pre-defined amount of money or buy items from specific departments at our store. The promotions may change frequently, but modeling them in Corticon will make future changes much easier.

Create the Rulesheet

Create a second Rulesheet under the MyAdvancedTutorial rule project. Name it coupons. Ensure that it uses the groceryStore.ecore Vocabulary.

Define the rule scope

Next, let's define the rule scope. Open the Scope pane, by selecting Rulesheet > Advanced View. Just like in the checks.ers Rulesheet, we will build our Scope around the Customer entity, since we want to apply promotional rules to each preferred customer. Define the rule Scope as shown:



Notice that once again, you assigned the alias currentCart to a customer's shopping cart, just like in the checks.ers Rulesheet. In addition, you have created two new aliases to define the currentCart.item perspective of the data. For now, the two aliases (allItems and sodaItems) represent the same perspective, but we'll differentiate between them shortly. Finally, as before, the account alias represents the preferredCard account associated with our customer.

Save the Rulesheet.

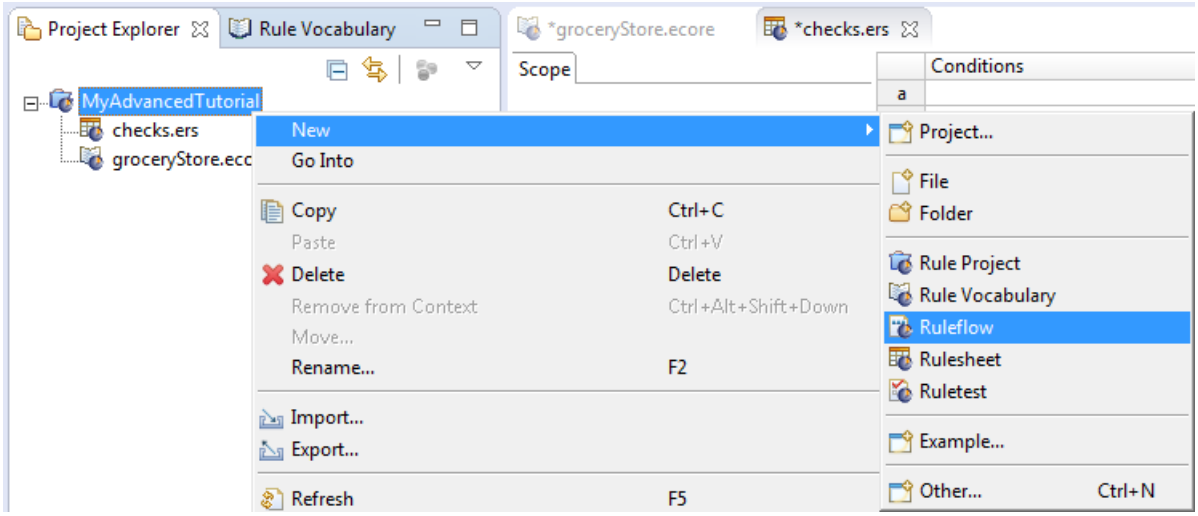
Create a Ruleflow

Before we define rules in the coupon.ers Rulesheet, let's create a Ruleflow and add the checks.ers and the coupons.ers Rulesheets to it. 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.

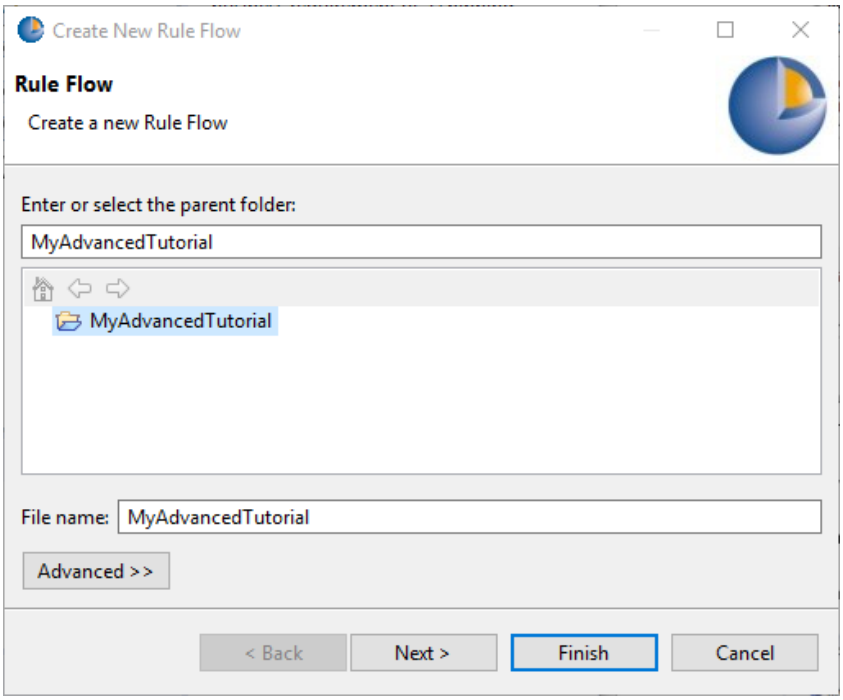
Creating a Ruleflow at this point is a good practice. Instead of testing only the Rulesheet as we develop it, we will test the whole Ruleflow, since the Ruleflow represents the decision step that needs to be automated. This enables us to test not only the rules as we define them in the Rulesheet, but also how the Ruleflow works, and how the rules behave as part of the Ruleflow. This way, we can detect and fix problems earlier in the lifecycle.

Create a Ruleflow as follows:

1. Right-click the MyAdvancedTutorial rule project in the Project Explorer view and select File > New > Ruleflow.



2. In the Create New Ruleflow wizard, enter MyAdvancedTutorial as the Ruleflow file name and click Next.

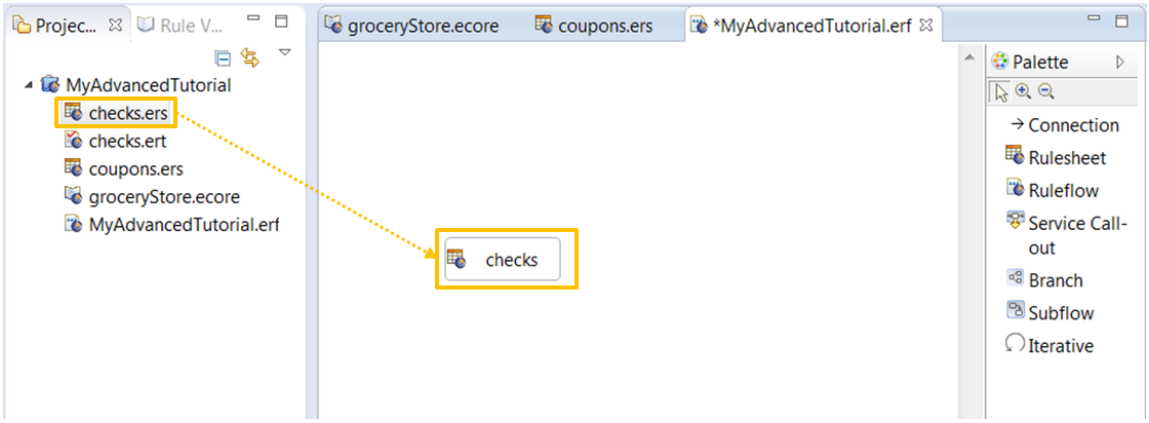


3. Confirm that groceryStore.ecore is selected as the Vocabulary, and then click Finish.

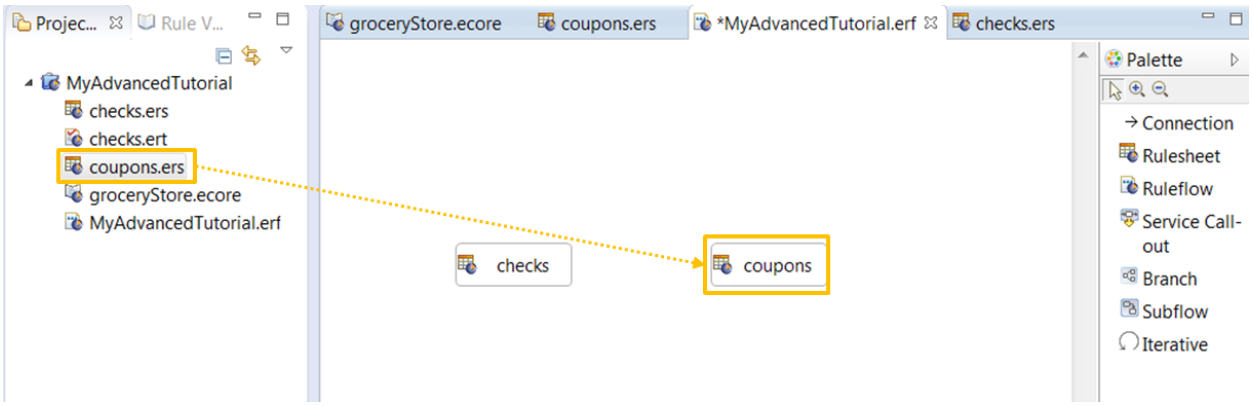
The Ruleflow opens in the its editor.

Now we'll add Rulesheets to the Ruleflow:

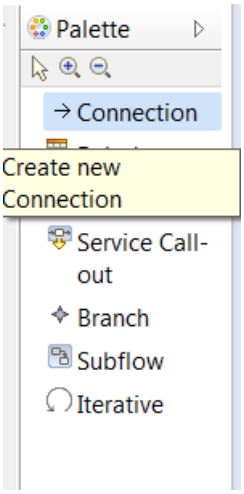
1. Drag and drop checks.ers from the Rule Vocabulary view to the Ruleflow editor.



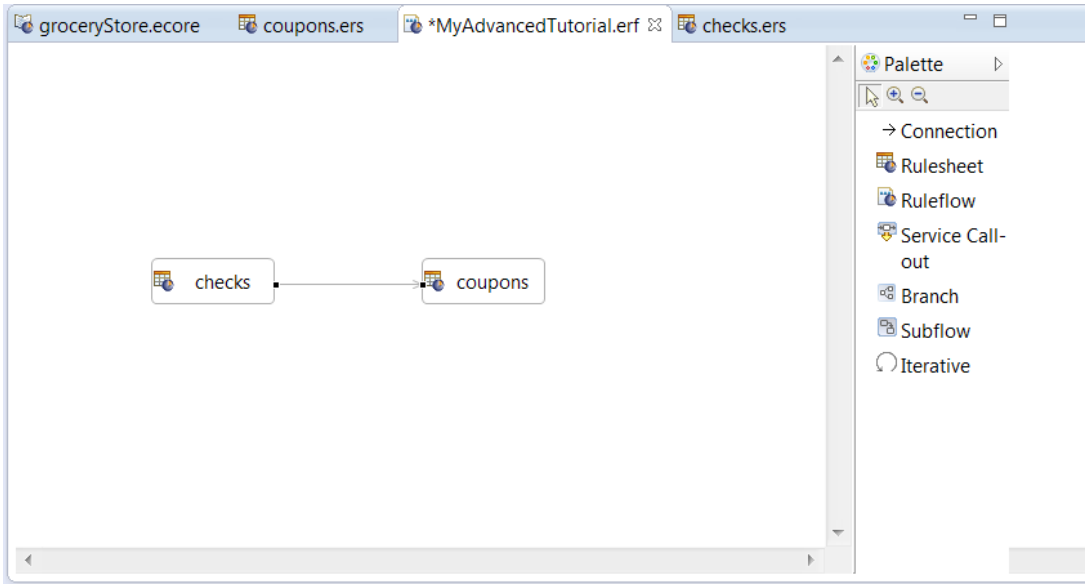
2. Drag and drop coupons.ers from the Rule Vocabulary view to the Ruleflow editor, placing it to the right of the checks.ers Rulesheet.



3. Click Connection in the Palette.



4. Click on checks.ers, keep the mouse pressed, and drag the connection to the coupon.ers Rulesheet. Your final output should look like this. The Ruleflow shows the Rulesheet processing sequence.



Save the Ruleflow.

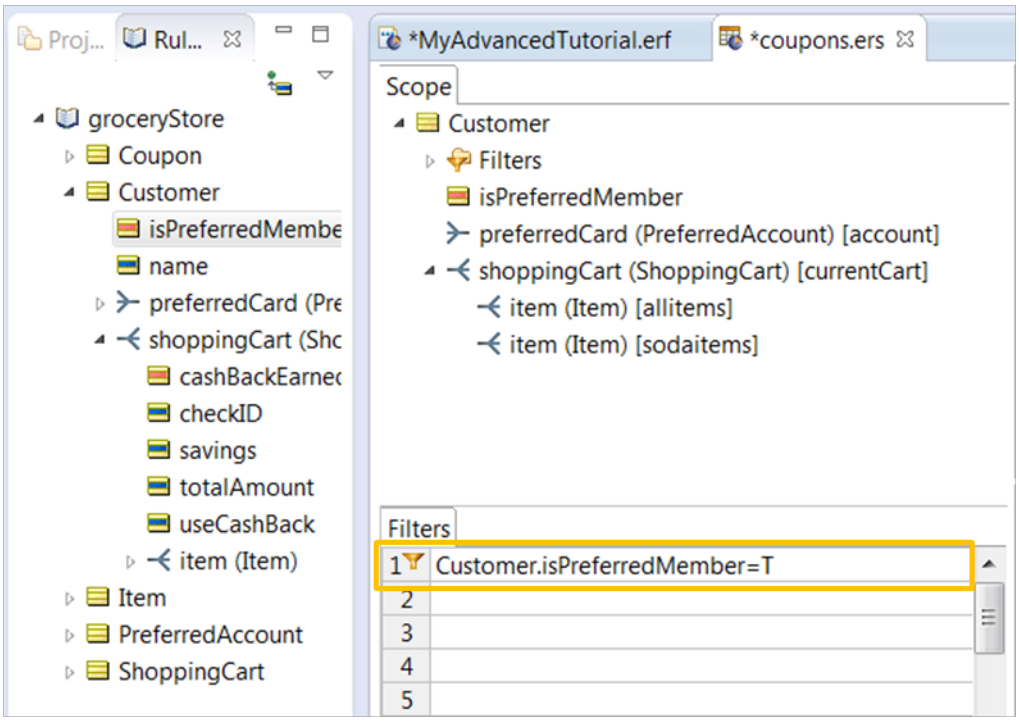
Define a Filter in the coupons.ers Rulesheet

Now, let's go back to our coupons.ers Rulesheet. Before we continue modeling our promotional rules, we need to define a filter to filter out customers who do not have preferred accounts. This is because the promotional rules apply only to customers who have a preferred account card.

To do this, we must define a Filter expression. 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. Note: Data filtered out in one Rulesheet is not also filtered out in other Rulesheets unless you include the Filter expression in those Rulesheets, too.

Create a Filter expression in the Filters pane as shown.



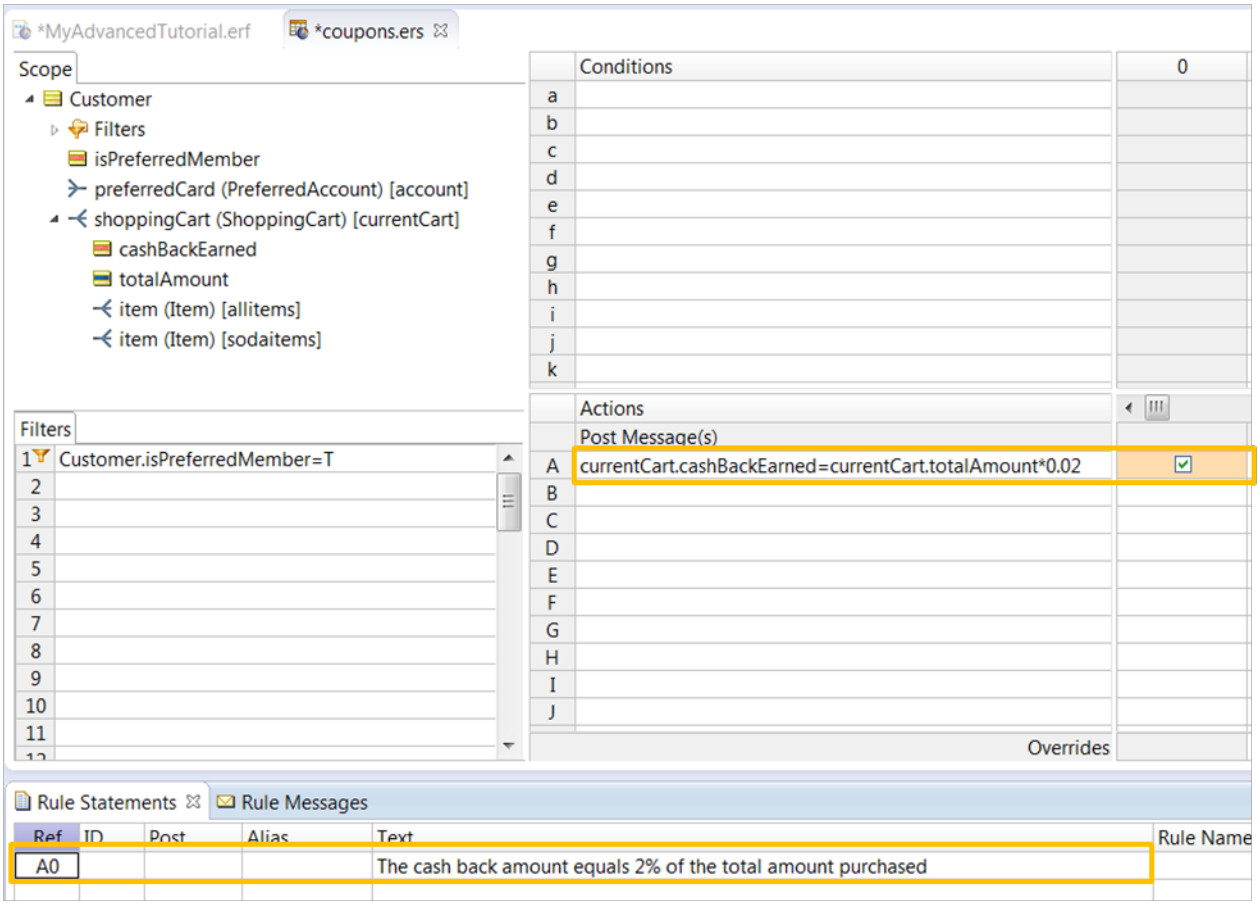
The Filter expression (Customer.isPreferredMember=T) "filters out" all non-preferred customers by allowing only those customers with an isPreferredMember attribute value of true to pass (survive). Those customers whose isPreferredMember attribute value is not true are filtered out and not evaluated by other rules in this Rulesheet.

Define a rule to calculate cashBackEarned

The first rule we will define in the coupons.ers Rulesheet is to calculate cash back for every preferred customer.

> Preferred Shoppers earn 2% cash back on all purchases at any branch.

Define an action-only rule (with a rule statement) as shown:



As you can see, action row A in column 0 calculates the cashBackEarned for a customer's total purchase. Our rule defines the formula as the totalAmount of all items in the customer's shopping cart multiplied by 0.02, which is the same as 2% of totalAmount.

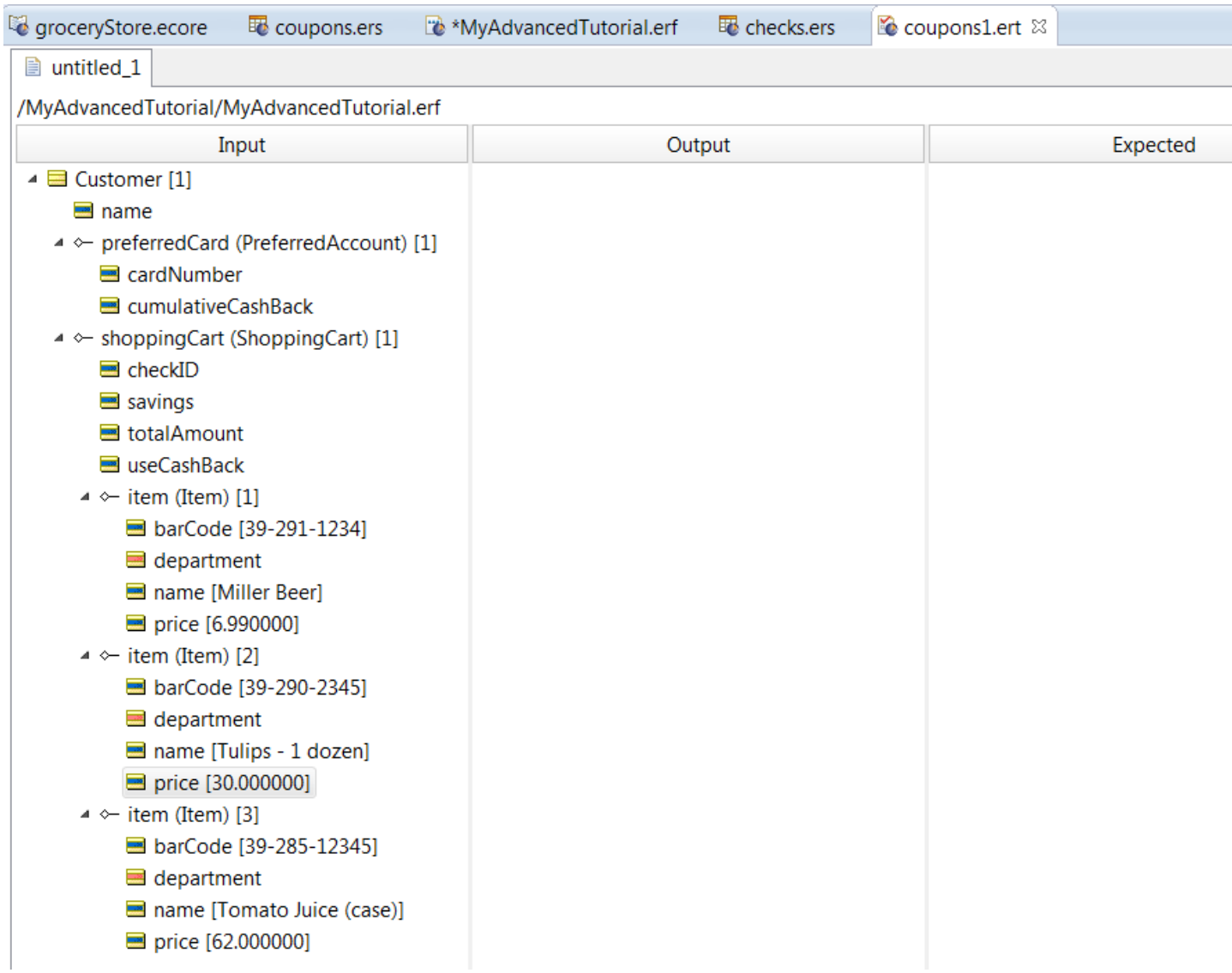
Note: Often, it's desirable to use another Vocabulary attribute (a "parameter") to hold a value, such as the percentage used in this formula, rather than "hard-coding" it (as in 0.02). If the value of an attribute such as cashBackRate is derived by other rules or maintained in an external database then it can be changed without changing the rule that uses it.

Save the Rulesheet.

Now, let's test this rule as part of the Ruleflow. Testing at the Ruleflow level ensures that Rulesheets are processed in the correct sequence and allows values derived in prior Rulesheets to be used in subsequent Rulesheets.

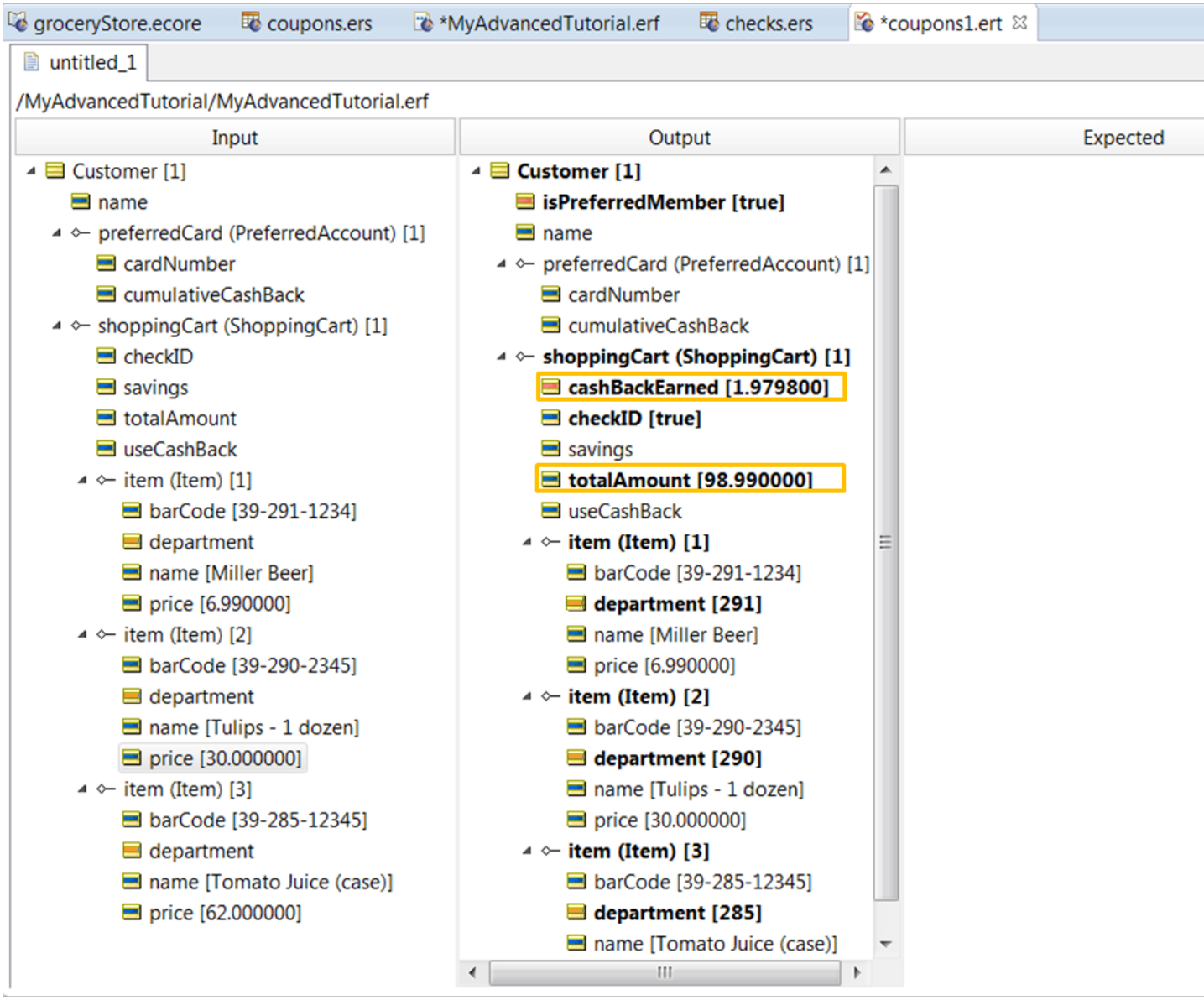
Create a Ruletest named coupons1. Ensure that the test subject of the Ruletest is the MyAdvancedTutorial.erf Ruleflow.

To test the rule, we need to provide input data where the customer is a preferred account holder. Add a few items to the shoppingCart and enter names and prices for each of them. Enter details in the Input pane of the Ruletest as shown here:



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. Also notice that the list of items contains an item from the Liquor department. This should raise an alert, based on the rules in the checks.ers Rulesheet, which is part of the Ruleflow.

Run the Ruletest.



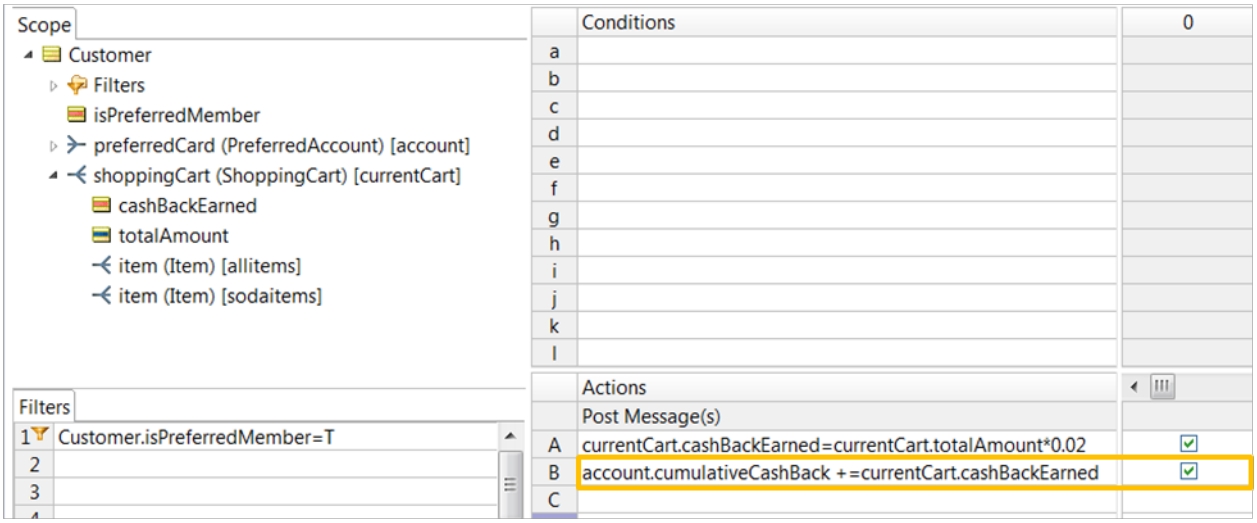
Notice that the totalAmount attribute now has a value of $98.99 (as calculated by a rule in checks.ers) and the cashBackEarned attribute has been assigned a value of $1.9798, or 2% of $98.99. Our rule has worked as expected.

Define a rule to calculate cumulative cash back

> Cash back earned by preferred shoppers should be added to the cumulative cash back in their preferred shopper account.

Now that we have calculated cash back earned, we need a rule to add it to cumulative cash back. Note: In our third Rulesheet, we will define a rule to address the scenario where a customer wants to apply their cumulative cash back to a purchase.

Define an action-only rule as shown:

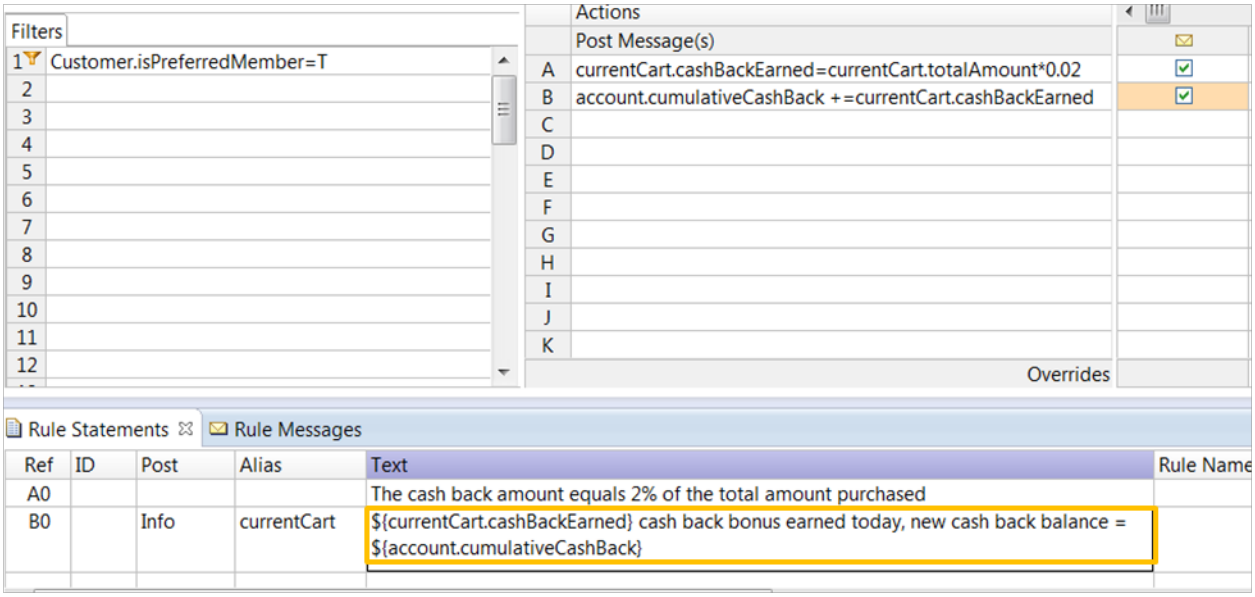


Action row B in Column 0 calculates the cumulativeCashBack amount in a customer's account by incrementing its value (using the += Decimal operator) by the cashBackEarned in the current shopping cart.

Add a rule statement with dynamic data

It is often a good idea to define rule statements that contain dynamic data—where the value of an attribute is extracted and added to the rule message. This can provide more meaningful and informative rule messages.

In this case, we want the value of cashBackEarned as well as cumulativeCashBack to be part of the rule message. We do this by enclosing the attributes in curly braces in the rule statement as shown:



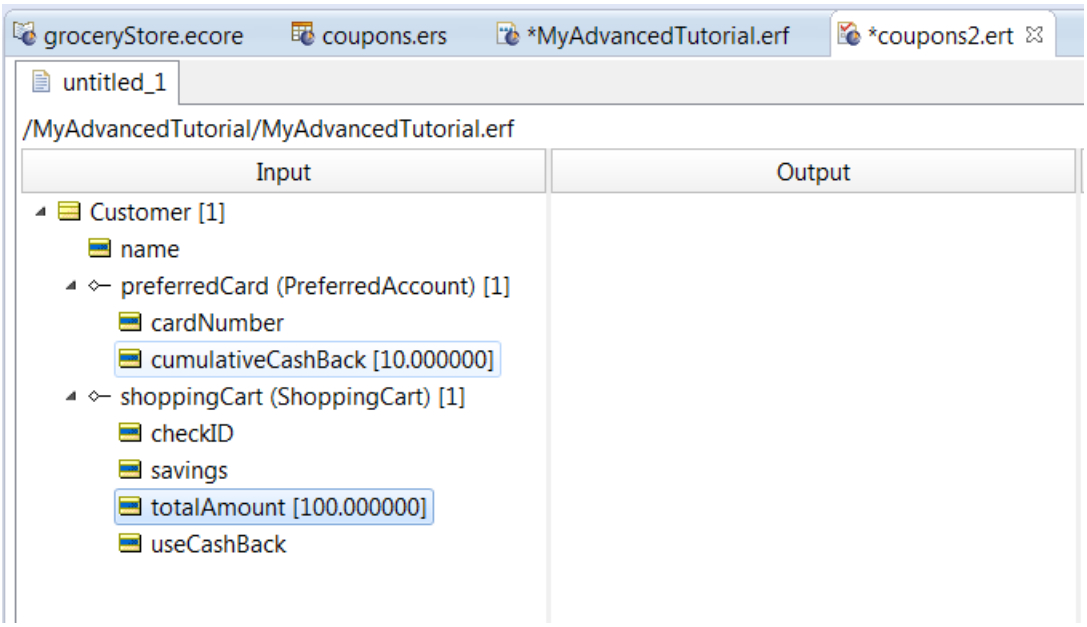
Save the Rulesheet.

Test the rule

Now, let's test the rule. Create a Ruletest named coupons2.ert that uses MyAdvancedTutorial.erf as its test subject.

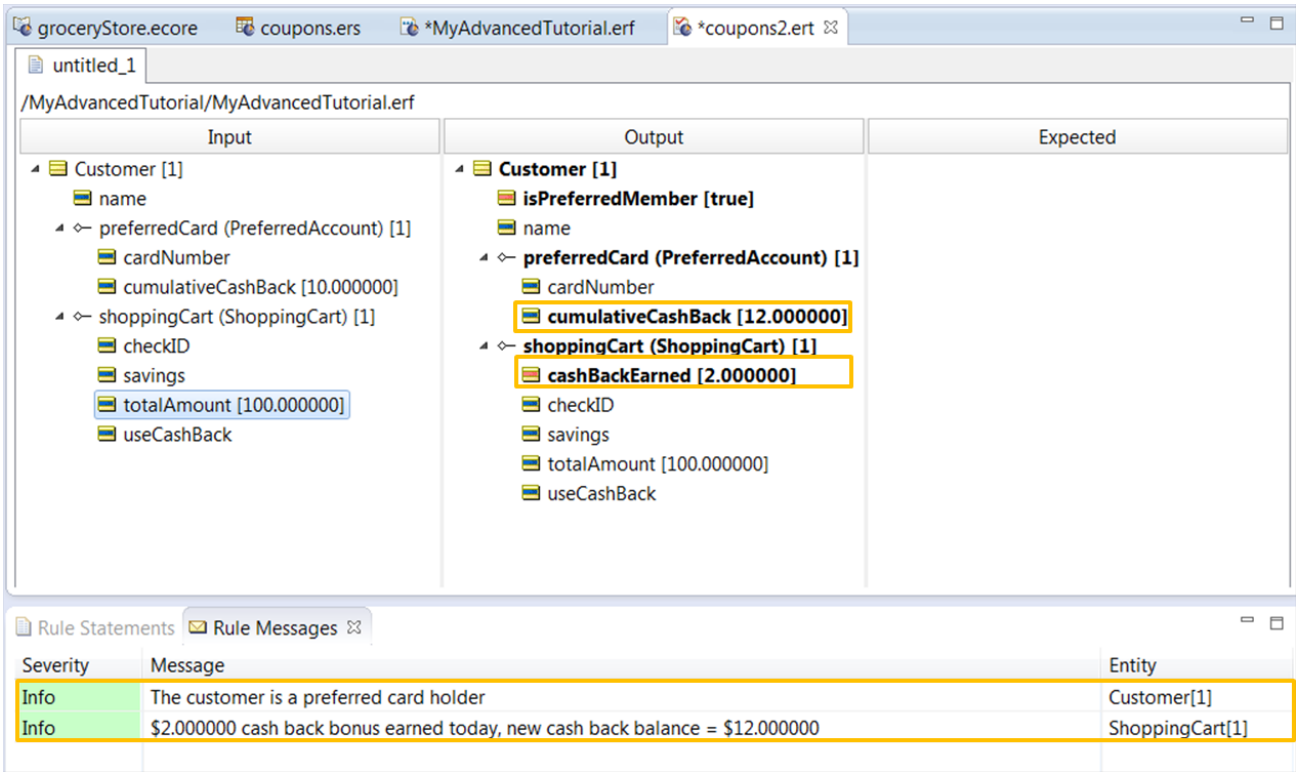
To test that cash back earned is being added to cumulative cash back, we need a starting amount for cumulative cash back and a total amount for the shopping cart. We define in the test input a customer who is a preferred account holder, whose cumulative cash back is $10, and whose total amount (for the shopping cart) is $100. We've already tested this Ruleflow's ability to sum up the prices of each individual item to calculate a totalAmount, so we don't need to enter individual item prices again.

Define the input as shown here:



Note: 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 defined a customer who is a preferred account holder in our test to ensure the Filter is satisfied, and our new rule model has a chance to execute.

Run the test.



As you can see, the rule works as expected. The rule calculates the cash back earned ($2) based on the total amount ($100), and adds it to the cumulative cash back ($10), giving it the updated value of $12. Notice that the rule message contains this data as well.

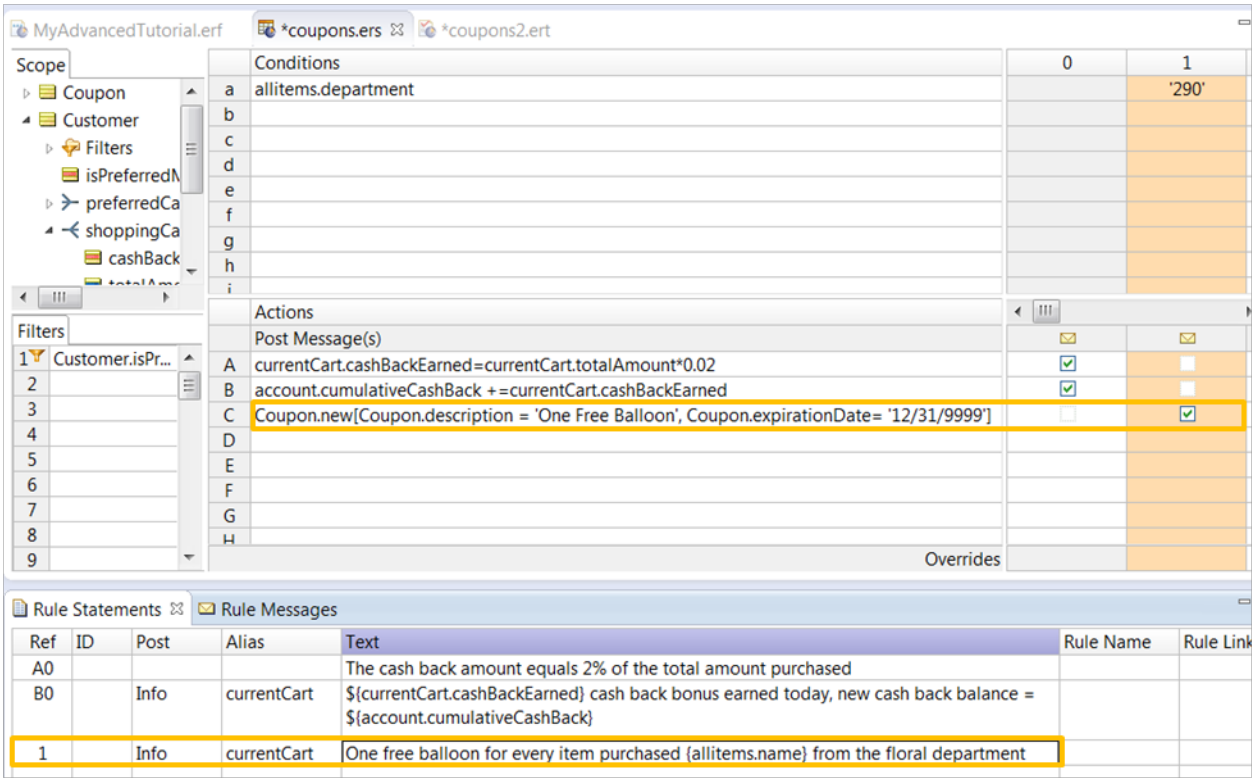
Define a promotional rule for customers purchasing from the Floral department

Now that the cumulative cash back rule is complete, let's move on the next business rule:

> Preferred Shoppers receive a coupon for one free balloon for every item purchased from the Floral department. Expiration date: none

For every item purchased from the Floral department, a coupon must be issued for a free balloon. Let's assume that the Floral department has the department code 290.

Define a rule in coupons.ers as shown.



The first Condition in 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 Entity operator .new) for a free balloon.

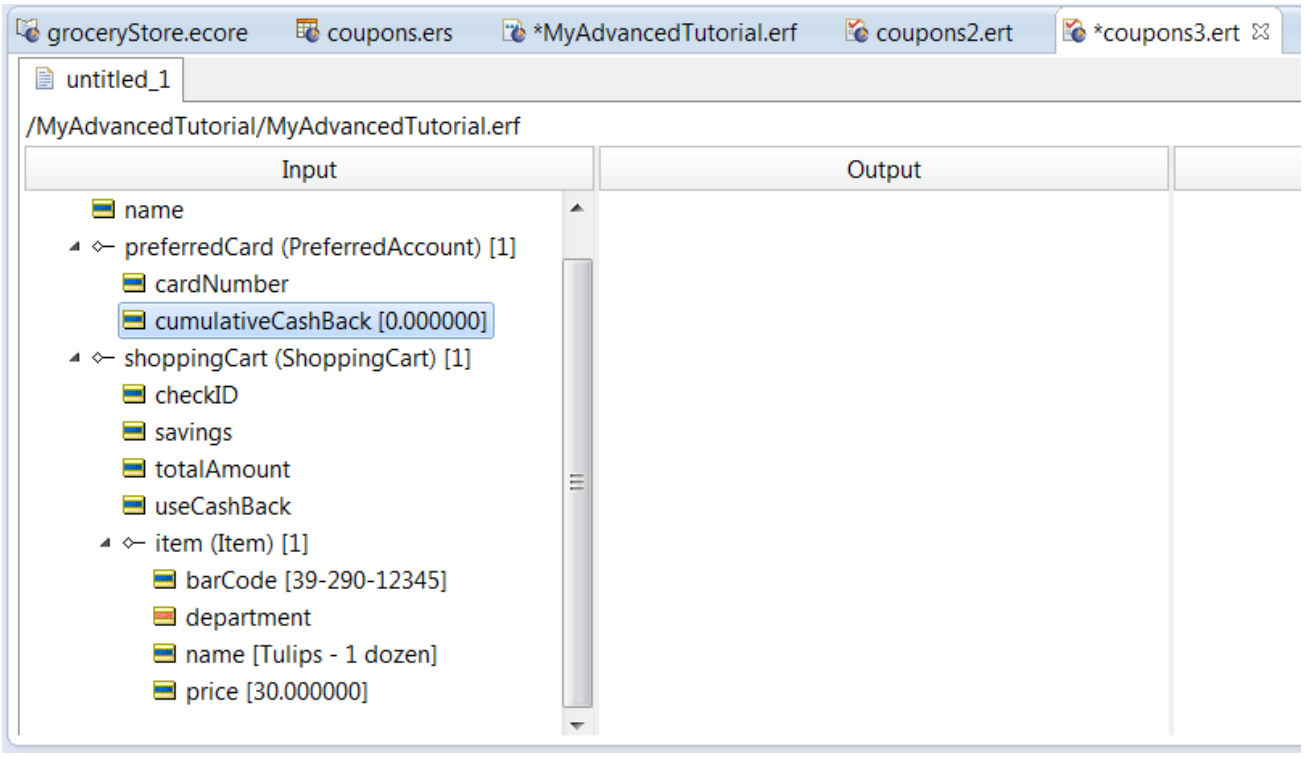
We assign the value of 12/31/9999 to the expirationDate attribute, which is one way to indicate that the expiration date is indefinite. This is the appropriate date format for our use case. Your preferred format might be 31/12/9999. Or any one of the dozen date formats defined in Corticon.

Note: 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.

Save the Rulesheet.

Testing the rule

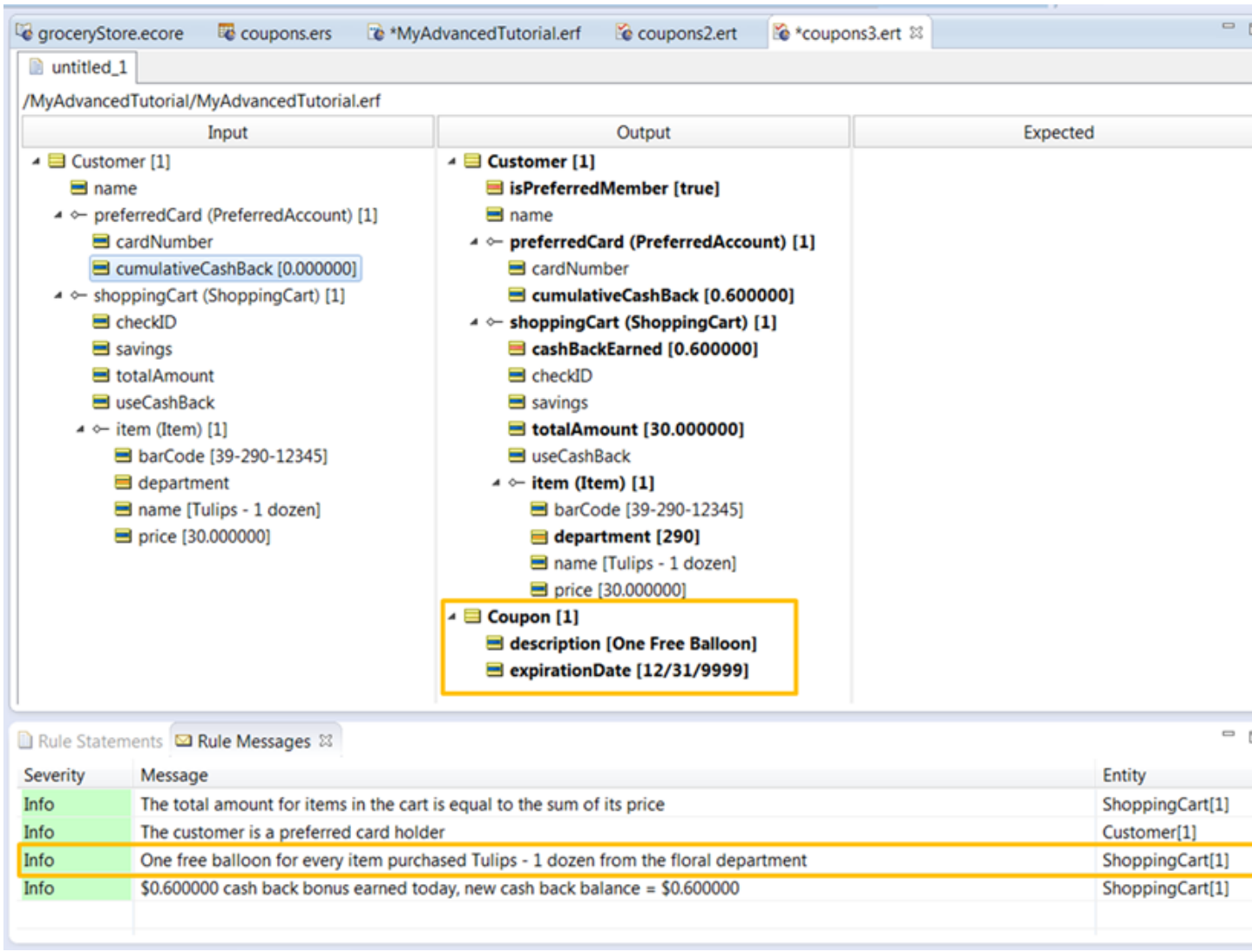
Create a Ruletest named coupons3.ert. Define the input data as shown.



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

Notice also that we set cumulativeCashBack to 0 for this test. One of the rules in the coupons.ers Rulesheet (in Action row B, column 0) needs a real initial value of cumulativeCashBack to increment. If its initial value is null, the rule will not fire.

Run the test.



Our rule has worked as expected. As you can see, 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 in it.

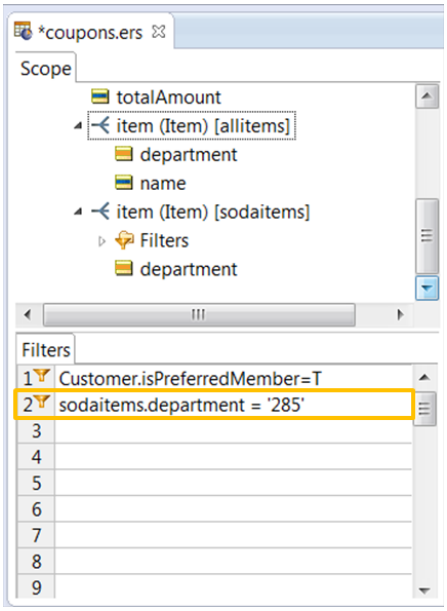
Define a promotional rule for customers purchasing more than 3 soda/juice items

Let's move on to the next business rule:

> 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.

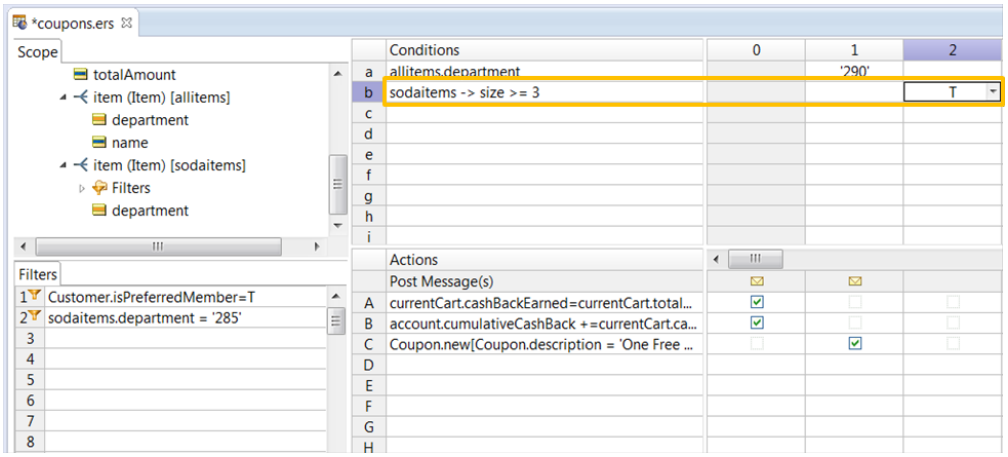
This rule must create a "$2 off" coupon when a customer buys 3 or more items from the Soda/Juice department. When we were determining whether any items from the Floral Department were in the shopping cart, we used the allItems alias. But to determine if 3 or more items were purchased from the Soda/Juice department, we don't need to count all items in a shopping cart, just those from the Soda/Juice department. 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. Let's assume that the department code for the Soda/Juice department is 285. Define a Filter expression as shown:



The filter in row 2 ensures that the "surviving" members of the sodaItems alias all have a department value of 285.

Now, let's model the rule. The Collection operator size counts the number of elements in a collection. We'll use this operator to check how many soda/juice items are in the shopping cart. Define a condition as shown:



If the number of items counted by the size operator in the sodaitems collection is 3 or more, then a $2 off coupon will be issued to the customer. Define the action for this as shown:

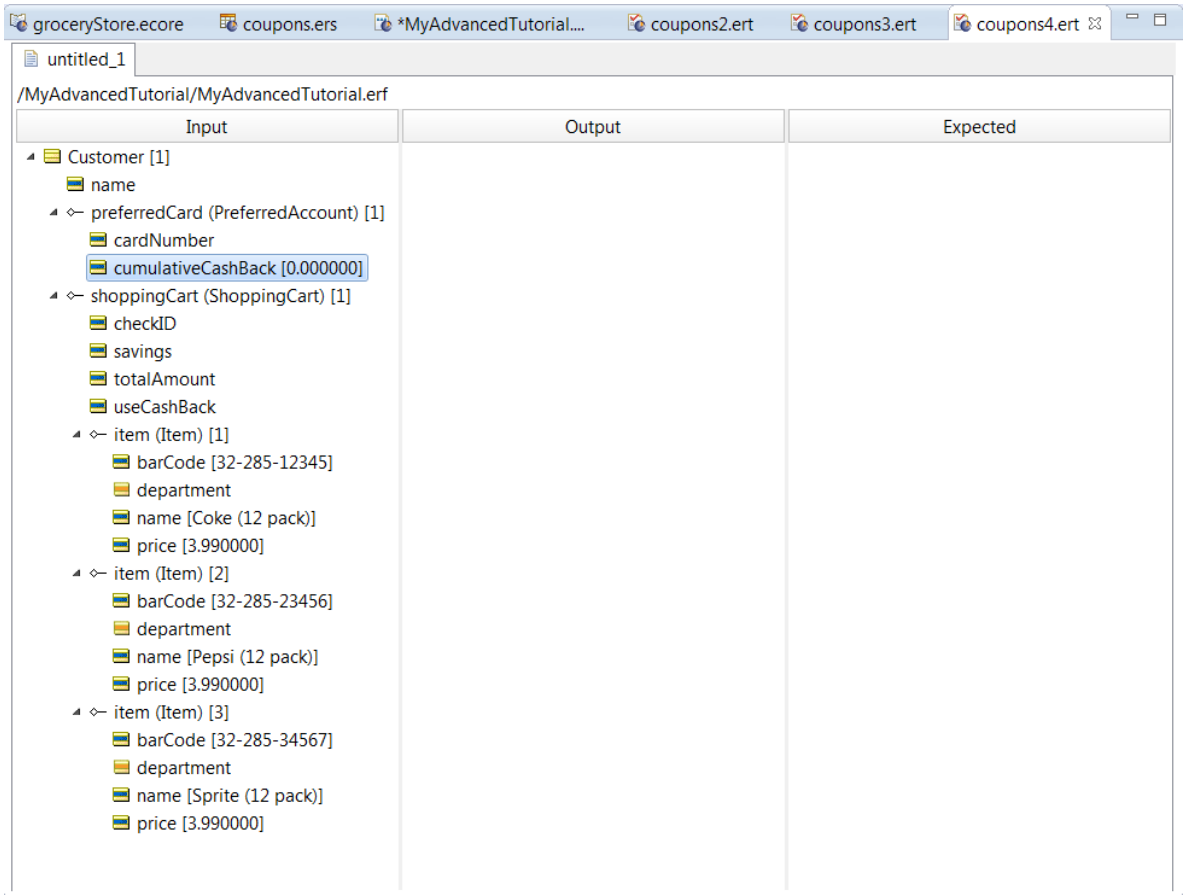
	Conditions	0	1	2
a	allitems.department		'290'	
b	sodaitems -> size >= 3			T
c				
d				
e				
f				
g				
h				
	Actions			
	Post Message(s)	✉	✉	
A	currentCart.cashBackEarned=currentCart.totalAmount*0.02	☑	☐	☐
B	account.cumulativeCashBack +=currentCart.cashBackEarned	☑	☐	☐
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate= '12/31/9999']	☐	☑	☐
D	Coupon.new[Coupon.description = '2$ off on the next purchase', Coupon.expirationDate = today.addYears(1)]	☐	☐	☑
E				
F				
G				

The action creates a new Coupon. The expirationDate attribute derives its value from the Date operator .addYears, set here to 1, so we know that the coupon will expire one year from its date of issue.

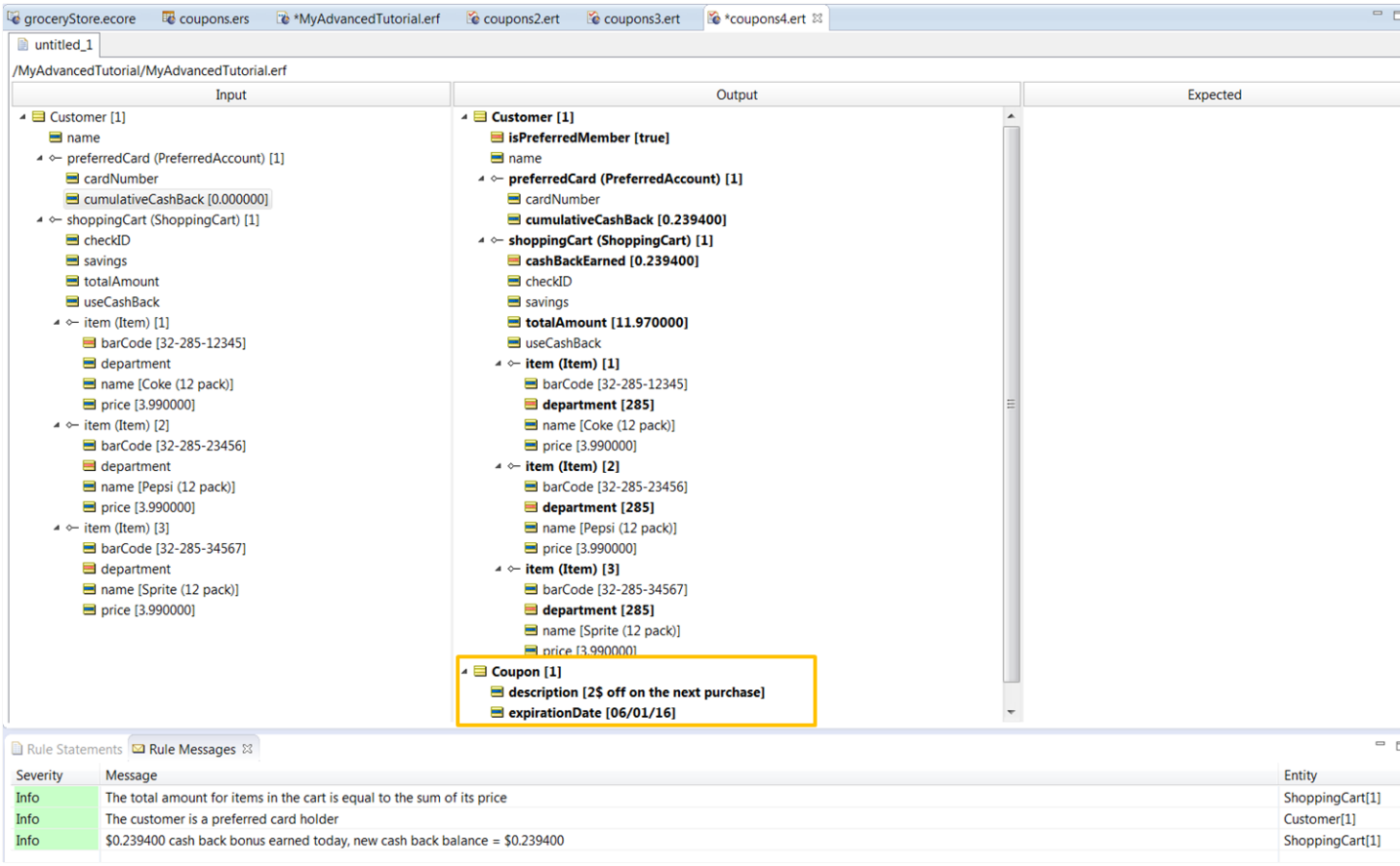
Save the Rulesheet.

Next, let's test this rule. Create a Ruletest named coupons4.ert that uses the MyAdvancedTutorial.erf Ruleflow as its test subject.

To test this rule, our shopping cart must contain 3 or more items from the Soda/Juice Department (department 285). Additionally, the customer must be a preferred account holder so that the data passes the filter (Customer.isPreferredMember=T) in this Rulesheet. Finally, the cumulative cash back must be set to 0, to enable an earlier rule to fire, where cashBackEarned is added to cumulativeCashBack (and it cannot be added to a null value). Define the input as shown:



Run the Ruletest.



The rule works as expected. The items from the Soda/Juice department have been identified and counted. A Coupon has been added with a “$2 off next purchase” description and an expirationDate of 06/01/2016 (which is 1 year from the date this test was run).

The other rules have fired as well. For example, the total amount of the shopping cart and the cash back earned have been calculated. The cash back earned has been added to the cumulative cash back (which was set to 0).

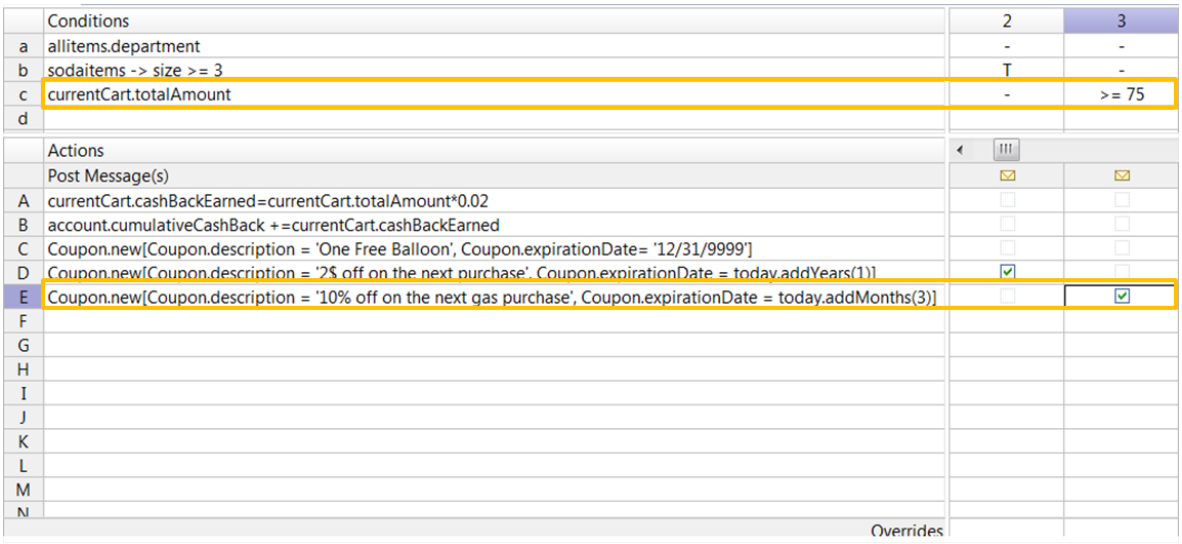
Note: It is a good practice to run multiple tests with different data to make sure that your rules work as expected in different scenarios. For example, you could change the department code of one of the items to 291 (liquor) and another one to 290 (floral). When you run the test with the new data, the alert to check the customer’s ID (liquor) and the free balloon coupon (floral) should be generated. However, the soda coupon should not get generated as you no longer have three soda items.

Define a promotional rule for customers purchasing more than $75 in this cart

Let’s now model the last rule in the coupons Rulesheet:

> 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.

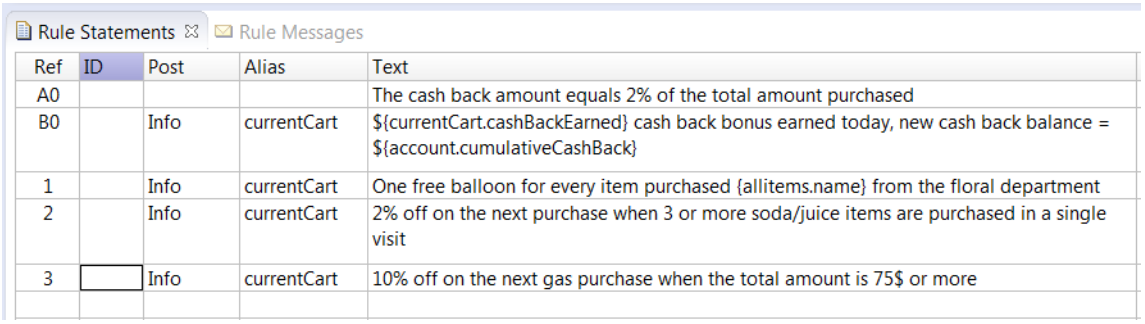
The last rule checks if the total amount is 75$ or more, and if so, issues a coupon for 10% off of a future gasoline purchase. Define this rule as shown:



	Conditions	2	3
a	allitems.department	-	-
b	sodaitems -> size >= 3	T	-
c	currentCart.totalAmount	-	>= 75
d			
	Actions		
	Post Message(s)	✉	✉
A	currentCart.cashBackEarned=currentCart.totalAmount*0.02	☐	☐
B	account.cumulativeCashBack +=currentCart.cashBackEarned	☐	☐
C	Coupon.new[Coupon.description = 'One Free Balloon', Coupon.expirationDate= '12/31/9999']	☐	☐
D	Coupon.new[Coupon.description = '2$ off on the next purchase', Coupon.expirationDate = today.addYears(1)]	☑	☐
E	Coupon.new[Coupon.description = '10% off on the next gas purchase', Coupon.expirationDate = today.addMonths(3)]	☐	☑
F			
G			
H			
I			
J			
K			
L			
M			
N			
	Overrides		

The rule condition identifies when the totalAmount of a customer's shopping cart is $75 or more. The rule action creates a new coupon (again, using the .new operator) for 10% off a future gasoline purchase. The expirationDate attribute derives its value from the Date operator .addMonths, set here to 3, so the coupon will expire three months from its date of issue.

As always, it is a good practice to add a corresponding rule statement, explaining in clear language what the business rule does.

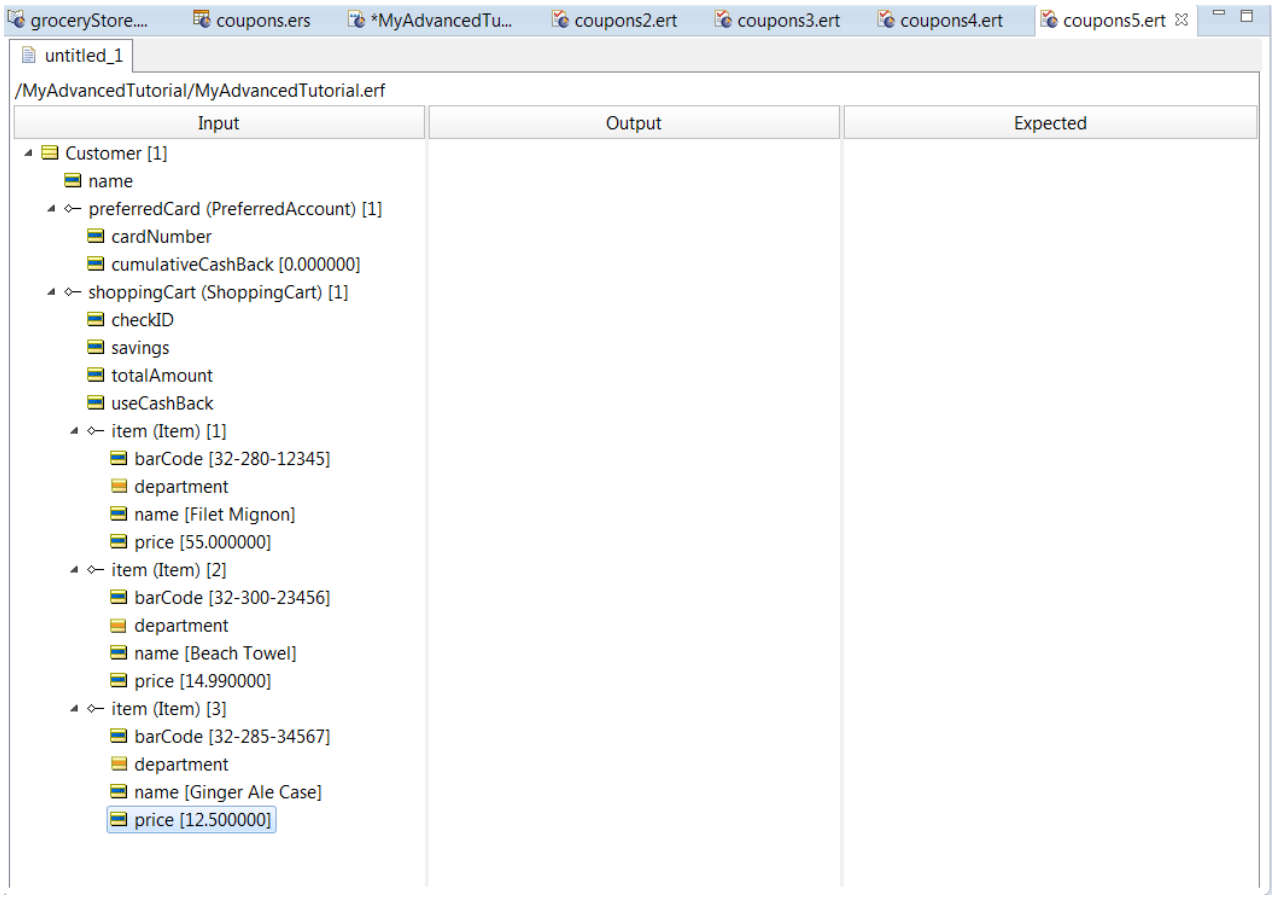


Rule Statements | Rule Messages

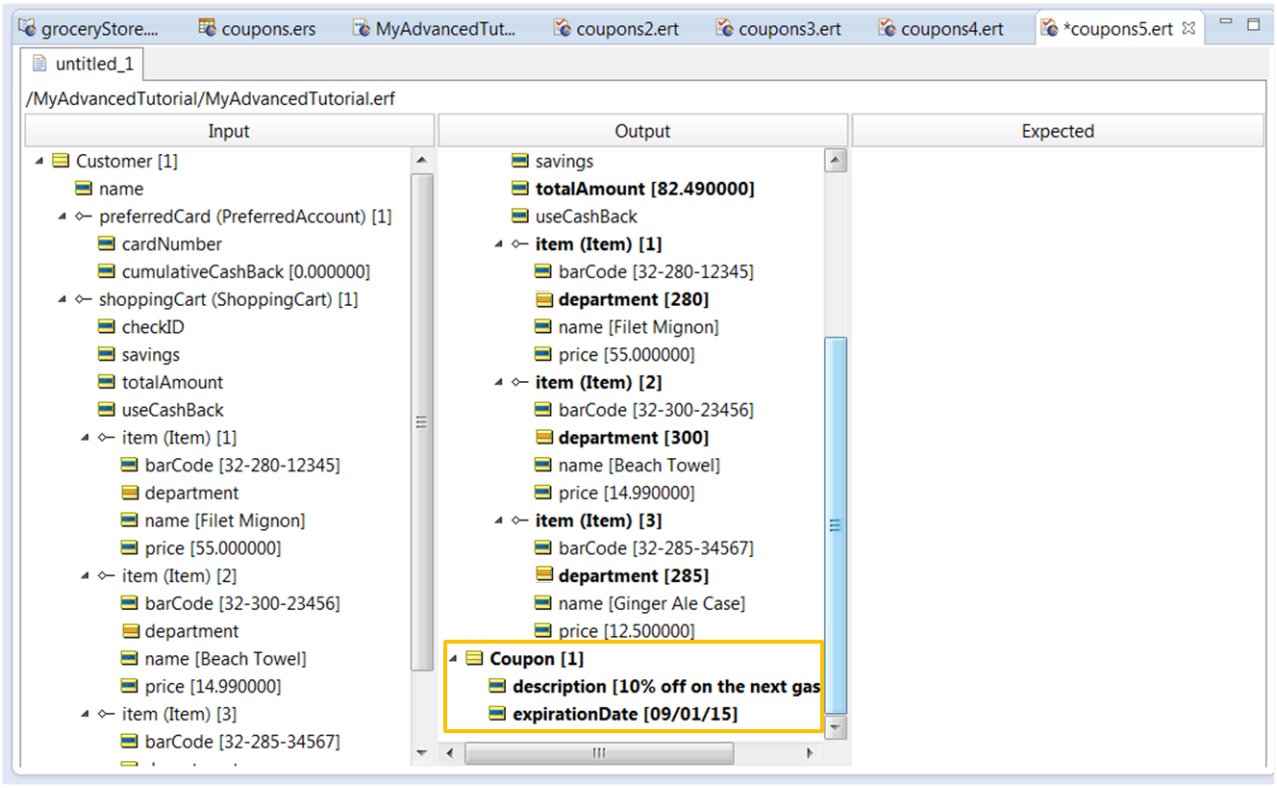
Ref	ID	Post	Alias	Text
A0				The cash back amount equals 2% of the total amount purchased
B0		Info	currentCart	${currentCart.cashBackEarned} cash back bonus earned today, new cash back balance = ${account.cumulativeCashBack}
1		Info	currentCart	One free balloon for every item purchased {allitems.name}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

Let's test this rule. Create a new Ruletest named coupons5.ert that uses the MyAdvancedTutorial.erf Ruleflow as its test subject.

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. Define the input as shown.

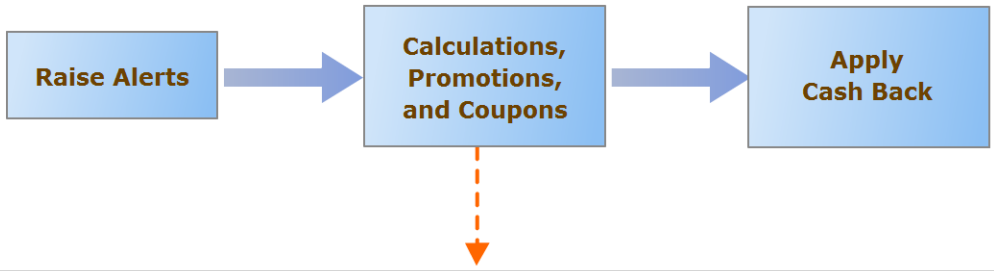

groceryStore....
coupons.ers
*MyAdvancedTu...
coupons2.ert
coupons3.ert
coupons4.ert
coupons5.ert
untitled_1
/MyAdvancedTutorial/MyAdvancedTutorial.erf
Input
Output
Expected
Customer [1]
name
preferredCard (PreferredAccount) [1]
cardNumber
cumulativeCashBack [0.000000]
shoppingCart (ShoppingCart) [1]
checkID
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 [32-285-34567]
department
name [Ginger Ale Case]
price [12.500000]

Run the test.


groceryStore....
coupons.ers
MyAdvancedTut...
coupons2.ert
coupons3.ert
coupons4.ert
*coupons5.ert
untitled_1
/MyAdvancedTutorial/MyAdvancedTutorial.erf
Input
Output
Expected
Customer [1]
name
preferredCard (PreferredAccount) [1]
cardNumber
cumulativeCashBack [0.000000]
shoppingCart (ShoppingCart) [1]
checkID
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 [32-285-34567]
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 [32-285-34567]
department [285]
name [Ginger Ale Case]
price [12.500000]
Coupon [1]
description [10% off on the next gas
expirationDate [09/01/15]

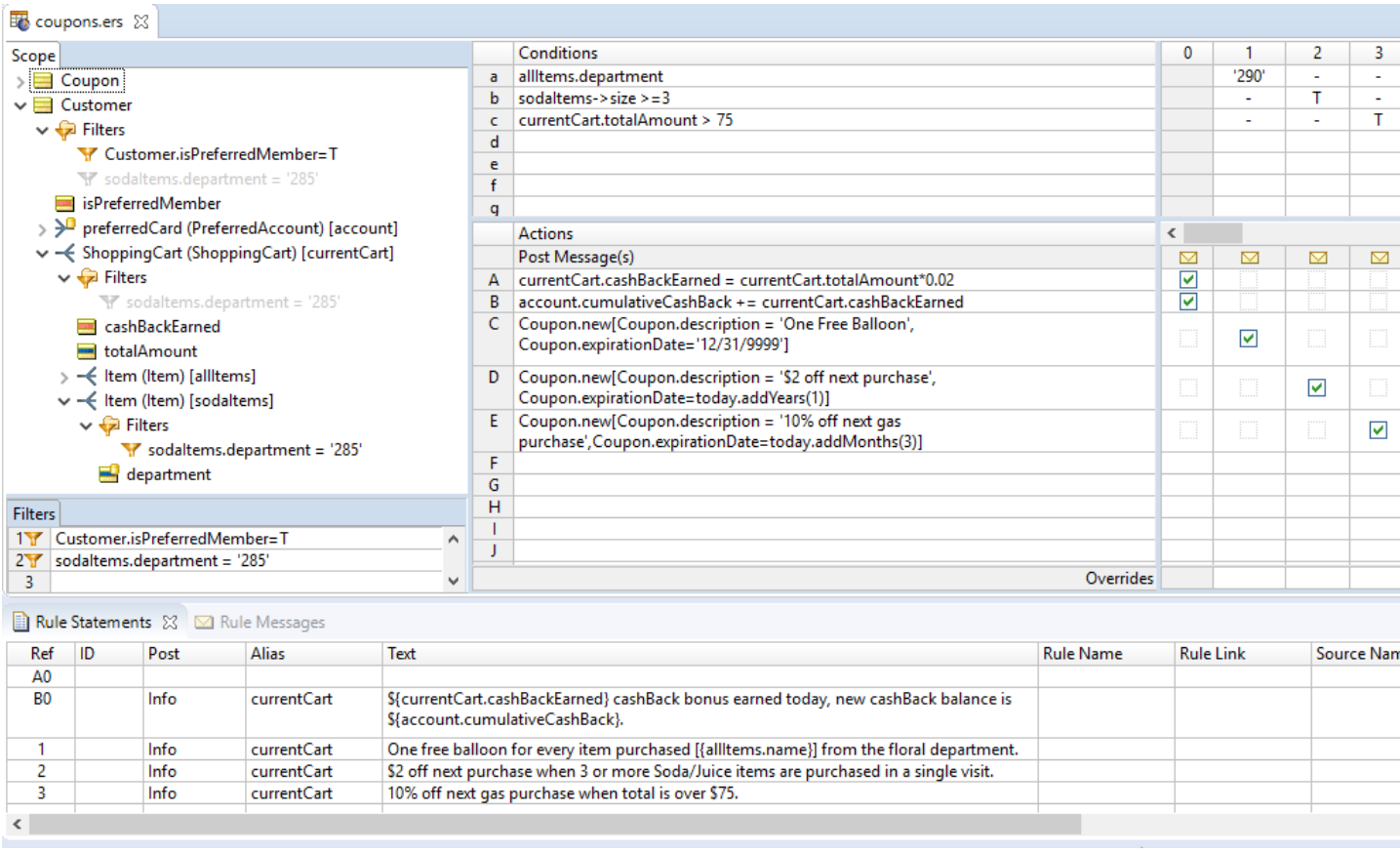
Our rule has worked as expected. The items have been totaled and the amount exceeds the $75 threshold so the 10% off Coupon has been created.

We have now modeled all the rules in the Calculations, Promotions and Coupons substep of the Checkout process.



- Preferred Shoppers earn 2% cash back on all purchases at any branch.
- Cash back earned by preferred shoppers should be added to the cumulative cash back in their preferred shopper account.
- 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.

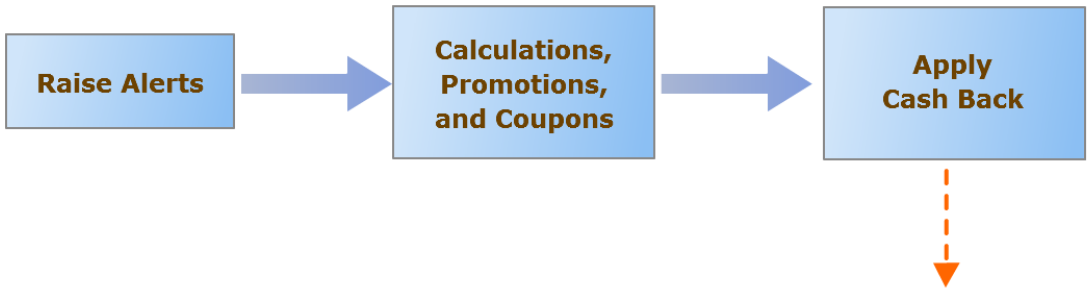
Here is our completed Rulesheet:


coupons.ers
Scope
Coupon
Customer
Filters
Customer.isPreferredMember=T
sodaItems.department = '285'
isPreferredMember
preferredCard (PreferredAccount) [account]
ShoppingCart (ShoppingCart) [currentCart]
Filters
sodaItems.department = '285'
cashBackEarned
totalAmount
Item (Item) [allItems]
Item (Item) [sodaItems]
Filters
sodaItems.department = '285'
department
Filters
1 Customer.isPreferredMember=T
2 sodaItems.department = '285'
3
Conditions
a allItems.department '290' - -
b sodaItems->size >=3 - T -
c currentCart.totalAmount > 75 - - T
d
e
f
q
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 gas purchase',Coupon.expirationDate=today.addMonths(3)]
F
G
H
I
J
Overrides
Rule Statements
Rule Messages
Ref ID Post Alias Text Rule Name Rule Link Source Nam
A0
B0 Info currentCart ${currentCart.cashBackEarned} cashBack bonus earned today, new cashBack balance is ${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 is over $75.

6

Model the third Rulesheet

We are now ready to model the last Rulesheet. Recall that the third Rulesheet corresponds to the third substep in the Checkout process.



- 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.

One of the rules in the previous Rulesheet calculated a preferred card member's cashBackEarned 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's totalAmount. If the shopper says "Yes" to using cash back, 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 will deduct cumulativeCashBack from the totalAmount, reducing the amount the shopper pays.

Finally, when a shopper applies his cumulativeCashBack balance, we reset the balance to zero.

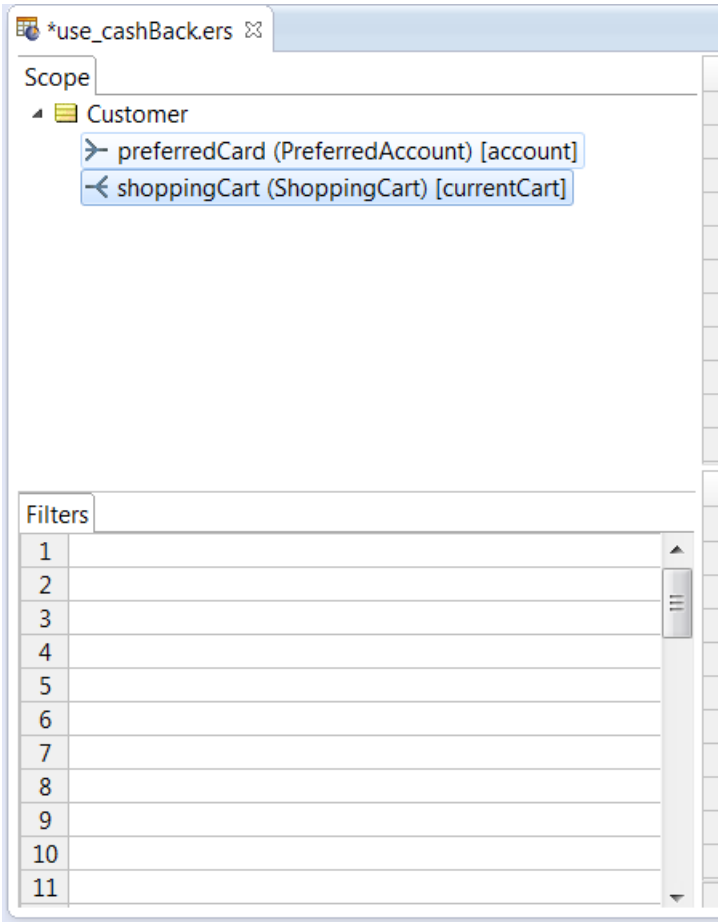
We'll model this as a single rule in Corticon Studio with one condition—check if useCashBack is true—and two actions—deduct cumulativeCashBack from totalAmount and set cumulativeCashBack to 0.

Create the Rulesheet

Let's begin by creating a Rulesheet. Name it use_cashBack. Ensure that it uses the groceryStore.ecore Vocabulary.

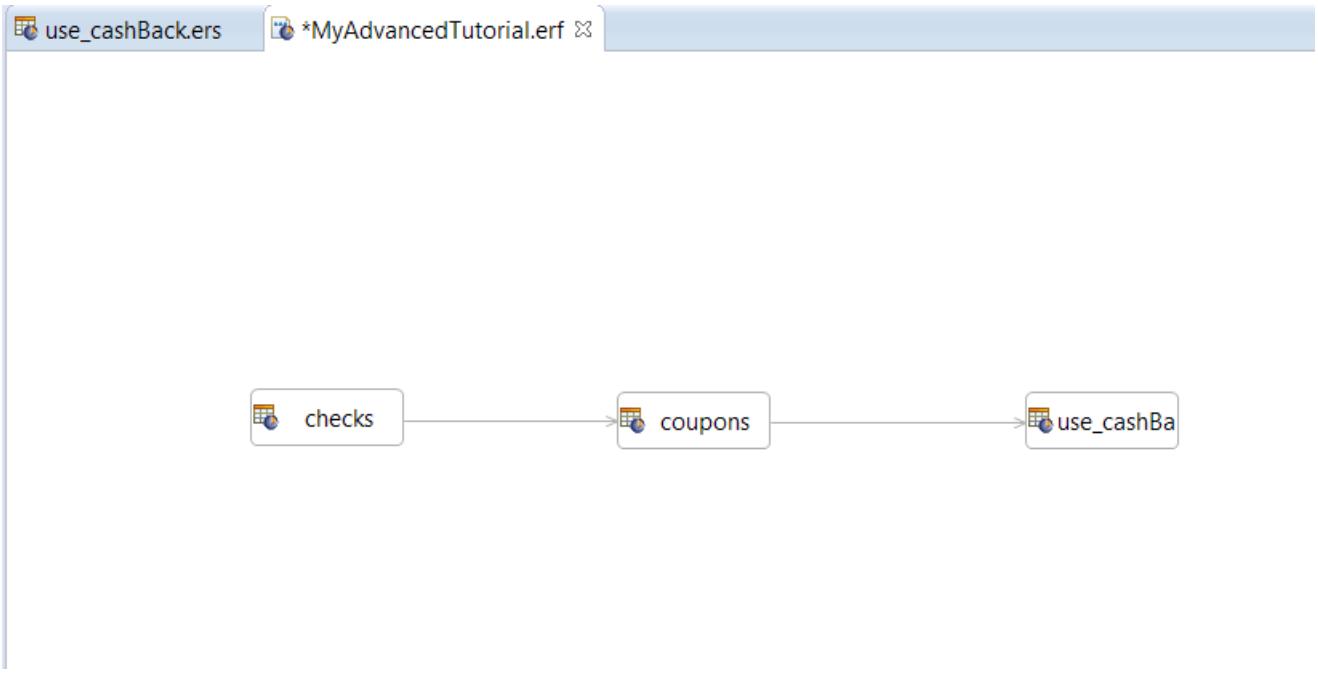
Define the rule scope

Because the rules in this Rulesheet deal with a preferred shopper's cart, we only need a few aliases to represent these perspectives of our Vocabulary. Define the rule scope as shown.



Add the third Rulesheet to the Ruleflow

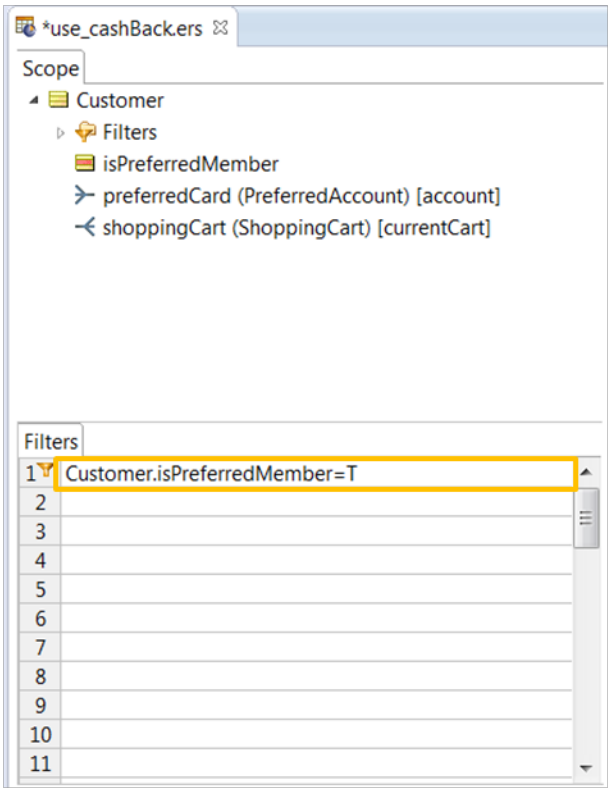
With the creation of the third Rulesheet, we can complete our Ruleflow, implementing the execution sequence of the 3 Rulesheets. Open MyAdvancedTutorial.erf (if it isn't already open), and add the use_cashBack.ers Rulesheet as shown.



Define a Filter in the use_cashBack Rulesheet

In the coupons.ers Rulesheet we defined a Filter to filter out customers who are not preferred account holders. We need to define the same filter in the use_cashBack.ers Rulesheet, since only preferred customers are eligible for cash back and bonus incentives. We need to define the Filter again because data that is filtered out in one Rulesheet is not automatically filtered out in other Rulesheets.

Define a Filter expression as shown.

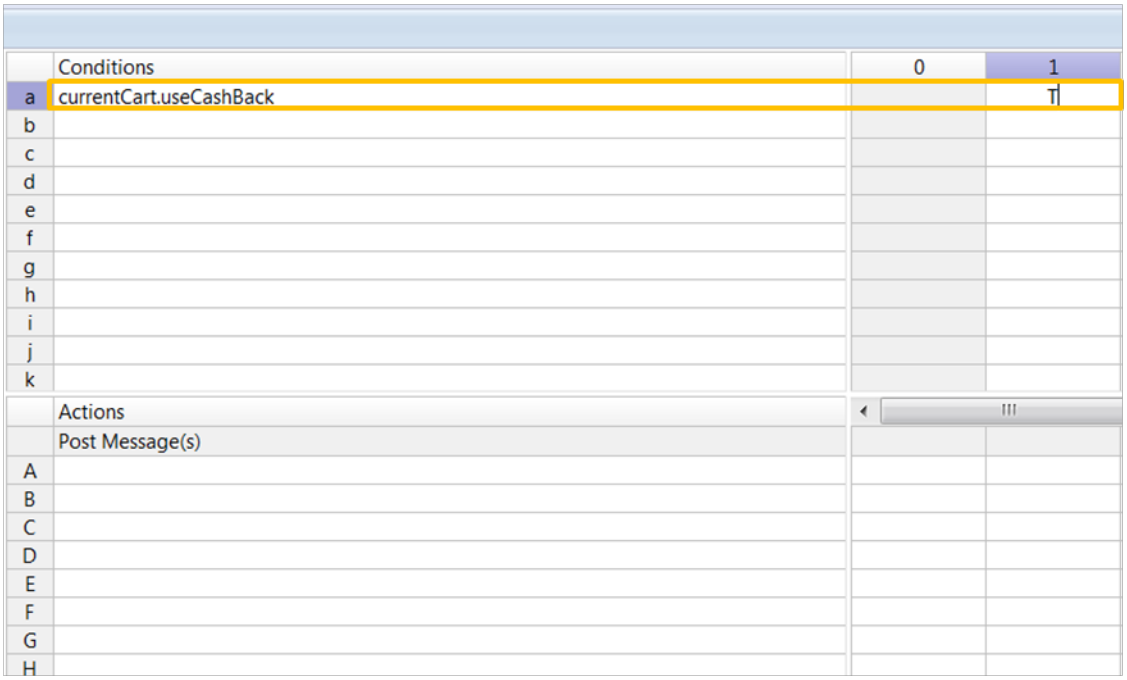


Define a rule to apply cash back

Let's start modeling the first business rule:

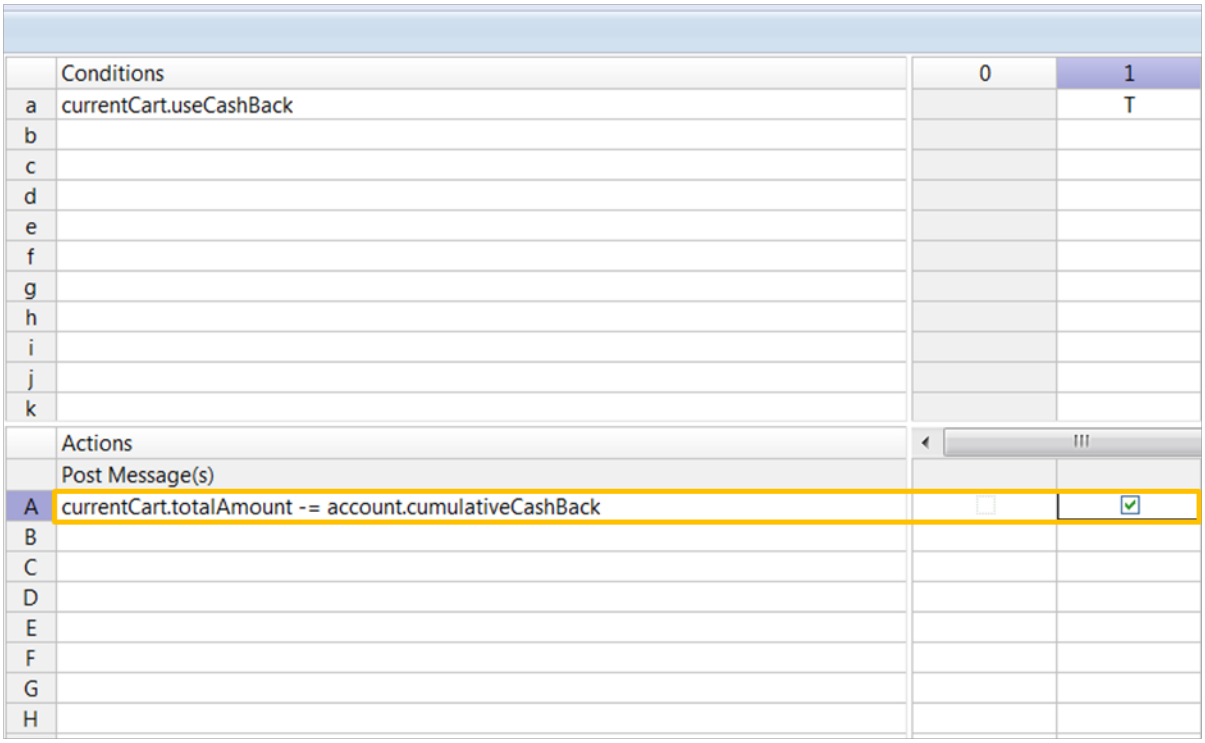
> A Preferred Shopper account will track the accumulated cash back and allow the customer to apply it to reduce any visit's total amount. The cashier will ask a Preferred Shopper if he/she would like to apply a cash back balance to his/her current purchase.

The rule has only one condition—check if the customer wants to apply their cumulativeCashBack to the totalAmount of the ShoppingCart. Define the condition expression as shown.



As you can see, we only want to process the currentCart when the shopper has chosen to apply their cashBack balance to the current purchase, in other words, when useCashBack = true.

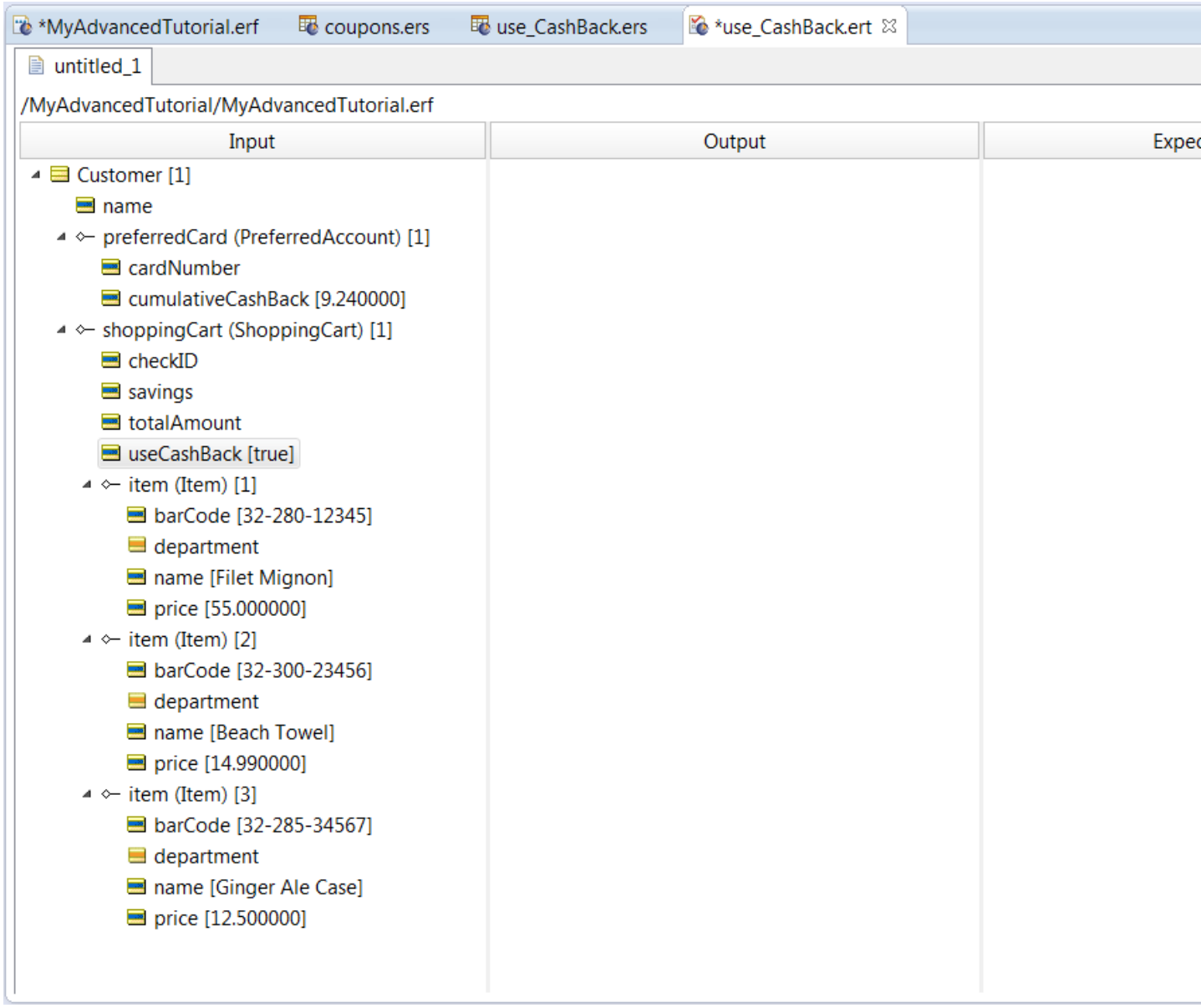
Next, define an action to deduct the cumulativeCashBack from the totalAmount.



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

In keeping with our model/test approach, we'll test the rule before adding more to it.

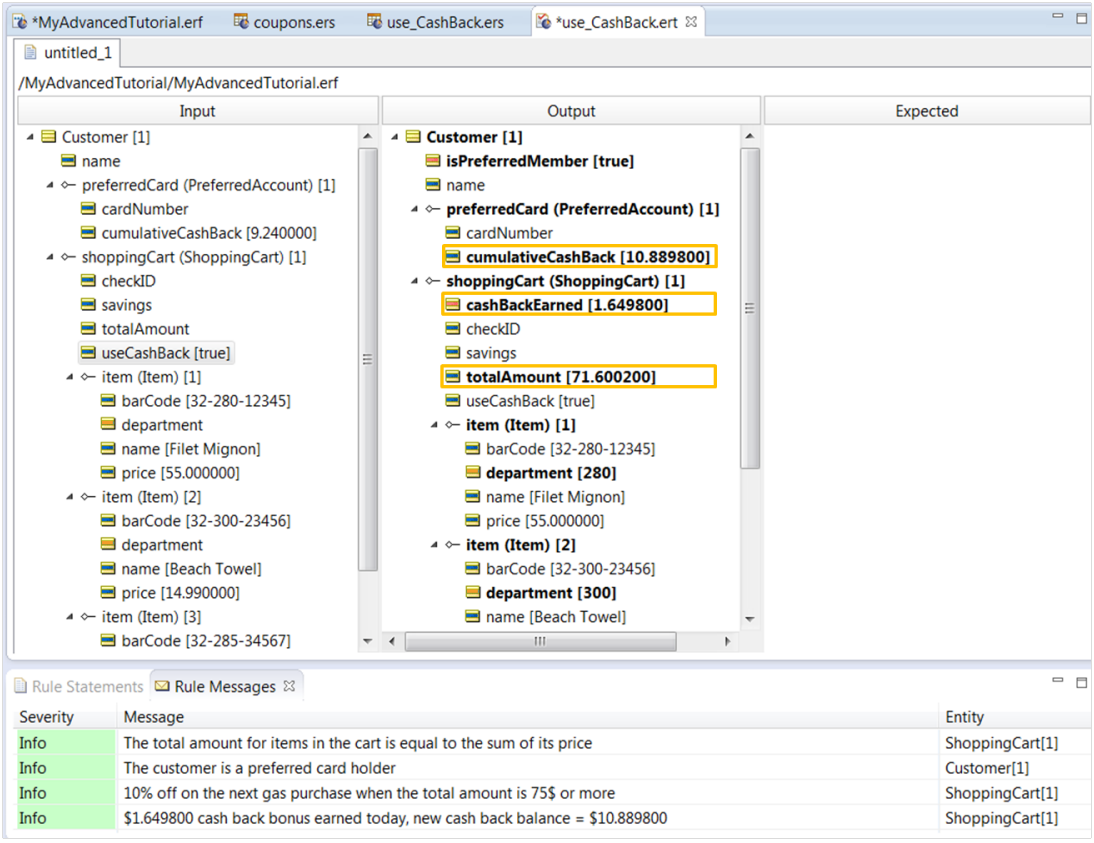
Create a Ruletest named use_cashBack.ert that uses the MyAdvancedTutorial.erf Ruleflow as its test subject. Define the input as shown.



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

According to our first rule in the use_cashBack.ers Rulesheet, 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.

Run the test.



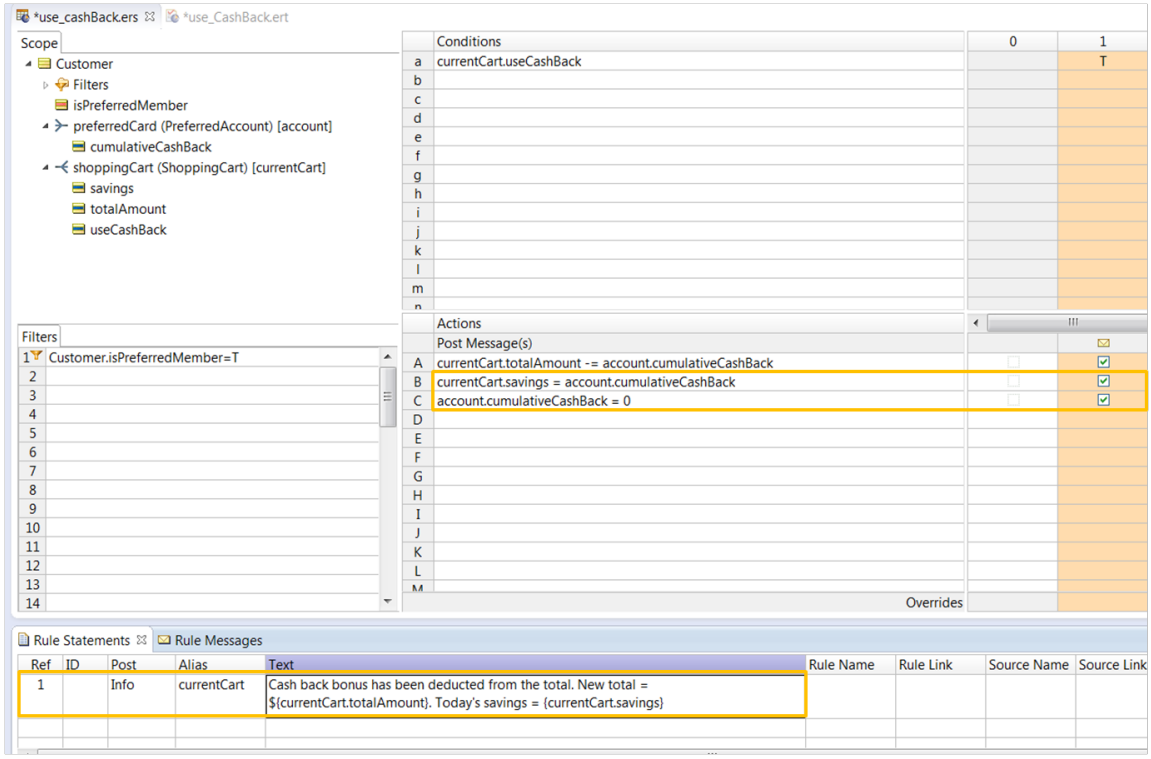
Our rule has worked as expected. The Output panel shows the new cashBackEarned ($1.64) is added to cumulativeCashBack ($9.24+$1.64=$10.88) and subtracted from totalAmount ($82.49-$10.88=$71.6). We also see the cashBackEarned and the cumulativeCashBack values embedded in a rule message from the previous Rulesheet.

But we're not done. We still need to model the second business rule:

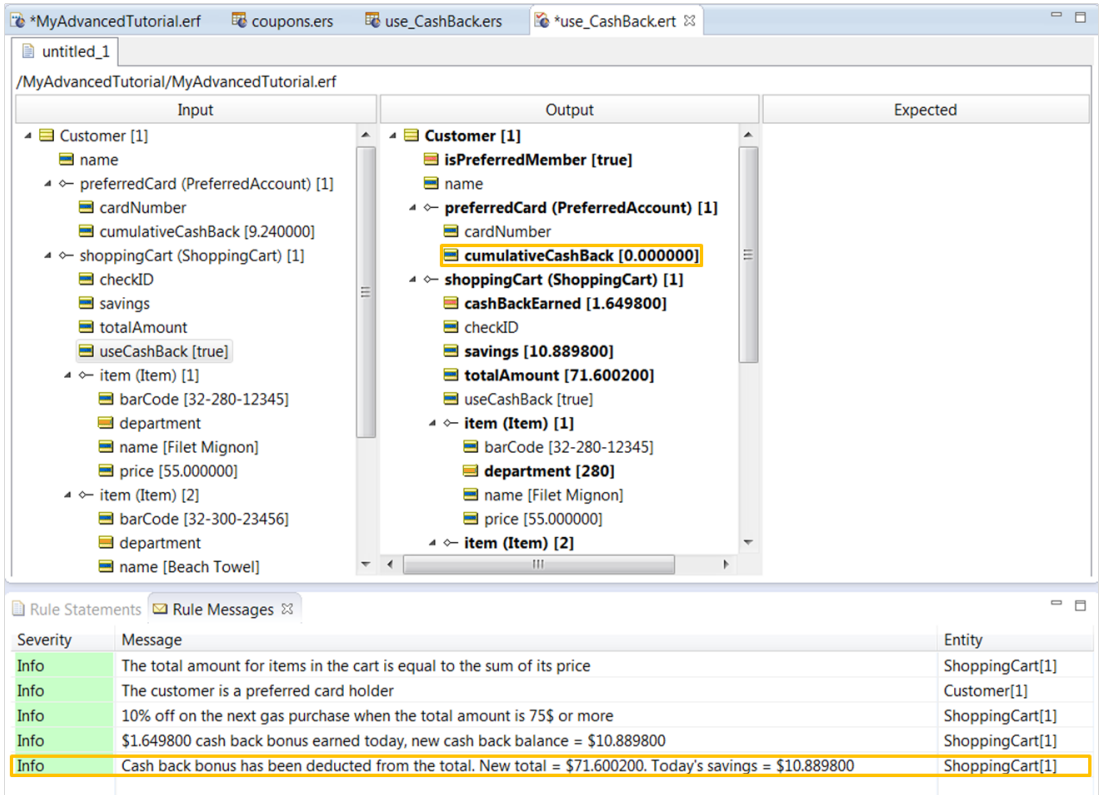
> Once a Preferred Shopper chooses to apply his cash back balance, the cumulative cash back total maintained by the system will be reset to zero, and the accumulation of cash back begins anew with the customer's next purchase.

We need to reset the cumulativeCashBack attribute to 0. Before do this, we want to ensure that our preferred customer is aware of their savings today. Let's assign the value of cumulativeCashBack to the attribute named savings. We'll assume that this savings value will be shared with the customer on the receipt or in some other way. Then, we can reset the cumulativeCashBack value to 0.

Define the actions and add a rule statement as shown.



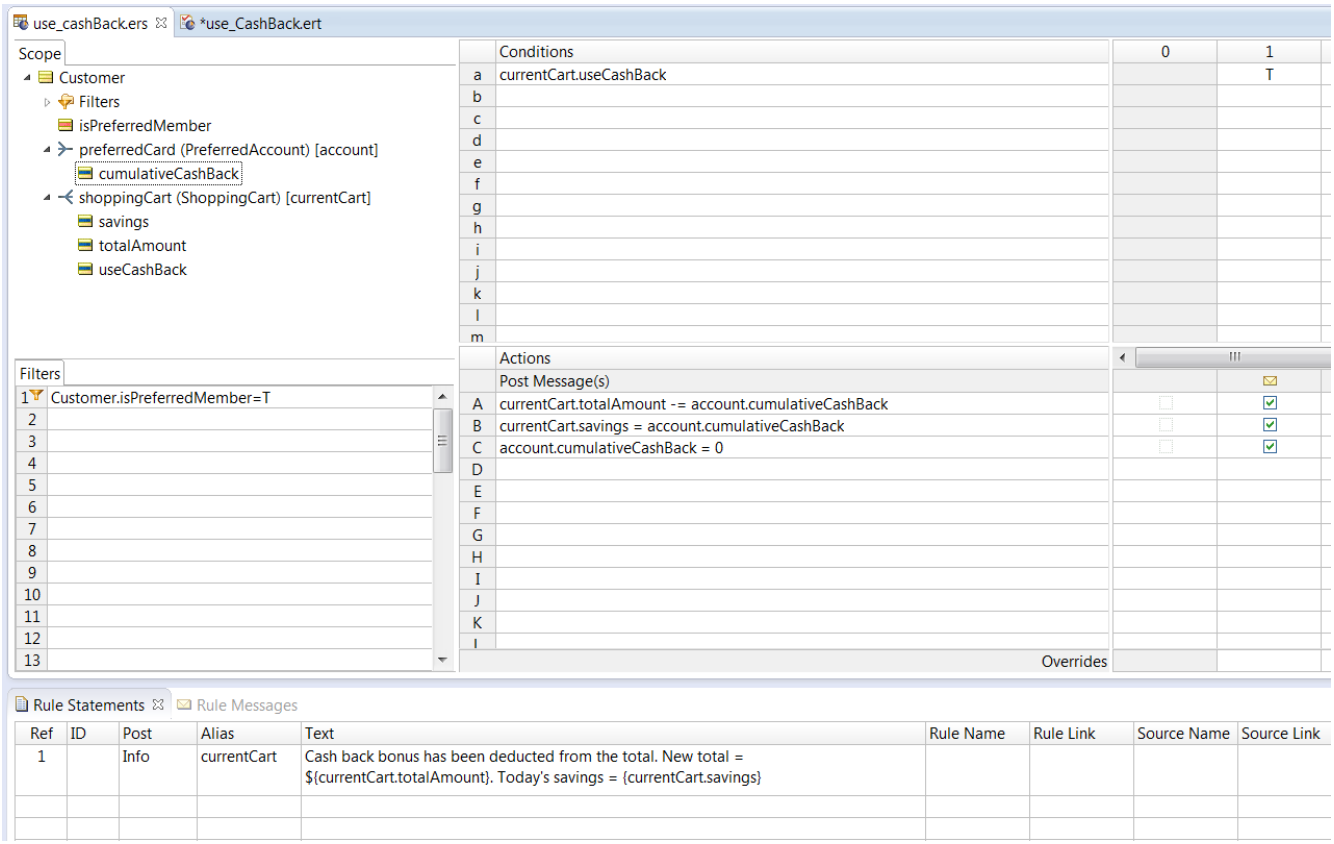
The rule is now complete. Let's test it using the same Ruletest. You do not need to modify or add additional input. Execute use_cashBack.ert again.



As you can see, cumulativeCashBack is now 0, and savings has the value previously held by cumulativeCashBack. There is also a new rule message explaining what has happened. Our final rule works as expected.

Since this was a cumulative test, we can also be assured that the entire Ruleflow (all 3 Rulesheets) works as expected. The business problem has now been fully modeled and tested.

Here is our third and final completed Rulesheet.



A note about logical validation

While these Rulesheets successfully model the scenario's business rules, they are not "complete" from a logical standpoint. Corticon Studio's completeness check will reveal incompleteness in each of the 3 Rulesheets. Identifying and resolving incompleteness or conflicts in these rules is left to you.

7

Tutorial summary

Congratulations on completing the Corticon Advanced Rule Modeling Tutorial! You have learned to incorporate some of Corticon Studio's more powerful functionality into your rule modeling process, including:

Building a Vocabulary – Based on the analysis of a business problem, you've learned how to identify the Vocabulary entities, attributes, and associations that are needed for rule modeling, and build the Vocabulary in Corticon Studio.

Scope and Aliases – Scope tells the Corticon rules engine which data to use when evaluating and executing rules. You've learned to define Scope for a Rulesheet and define an Alias to represent a scope perspective in your rules.

Collections and Collection Operators – A Collection is comprised of one entity associated with one or more other entities, called elements of the collection. Collection Operators operate on groups of entities rather than individual entities. You've learned how to use Collection operators to operate on collections in rules. You also learned that it is mandatory to use Aliases to represent collections.

Action-only rules in column 0 – You've learned how to use column 0 to define non-conditional rules. Column 0 rules can be used to perform calculations that contribute data to other rules in the Rulesheet, or in downstream Rulesheets in the same Ruleflow.

Filters – You've learned how to define Filter expressions to limit data being evaluated to only the subset that survives the filter. 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 – You've learned how to create a Ruleflow, add Rulesheets in a sequence, and test the Rulefllow. 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 you visualize the logic and maintain and reuse them more easily.

Transient Attributes – You've learned how to use Transient attributes in your rules. Transient attributes are used as "intermediate" value holders that do not need to be returned in a response.

Embedding Attributes within Rule Statements – You've learned how to embed attributes within rule statements to make rule messages more meaningful.