Progress®

Corticon Tutorial

Basic Rule Modeling in Corticon Studio

Progress® Corticon®

Copyright

© 2019 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Updated: 2019/11/15

Table of Contents

Tutorial - Basic Rule Modeling in Corticon Studio.....7

Scope the business problem.....9

Discover business rules.....11

Get started with Rule Modeling.....13

Define the Vocabulary.....17

Build a Rulesheet.....21

Model the first rule.....25

Model the second rule.....29

Post rule statements.....33

Analyze rules.....35

Test rule execution.....45

Iterate through the lifecycle again.....55

1

Tutorial - Basic Rule Modeling in Corticon Studio

This tutorial provides an introduction to business rule modeling in Corticon Studio. You will learn how to:

- Capture rules from business specifications,
- Model the rules,
- Analyze them for logical errors, and,
- Test the execution of your rules

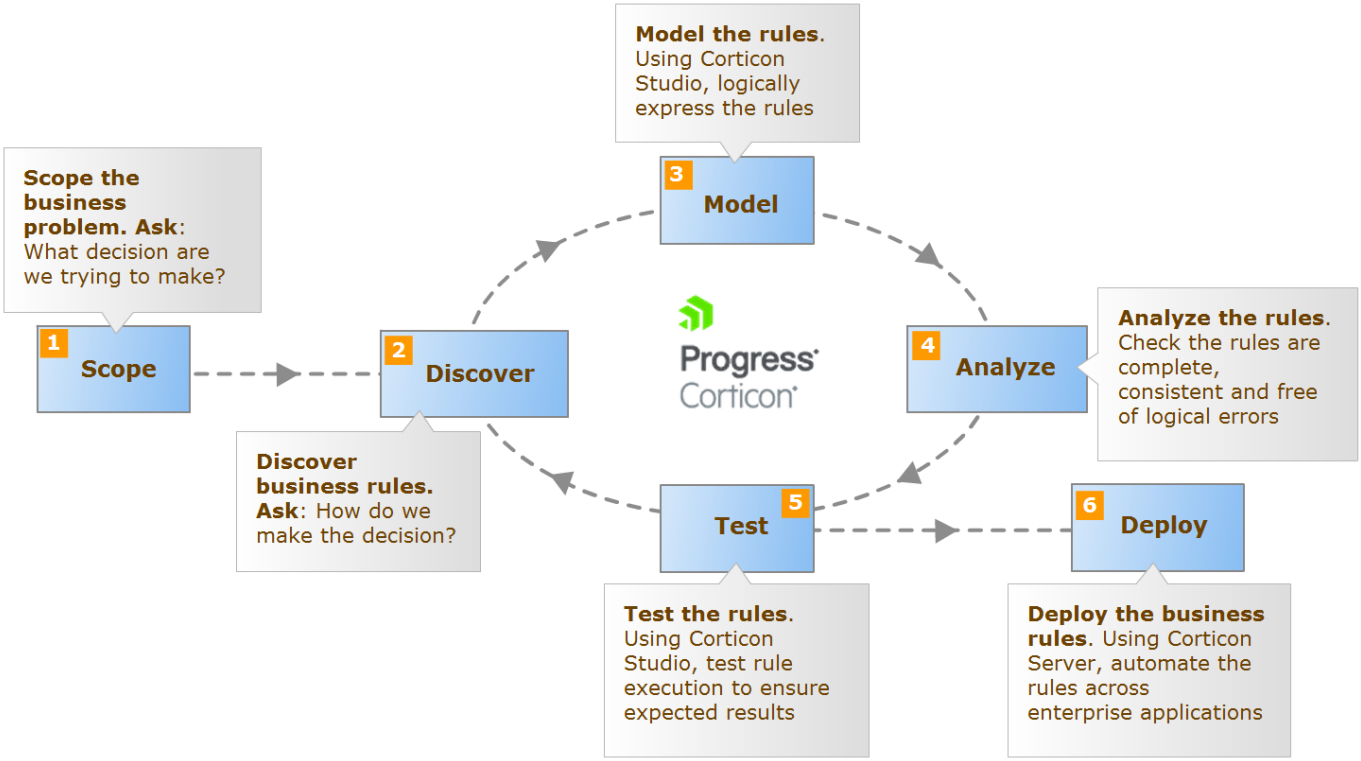
In Corticon, you do all of this without programming.

Your goal in the tutorial is to create a Decision Service. A Decision Service is a group of rules that captures the logic of a single decision-making step in a business process. The Decision Service that you build may be deployed as an executable, standards-based service that can be made available to other software applications using Java, .NET, REST, or SOAP messaging.

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 lifecycle of a Decision Service

Corticon provides a simple yet powerful methodology for modeling business rules. The methodology involves the following steps:



This methodology is iterative. You repeat steps 2 to 6 when rules change.

This tutorial includes two iterations through the lifecycle, including an initial implementation and a subsequent change cycle.

Note: Deployment is not covered in this tutorial. For information on deploying rule models, refer to the Corticon Deployment topics in the Corticon documentation set.

2

Scope the business problem

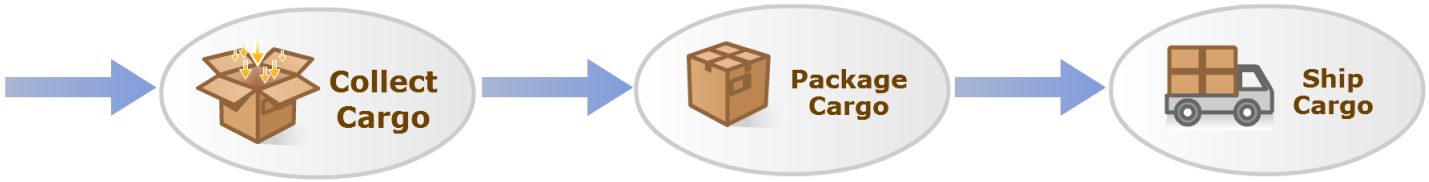
The example used in this tutorial describes a flight planning business problem in which an air cargo company collects cargo of various sizes and weights, loads them into containers, and then places the containers onto its fleet of aircraft prior to shipment.

To operate safely, the company must ensure that an aircraft is never loaded with cargo that exceeds an aircraft's capabilities. Part of the flight planning process involves verifying that no flight violates any safety or operational rule.

The air cargo company wants to improve the quality and efficiency of the flight planning process by modeling and automating business rules using Corticon.

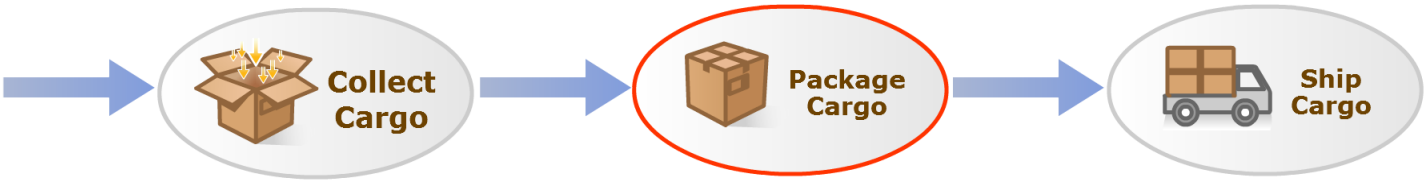
Complex problems such as flight planning are better described in their component parts. The best way to do this is by describing the business process. From a process diagram, we can easily identify the decision-making activities, which in turn are described by business rules.

First, we define the business process as a sequence of activities or steps:



Next, we determine which process steps involve decisions. Any step involving a decision is a candidate for automation using Corticon.

In this process, all three steps involve decisions, in addition to physical labor. The scope of this tutorial is the Package Cargo step, which involves the decision about what container to use for various cargo based upon the cargo's weight and volume.



Today, the packaging decision is made by shipping personnel based upon their experience. The problem is that some people make better decisions than others, which leads to inconsistent practices. We want to use Corticon to standardize and automate the packaging decision.

3

Discover business rules

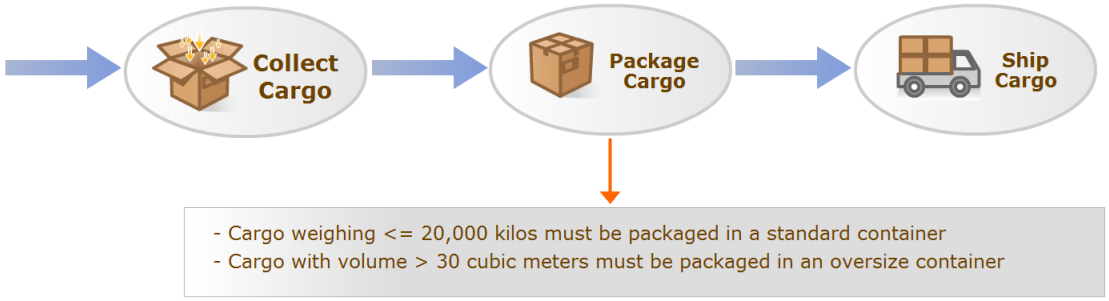
Now that we have scoped our problem, we need to discover our business rules.

To discover our business rules, we ask the question: “How do we make this decision?”

In our flight planning example, we ask “How do we package cargo?”

We ask this question of the people who perform and manage the step in the process. They often provide the answer in the form of a policy or procedure manual, or simply as a set of rules that they follow. Sometimes the rules are embedded in the code of legacy systems. In all cases, we can capture the discovered rules directly into Corticon Studio.

For the packaging decision, let us assume that we have discovered two rules:



4

Get started with Rule Modeling

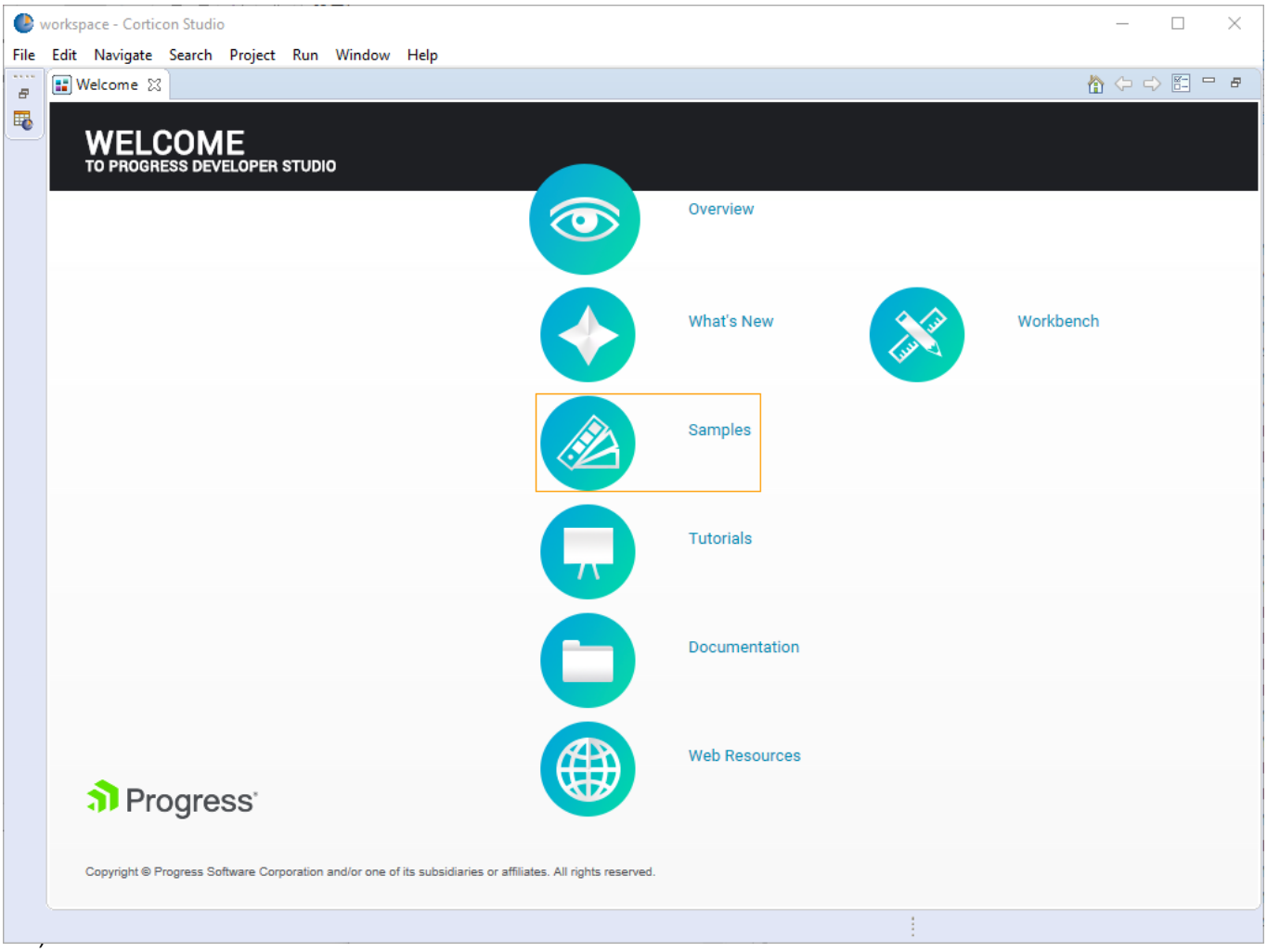
Now that the rules have been discovered, we can model them in Corticon Studio. To get started, launch Corticon Studio by selecting Progress > Corticon Studio.

A workspace is a Windows directory where Corticon Studio rule projects are stored. Retain the default location and click OK in the Workspace Launcher dialog box. This opens Corticon Studio.

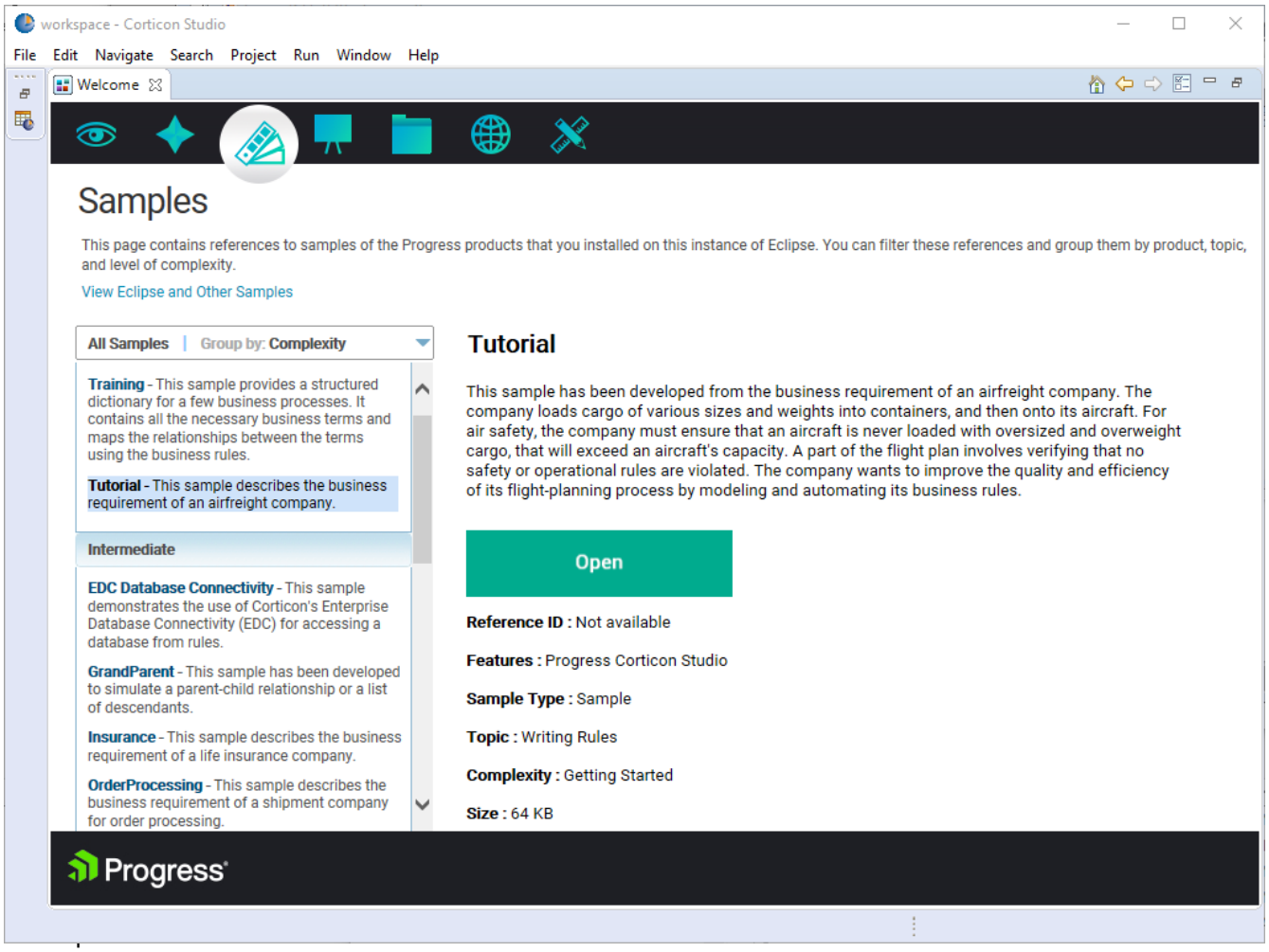
Set up the tutorial rule project

Corticon Studio comes with a built-in sample rule project for this tutorial. To open the sample:

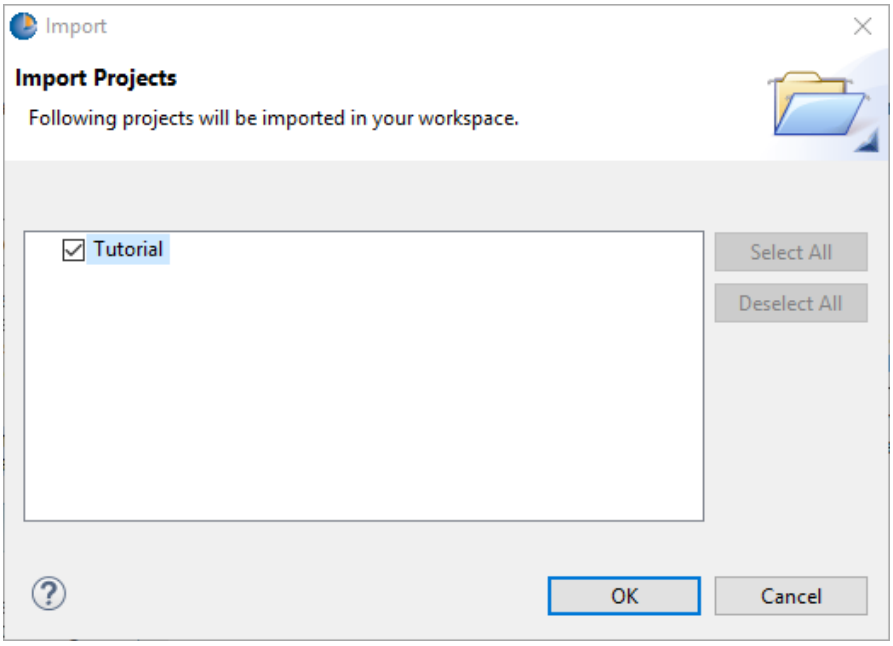
1. Click Samples on the Welcome page. (If you are not able to see the Welcome page right after you launch Corticon Studio, select Help >



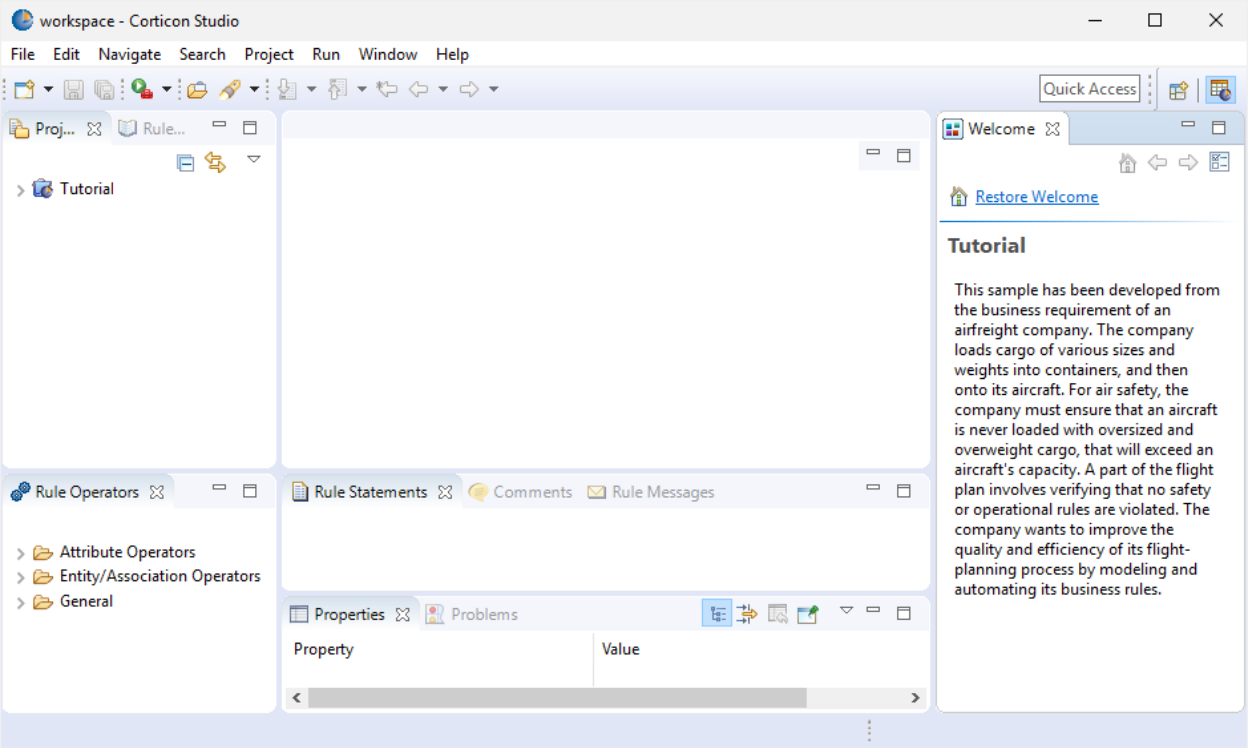
2. On the Samples page, select Tutorial, and then click



3. In the Import Projects window, click OK.



The sample rule project opens in Corticon Studio.



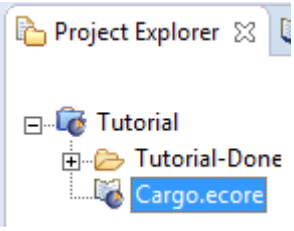
5

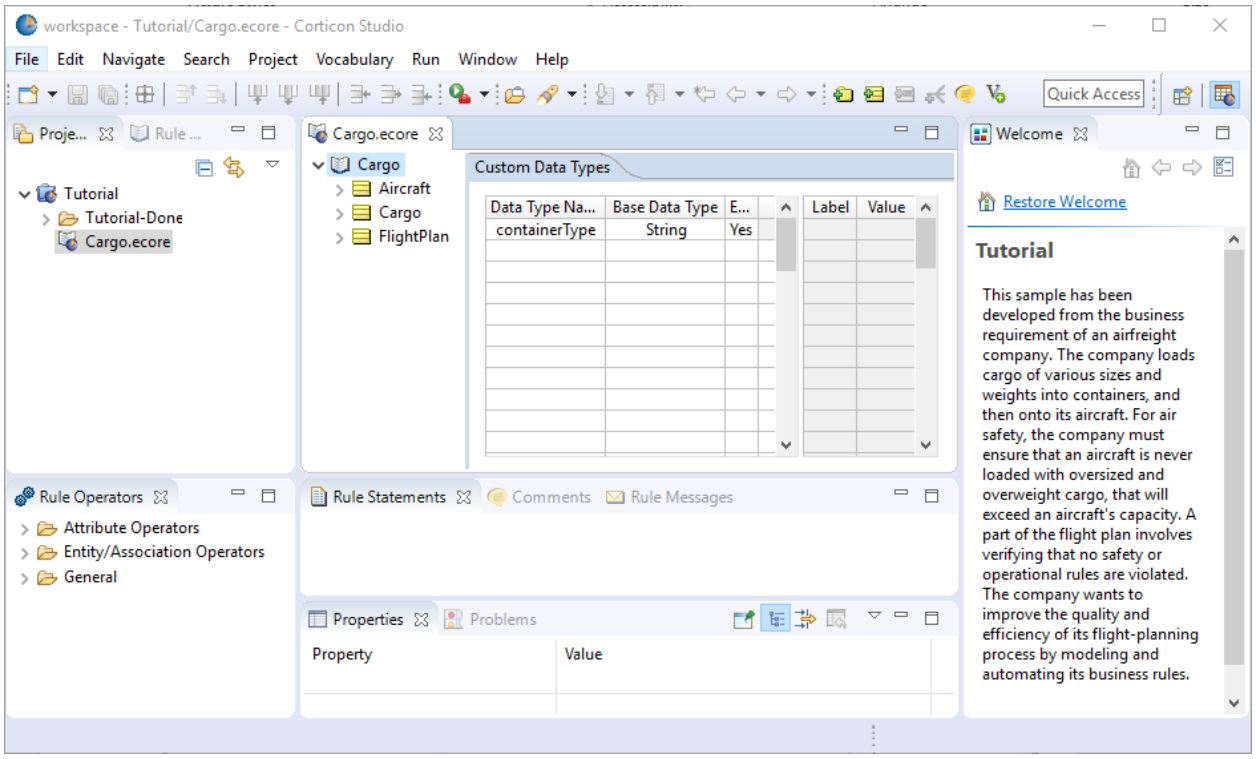
Define the Vocabulary

An important component of all rule modeling projects is the Vocabulary. The Vocabulary includes the terms, called entities, referenced by the business rules. In our example, the Vocabulary would contain entities such as “Cargo” and “Aircraft.” You can use Corticon Studio to define and edit Vocabularies.

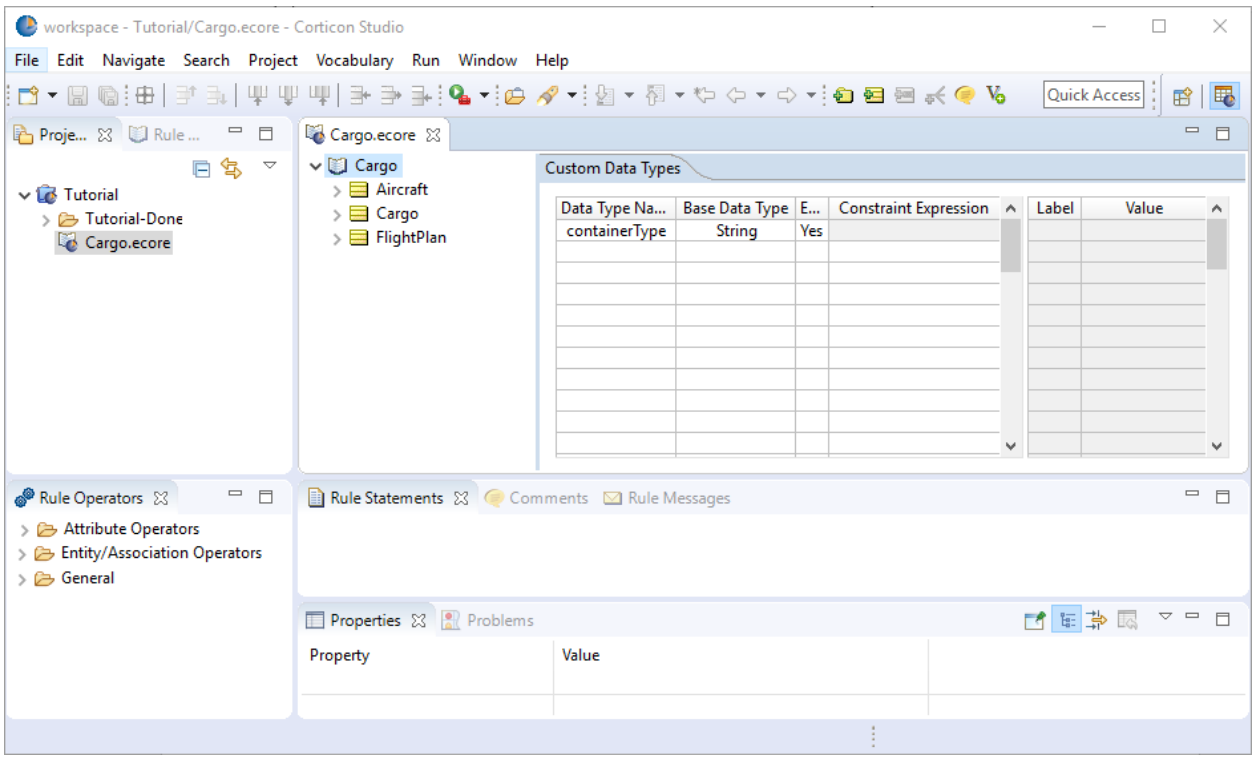
In your project, you may be responsible for defining the Vocabulary or you may simply use an existing Vocabulary that some else defines. In our example, we will use an existing Vocabulary that is part of the Tutorial sample.

To open the sample Vocabulary file, expand the Tutorial Rule Project, and then double-click Cargo.ecore:

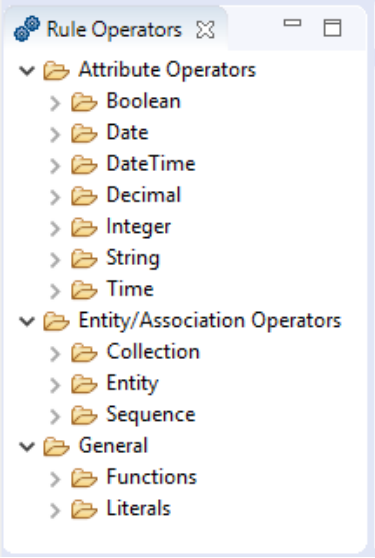




The Vocabulary opens its editor in the area where all the editors will do their work. A feature of the Eclipse development environment lets you resize, move, and close tabs and panels to suit your needs. By closing the Welcome pane and resizing other views, you get more work area.



Take a moment to explore the Rule Operators, which is the view panel in the lower left. Operators act on elements in our Vocabulary like cargo, cargo weight, aircraft, aircraft type. Corticon Studio comes with a rich set of operators for manipulating data, not unlike the Excel function library.

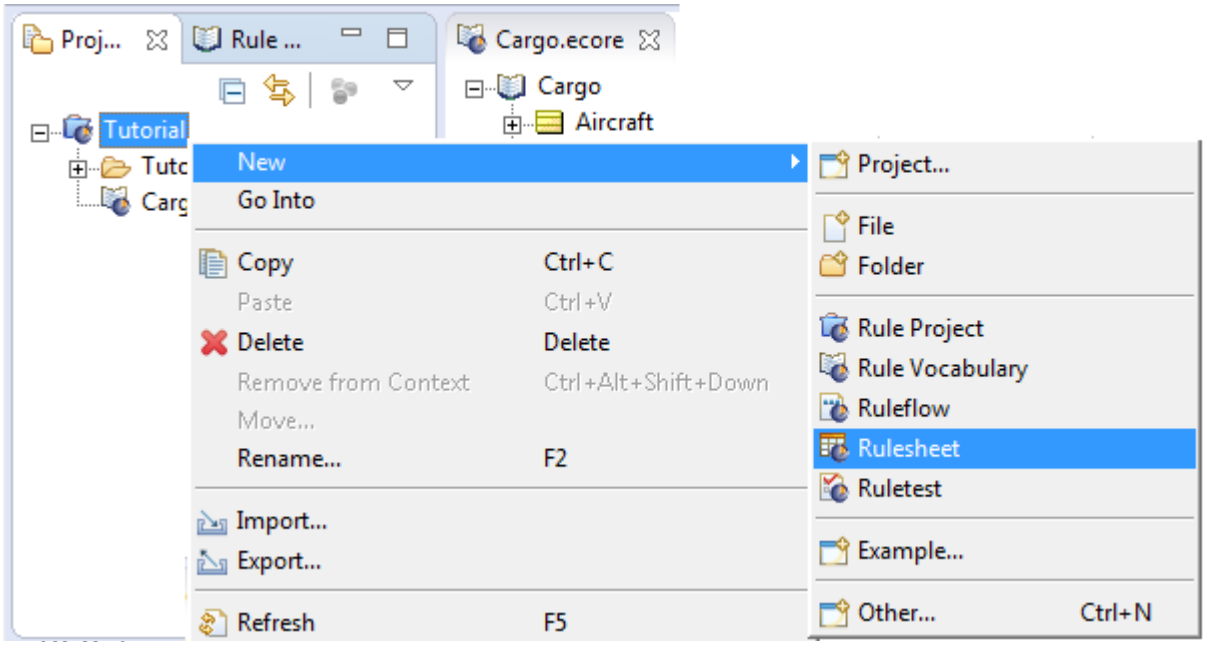

Rule Operators
Attribute Operators
Boolean
Date
DateTime
Decimal
Integer
String
Time
Entity/Association Operators
Collection
Entity
Sequence
General
Functions
Literals

6

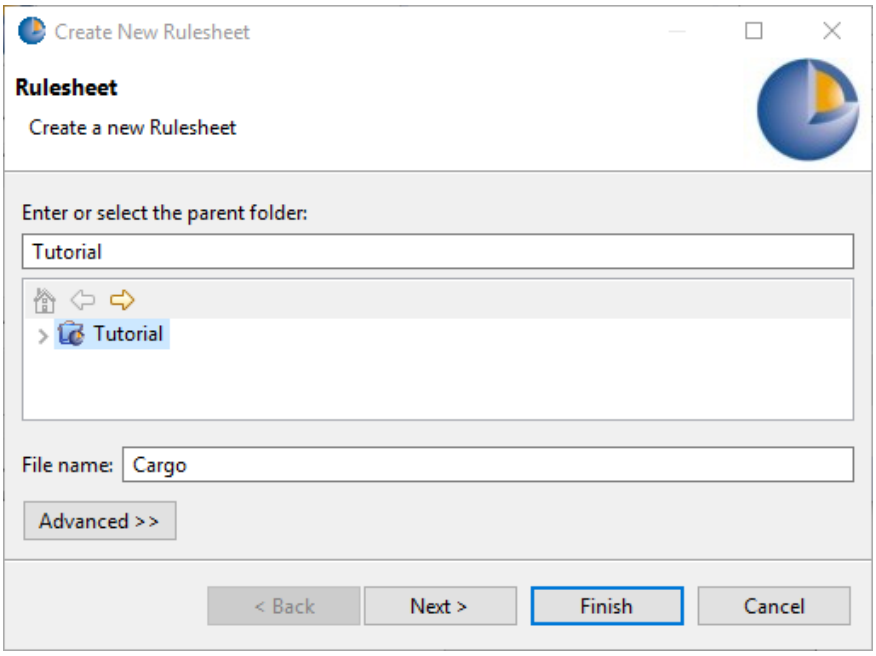
Build a Rulesheet

Next, let's create some rules using the Vocabulary. To begin, we need to create a Rulesheet:

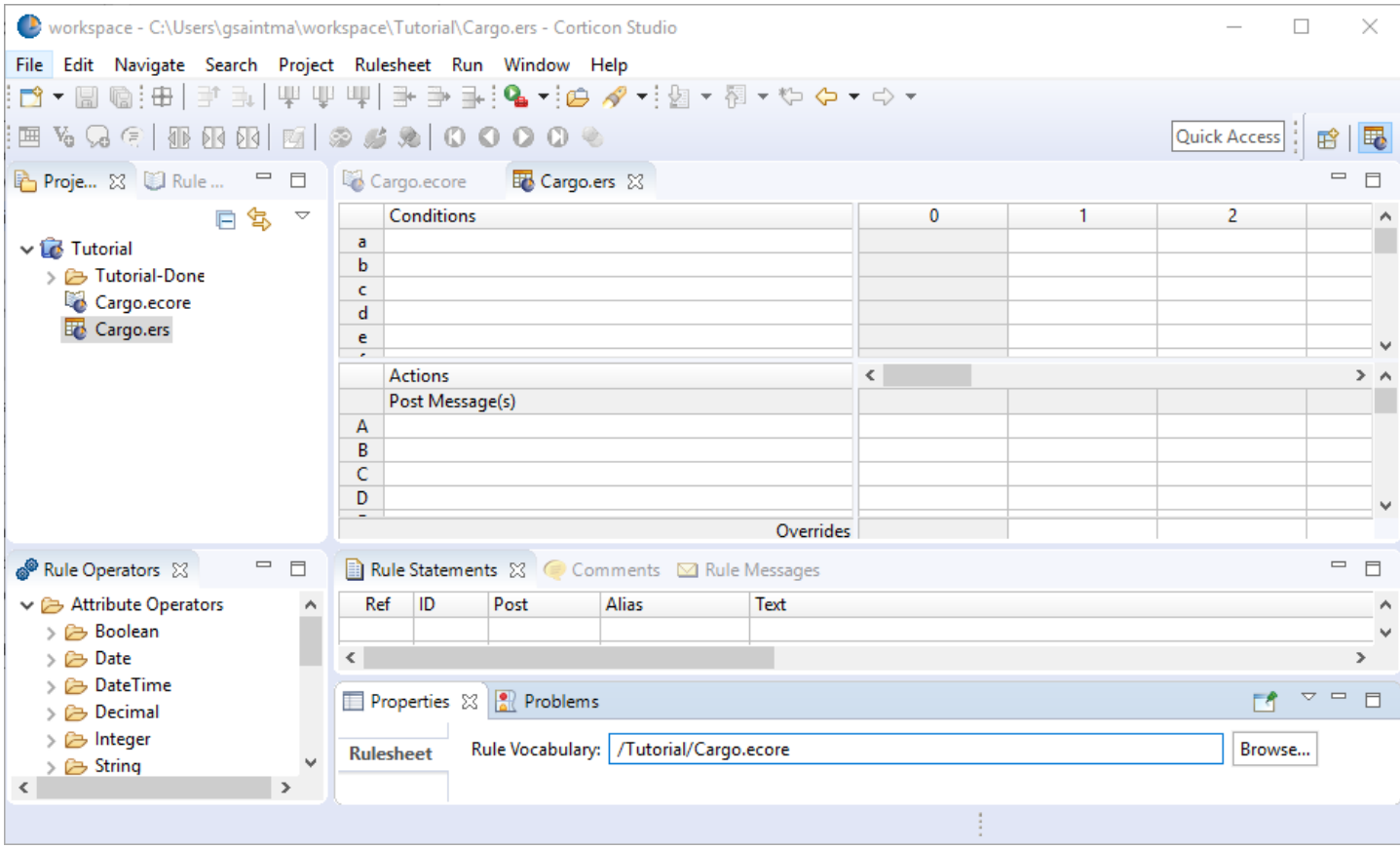
1. In Corticon Studio, right-click Tutorial and select New >



2. In the Create New Rulesheet wizard, enter Cargo as the Rulesheet name and click Next.
3. Ensure that Cargo.ecore (under Tutorial) is selected as the Vocabulary to associate with your new Rulesheet and click Finish.

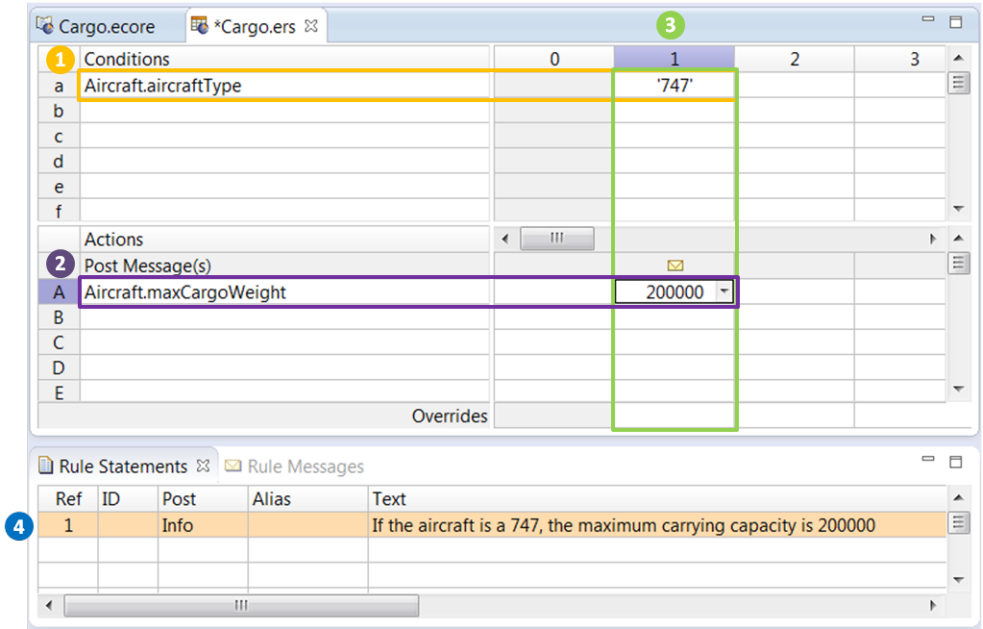


The Rulesheet opens in Corticon Studio.



A Rulesheet is like a decision table or spreadsheet. You can extend Rulesheets to model any kind of business logic—from simple to complex.

Here is an example of a Rulesheet with a rule:



Rulesheets contain sections for specific parts of rules. Sets of Conditions 1 and Actions 2 that are tied together by vertical columns form rules.

Each column is a rule. For example, the rule in column 1 3 can be read as “if the aircraft is a 747, its maximum cargo weight is 200,000’.

This column provides the model or implementation of the rule statement 4 .

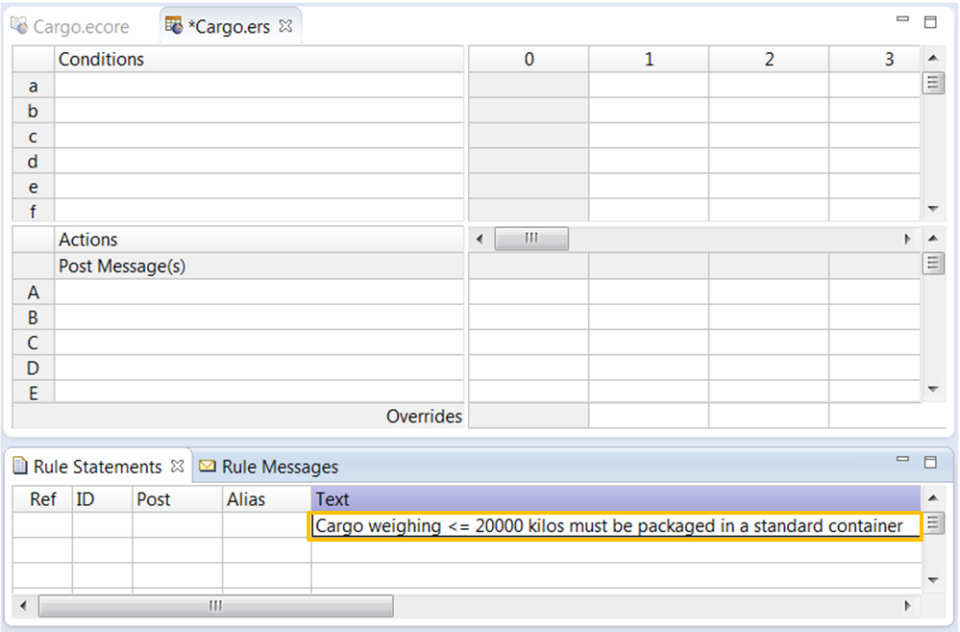
7

Model the first rule

Let's model the first rule: Cargo weighing <= 20,000 kilos must be packaged in a standard container.

Step 1—Defining the rule statement

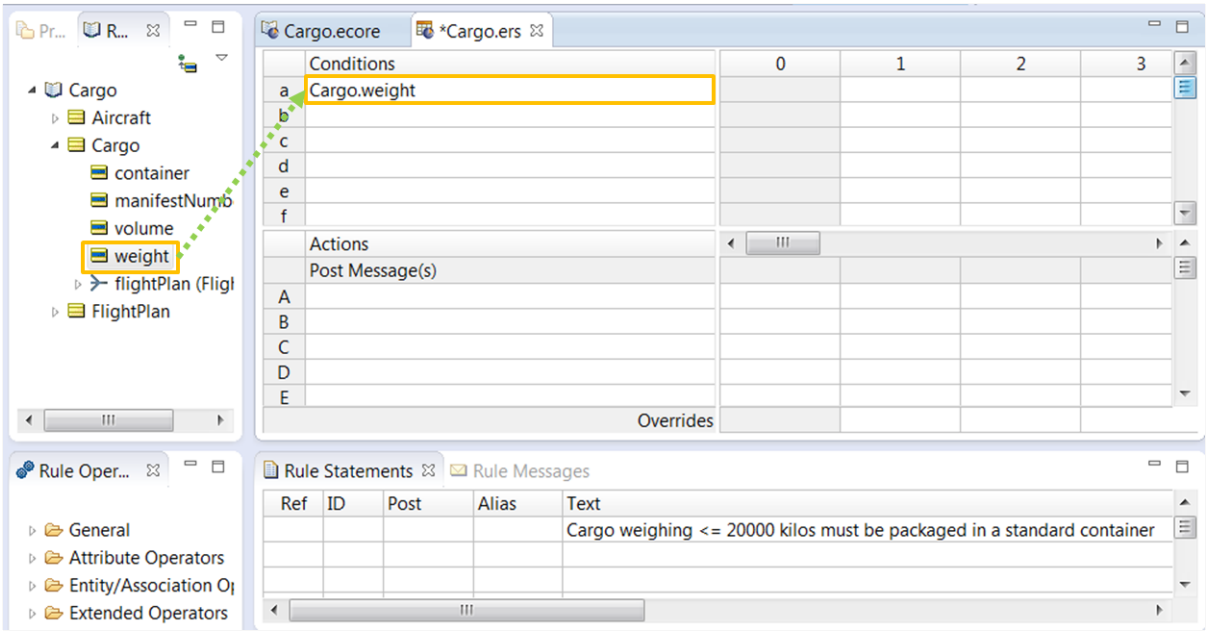
Enter the plain-language business rule into the Rule Statements section of the Rulesheet as shown.



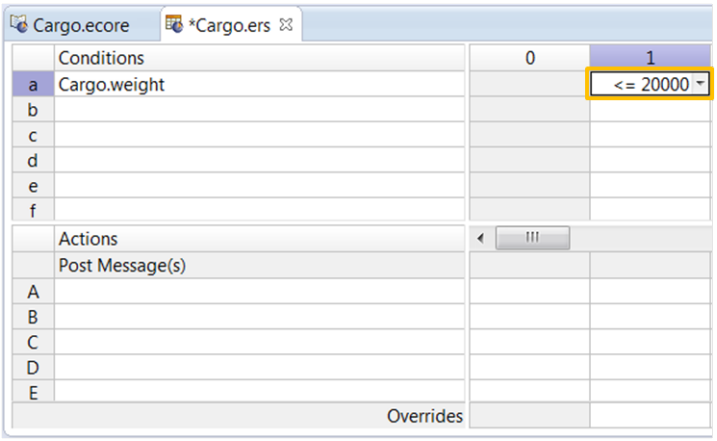
Step 2—Define rule conditions

Let's model the condition of our first rule—Cargo weighing <= 20,000. This condition can be represented as Cargo.weight <= 20000.

1. Click the Rule Vocabulary tab, expand the Cargo entity, and then drag weight from the Vocabulary and drop it in Row a of the Conditions pane.



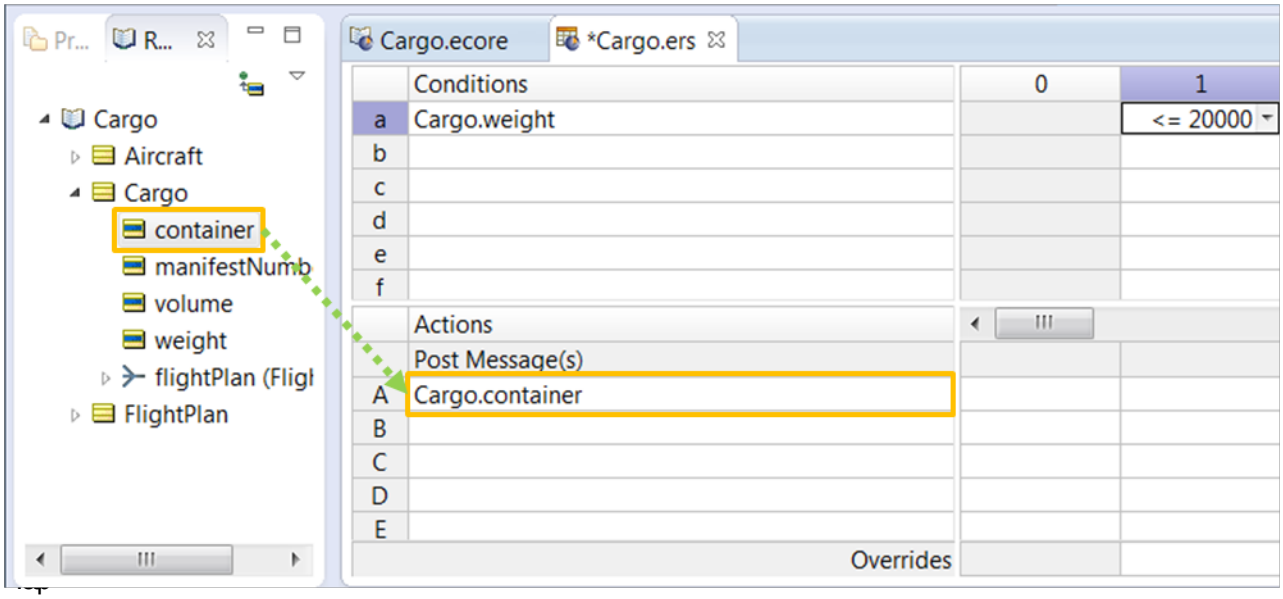
2. Next, we need to specify a value expression for cargo weight in cell 1a (row a, column 1). Enter <= 20000 in the cell. Note: Don't use commas in value expressions because commas are used to indicate multiple values in Corticon rule modeling.



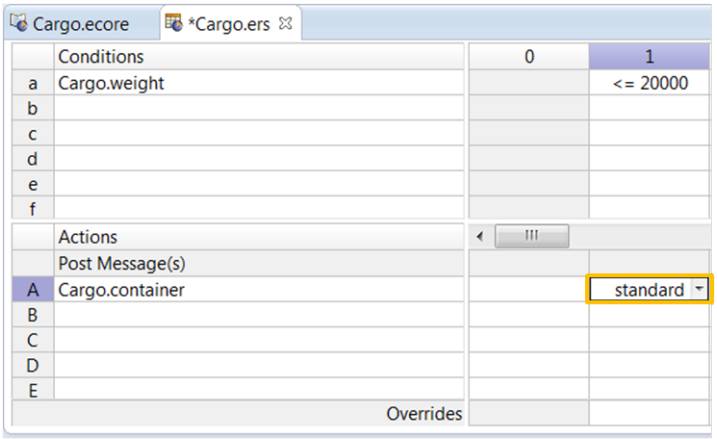
Step 3—Define rule actions

Now, we need to define the second part of the rule (Cargo weighing <= 20,000 kilos must be packaged in a standard container). To do this, we define the rule action:

1. Drag and drop Cargo with volume >container from the Vocabulary and drop it in Row A of the Actions



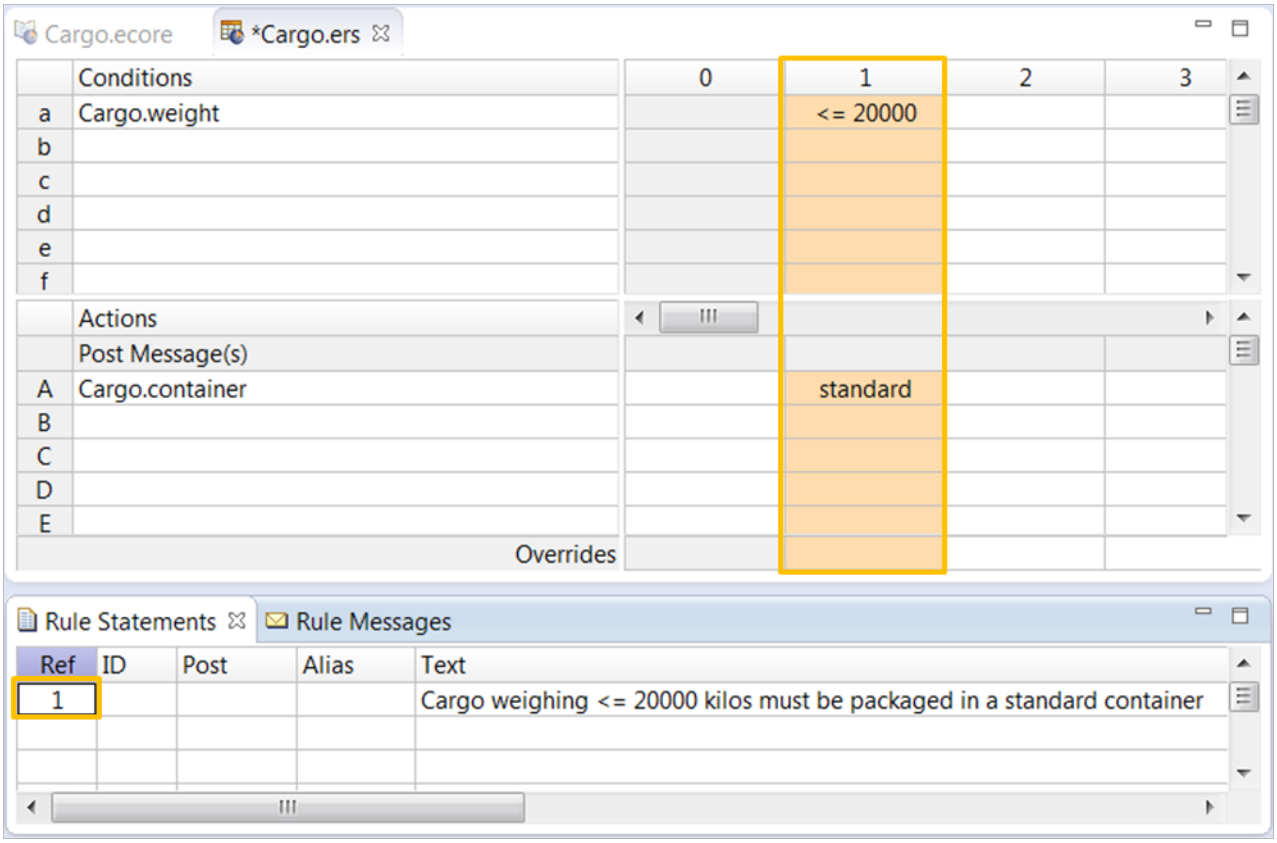
2. Next, we need to specify a value expression for cargo container in cell 1A (row A, column 1). Select “standard” in the drop-down menu in the cell. Note: The drop-down menu options were defined in the Vocabulary, which we will edit later. This action assigns the value of “standard” to Cargo.container.



Step 4—Link the rule statement to the rule

Now that we have defined the rule, we link the rule statement to the rule. The rule statement is used when the rule is executed. If the conditions in the rule are met, the rule ‘fires’, triggering the rule action and sending the rule statement as a message to the application that uses the rule.

To link the rule statement to the rule, enter the Rule Statement’s Reference number. This connects the Rule Statement to the corresponding rule column in the Rulesheet. In our example, the rule is defined in column 1, so enter 1 in the Reference column of the rule statement.



The rule column is highlighted in orange, indicating that the rule is linked to the rule statement.

8

Model the second rule

The first rule is now defined. Define the second rule—Cargo with volume > 30 cubic meters must be packaged in an oversize container—in the same way.

1. Enter the rule in plain language in the Rule Statements section.
2. Drag volume from the Vocabulary to row b in the Conditions pane.
3. Specify the value expression for cargo volume in cell 2b (row b, column 2). Enter > 30 in the cell.
4. Define the action. Select oversize from the drop-down in cell 2A (row A, column 2).
5. Link the rule statement with the rule.

The final result should look like this:

Cargo.ecore | *Cargo.ers

	Conditions	0	1	2	3
a	Cargo.weight		<= 20000		
b	Cargo.volume			> 30	
c					
d					
e					
f					

	Actions				
	Post Message(s)				
A	Cargo.container		standard	oversize	
B					
C					
D					
E					
	Overrides				

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
1				Cargo weighing <= 20000 kilos must be packaged in a standard container
2				Cargo with volume > 30 cubic meters must be packaged in an oversize conta

Now, save your Rulesheet by selecting File > Save from the Corticon Studio menu bar.

When saved, Corticon Studio places a dash in the empty cells, meaning that the condition is ignored. The conditions and actions in a column are “anded” together. For example, the logic in Rule 1 is modified to: “Cargo weighing <= 20,000 kilos, regardless of volume, must be packaged in a standard container.”

Cargo.ecore | Cargo.ers

	Conditions	0	1	2	3
a	Cargo.weight		<= 20000	-	
b	Cargo.volume		-	> 30	
c					
d					
e					
f					
	Actions				
	Post Message(s)				
A	Cargo.container		standard	oversize	
B					
C					
D					
E					
	Overrides				

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
1				Cargo weighing <= 20000 kilos must be packaged in a standard container
2				Cargo with volume > 30 cubic meters must be packaged in an oversize conta

9

Post rule statements

Finally, you should post rule statements to a Vocabulary entity. This provides an audit trail during rule execution, which you will see during rule testing. In our example, we post rule statements to the Cargo entity.

To post, first select the appropriate severity level from the drop-down in each rule statement's Post column. In this case, we select "Info". Next, we select an Alias for each rule statement. The Alias defines what entity the rule statement is posted to. In this case, we select "Cargo", which is the only option, because our rule conditions and actions only use Cargo.

When you post a rule statement, an icon appears in the Post Message(s) row of the linked rule column.

Cargo.ecore | *Cargo.ers

	Conditions	0	1	2	3
a	Cargo.weight		<= 20000	-	
b	Cargo.volume		-	> 30	
c					
d					
e					
f					

	Actions				
	Post Message(s)				
A	Cargo.container		standard	oversize	
B					
C					
D					
E					
	Overrides				

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
1		Info	Cargo	Cargo weighing <= 20000 kilos must be packaged in a standard container
2		Info	Cargo	Cargo with volume > 30 cubic meters must be packaged in an oversize conta

10

Analyze rules

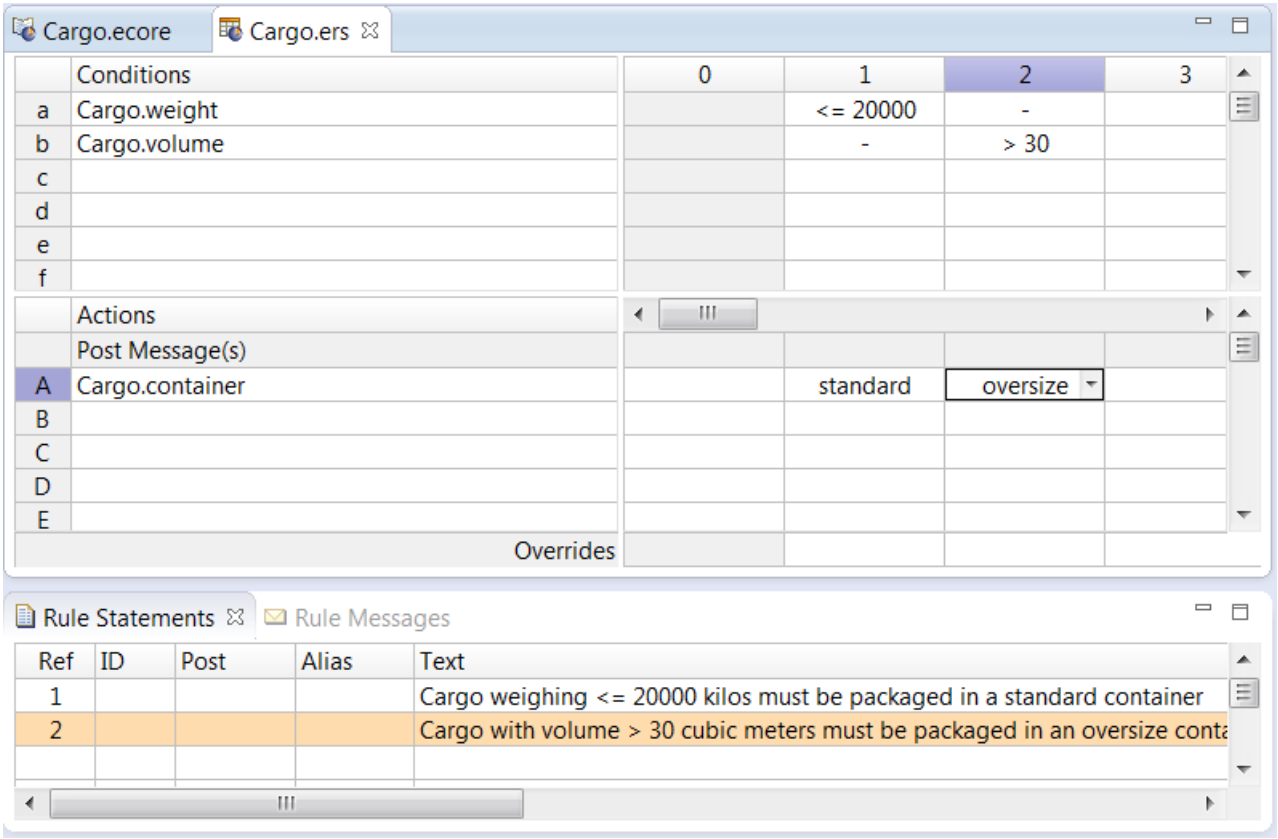
Now that we have finished modeling our rules, we should analyze our rules for logical errors. Often, initial business rule specifications are ambiguous and incomplete. By ambiguous, we mean that the rules are conflicting under certain scenarios. By incomplete, we mean that the rules fail to address all possible scenarios.

Before we automate our rules, it is critical to eliminate logical errors in order to ensure that our decision service provides correct, consistent results. Corticon Studio provides unique and powerful features to help you ensure that rules are complete and consistent.

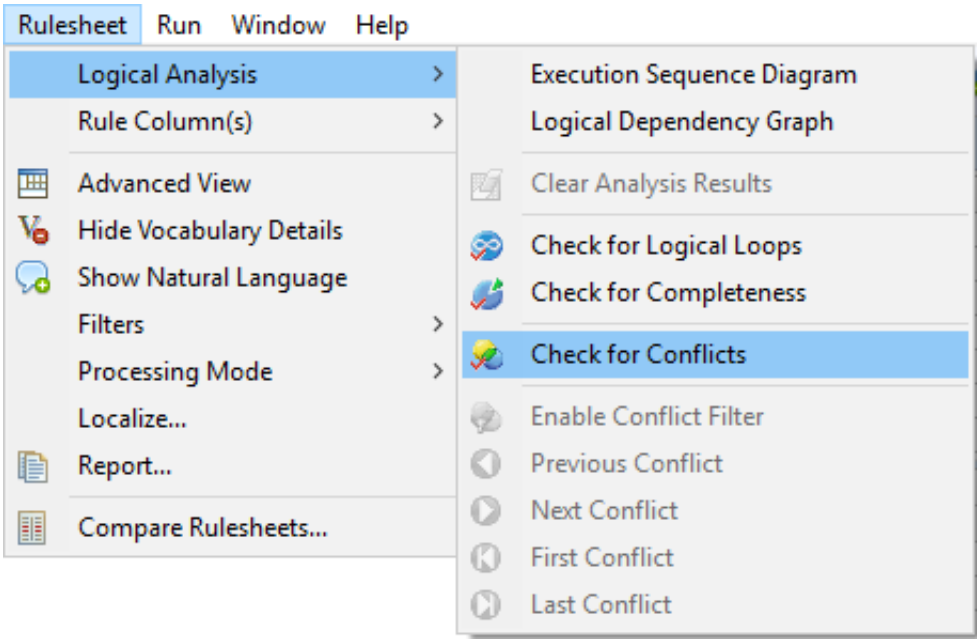
Check for conflicts

Let’s begin by checking for conflicts in our rules:

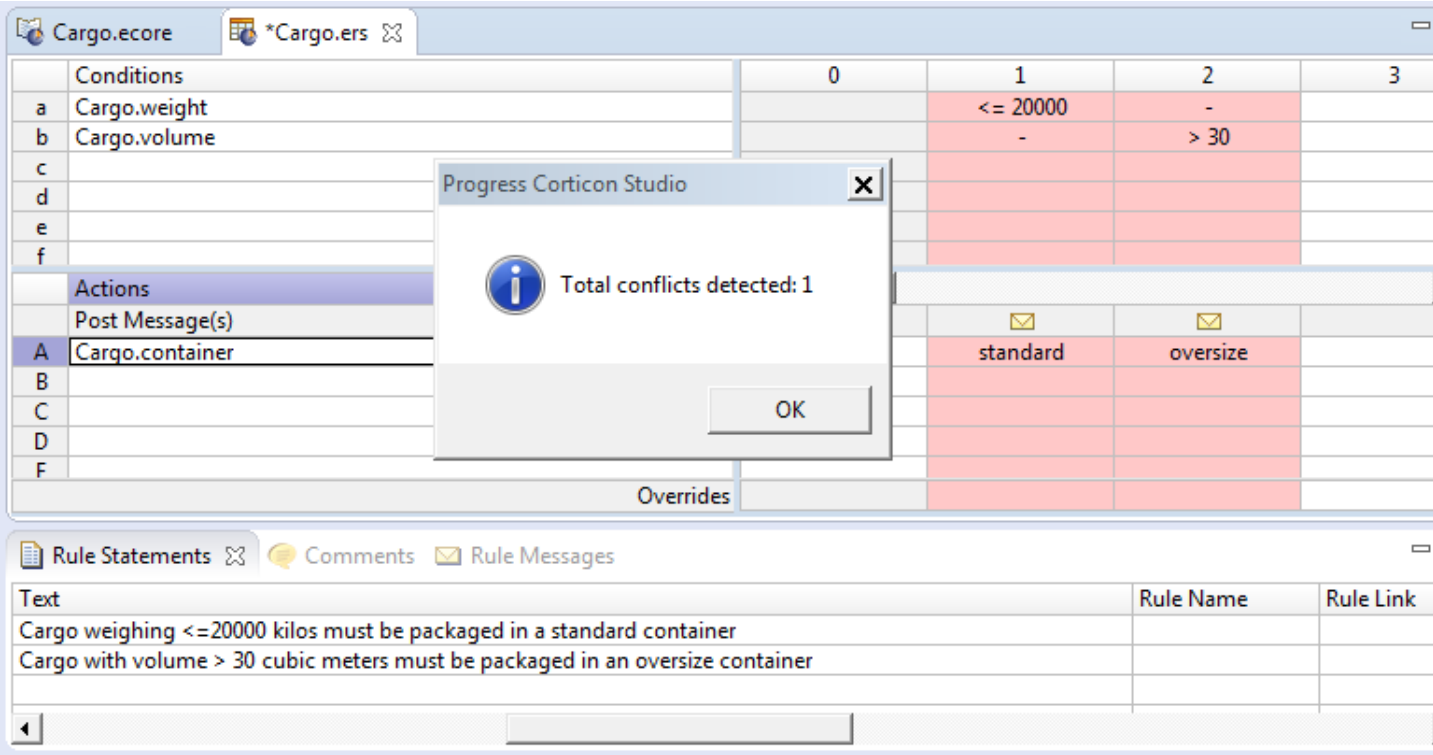
1. Ensure that the Rulesheet is open and that the Cargo.ers tab is selected.



2. Select Rulesheet > Logical Analysis > Check for Conflicts.

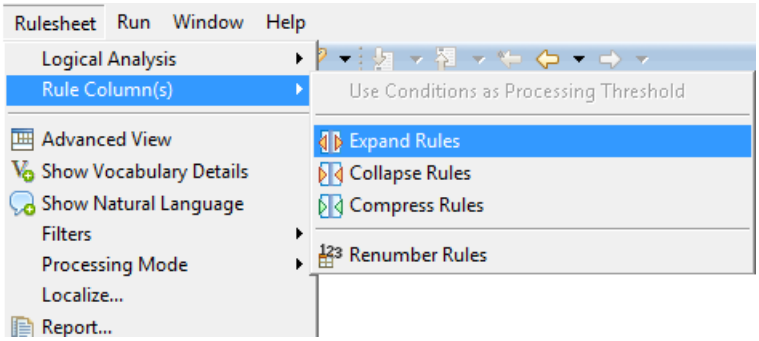


If Corticon Studio detects conflicts, columns containing conflicting rules are shaded in pink. This indicates that one or more conflicts exist between the shaded rules. The total number of conflicts is also displayed in a dialog box. Click OK to dismiss the dialog box.



Resolve conflicts

Sometimes, conflicts may not be immediately visible just by looking at the rules. This is because each rule is actually made up of sub-rules (rules without dashes) and it is the sub-rules that are in conflict. To see these sub-rules, select Rulesheet > Rule Column(s) > Expand Rules.



This helps you pinpoint the source of the conflict.

Cargo.ecore | *Cargo.ers

	Conditions	0	1.1	1.2	1.3	2.1	2.2	2.3
a	Cargo.weight		<= 20000	<= 20000	<= 20000	<= 20000	> 20000	null
b	Cargo.volume		<= 30	> 30	null	> 30	> 30	> 30
c								
d								
e								
f								
	Actions							
	Post Message(s)							
A	Cargo.container		standard	standard	standard	oversize	oversize	oversize
B								
C								
D								
E								
	Overrides							

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text	R
1		Info	Cargo	Cargo weighing <= 20,000 kilos must be packaged in a standard container	
2		Info	Cargo	Cargo with volume > 30 cubic meters must be packaged in an oversize container	

Rule 1 is expanded into three columns, 1.1, 1.2, and 1.3, and rule 2 is expanded into three columns 2.1, 2.2, and 2.3. The expansion shows all of the logical possibilities for each rule. Rule 1 states Cargo weighing <= 20,000 kilos, regardless of volume, must be packaged in a standard container. Corticon Studio recognizes that there are three possible ranges for Cargo.volume (<=30, >30, and null), which we see in the expanded rules.

With the rules expanded, the source of the conflict becomes obvious. Scenarios with Cargo.weight <=20000 and Cargo.volume > 30 are in conflict, since they define mutually exclusive actions (rule 1.2 assigns a standard container while rule 2.1 assigns an oversize container). To get your rules right, this conflict must be addressed.

To resolve the conflict, you can either change your original rules, or decide that one rule should override the other. Let's implement the override:

1. Collapse your rules back to the original state by selecting Rulesheet > Rule Column(s) > Collapse Rules.
2. Override Rule 1 with Rule 2. In the Overrides cell in Rule 2, select the column number of the rule that you want Rule 2 to override—in this case, Rule 1.

Cargo.ecore | *Cargo.ers

	Conditions	0	1	2
a	Cargo.weight		<= 20000	-
b	Cargo.volume		-	> 30
c				
d				
e				
f				
	Actions			
	Post Message(s)			
A	Cargo.container		standard	oversize
B				
C				
D				
E				
	Overrides			1

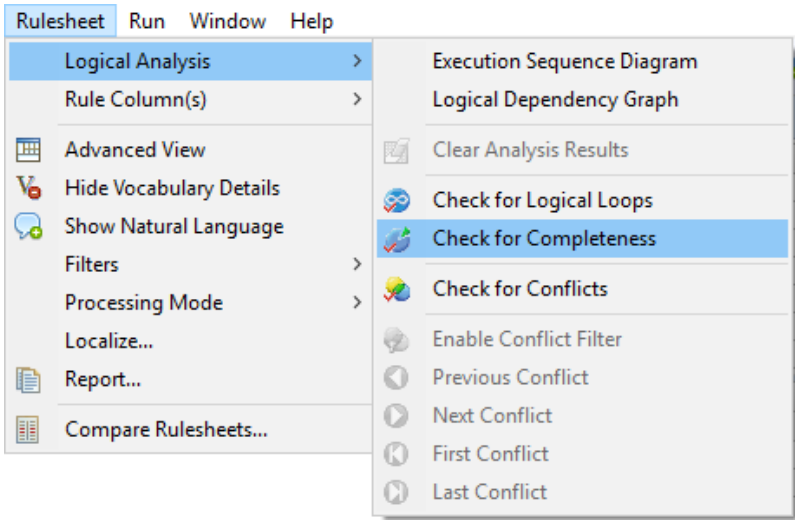
3. Check for conflicts again by selecting Rulesheet > Logical Analysis > Check for Conflicts. You will see that the conflict has been resolved. With the override, Rule 2 now means “Use oversized containers when volume is >30, even when weight is <=20,000.”

Dismiss the dialog box by clicking OK. Save your Rulesheet. Click on the Save icon on the toolbar or choose File>Save.

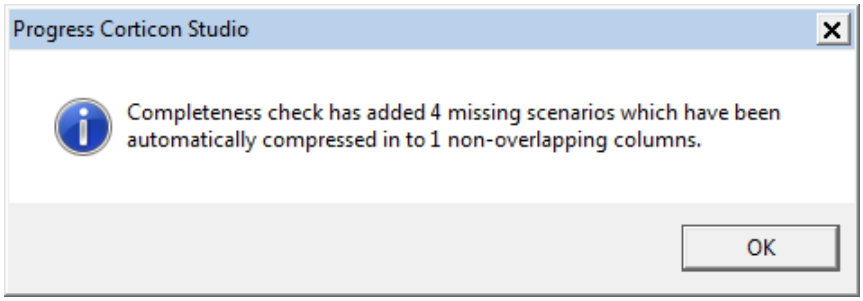
Check for completeness

Conflict is one form of logical error. Another form is incompleteness in our logic.

To see if our rules are complete, select Rulesheet > Logical Analysis > Check for Completeness.



A message window opens informing us that the rules are incomplete. We missed some scenarios.



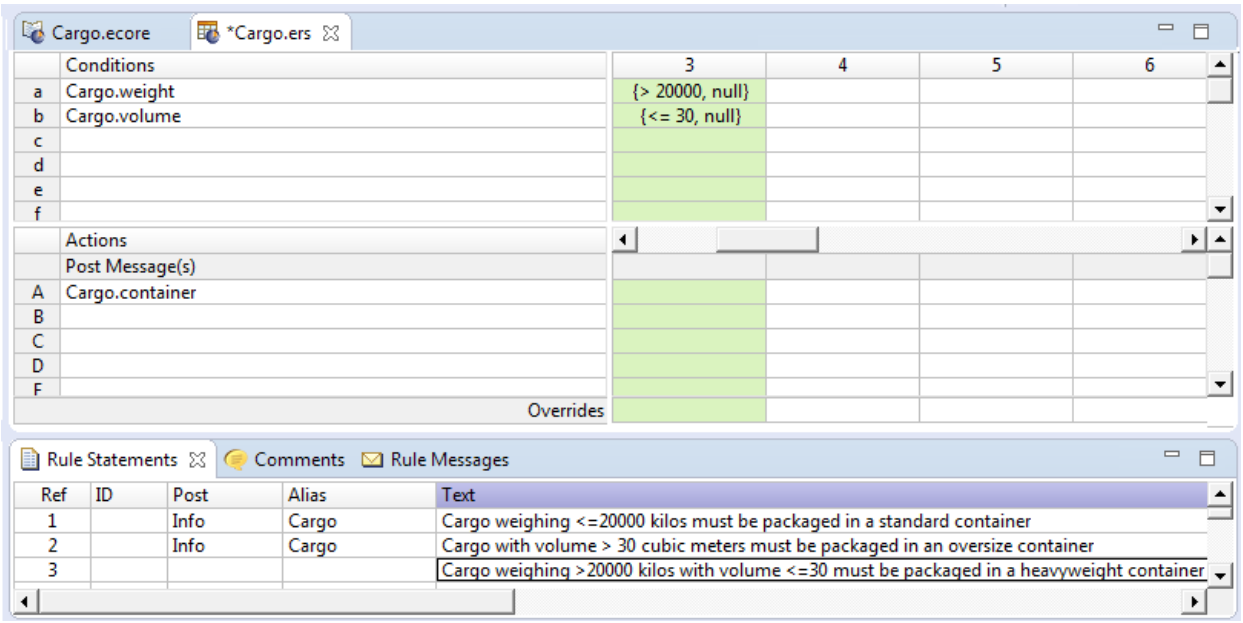
The completeness checking algorithm calculates the set of all possible combinations of values in all conditions. The algorithm then compares this set of possible combinations to those already specified in the Rulesheet and automatically inserts missing combinations of conditions as new columns. These new columns are shaded in green.

In this case, Corticon Studio has added a new rule in column 3—where the cargo weighs > 20000 and the cargo volume is less than or equal to 30. The completeness check adds condition values, but does not choose actions—that is left to the rule modeler.

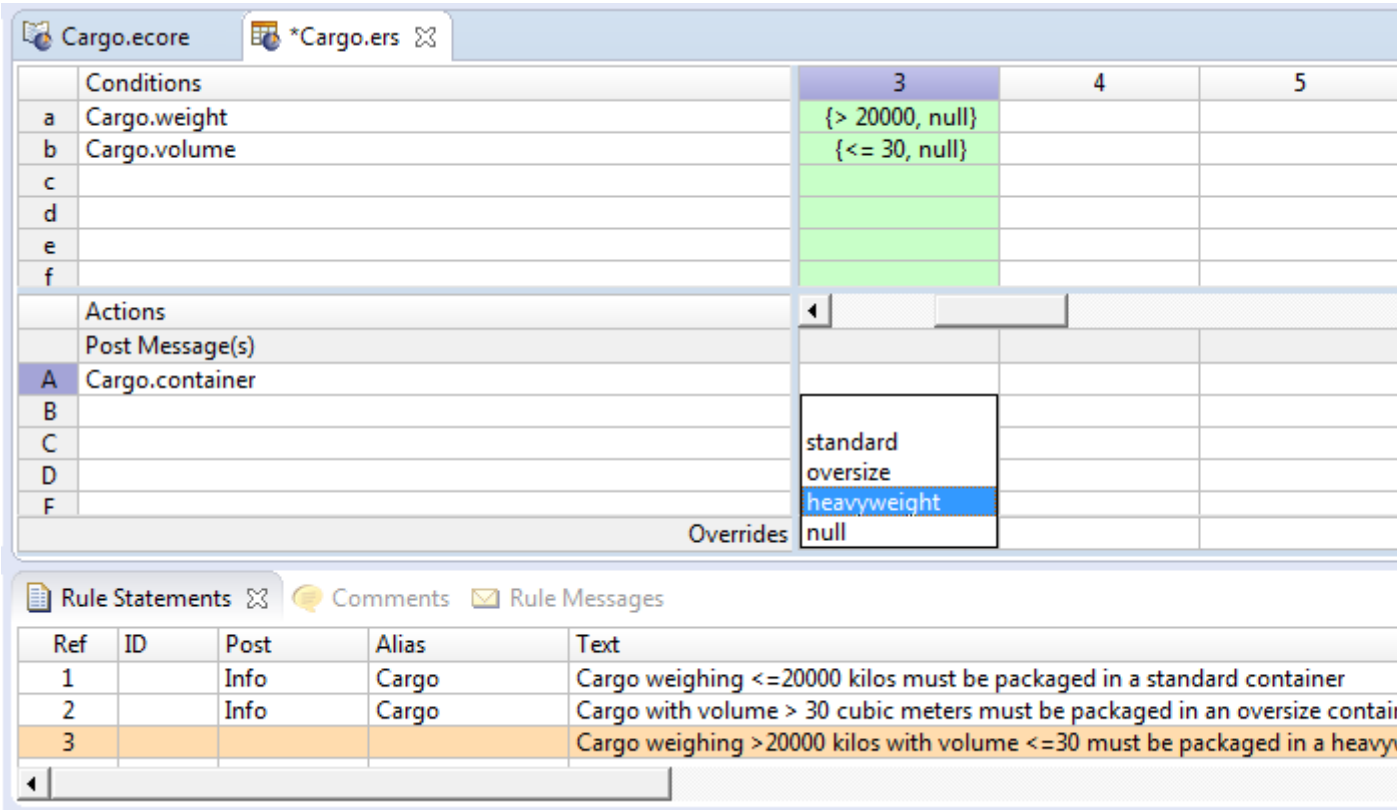
Click OK to dismiss the window.

Resolve completeness errors

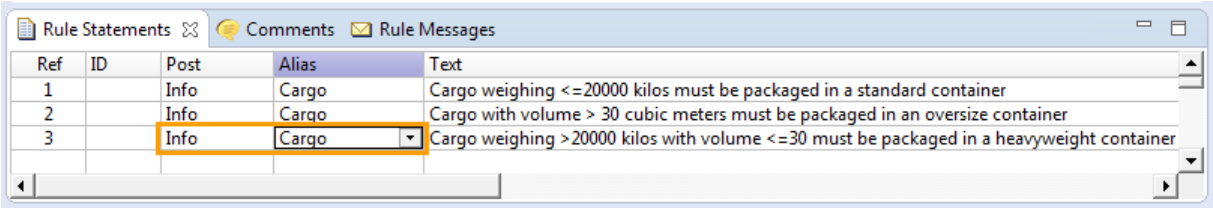
1. We begin by adding a new rule statement for Rule 3: Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be packaged in a heavyweight container. Note: Don't forget to link the rule statement with the corresponding column.



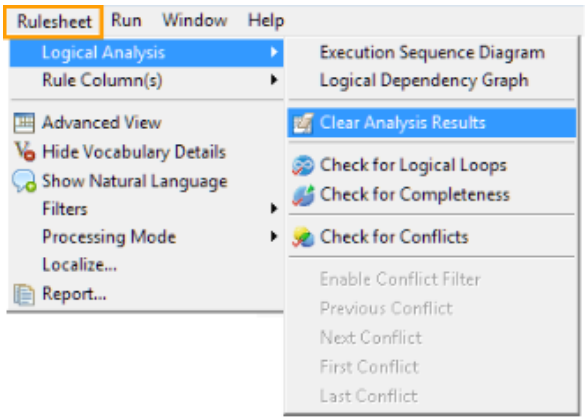
2. Define an action in rule cell 3A. In this case, select "heavyweight" as the container option.



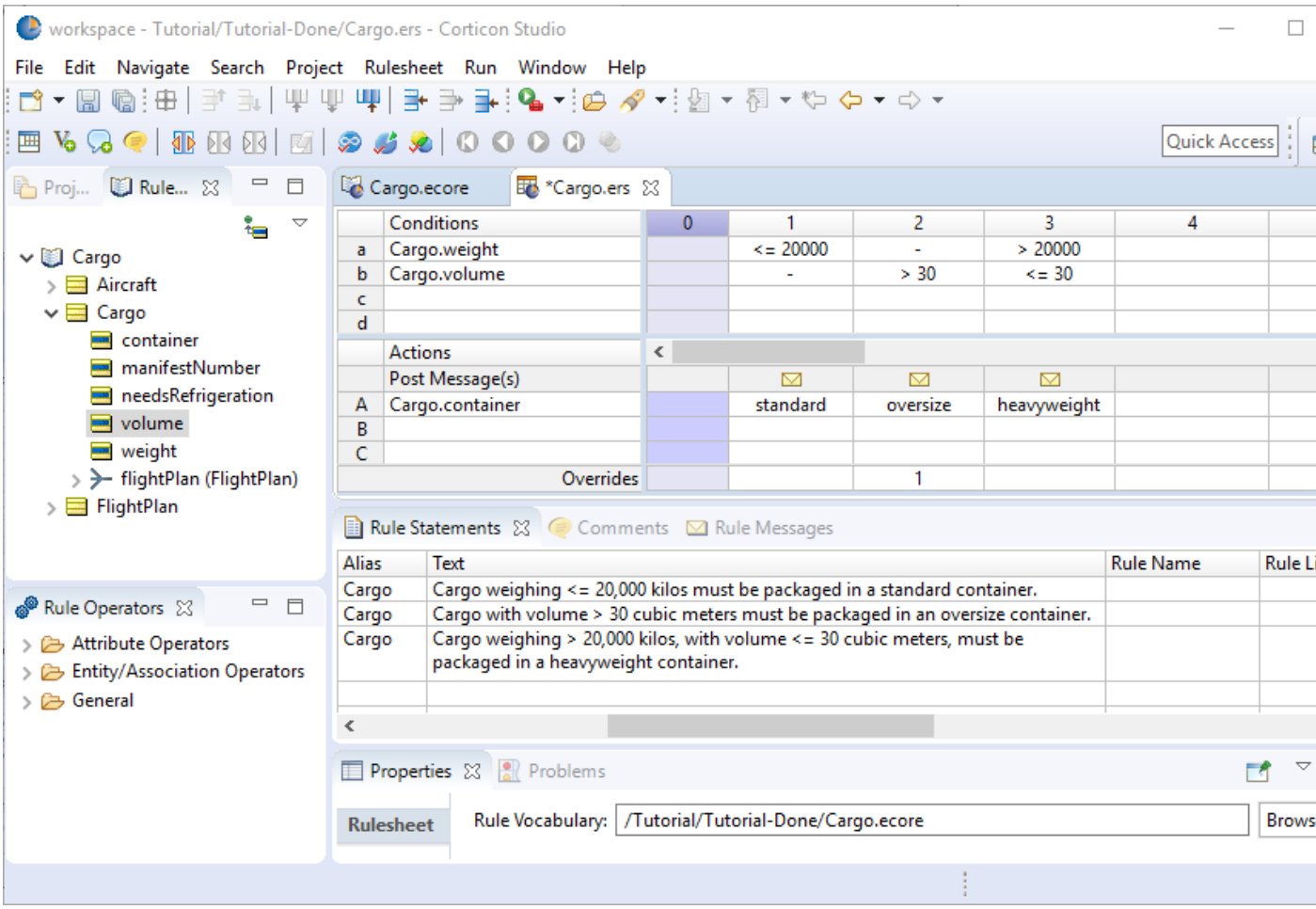
3. Post an Info message to the Cargo entity as we did for the first two rules in the Rulesheet.



4. Select Rulesheet > Logical Analysis > Clear Analysis Results. This removes the highlighting in Rule 3.



After you clear analysis results, your Rulesheet will look like this.



5. Run the completeness check again. The dialog box should indicate that the Rulesheet is complete.

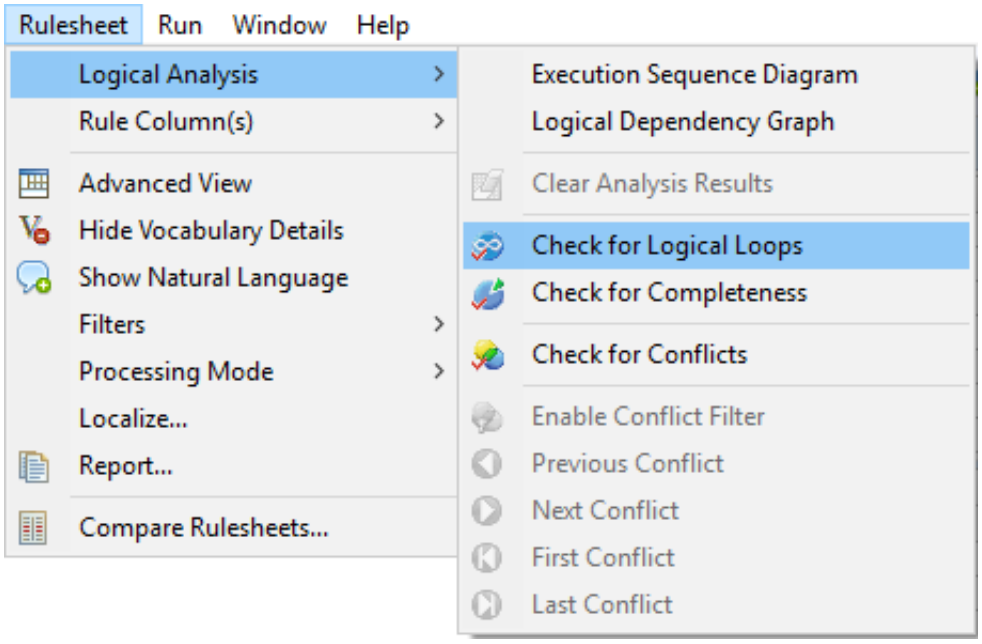
Click OK in the dialog box.

Note: Although checking for completeness can identify rules that you should include in your Decision Service, there may be situations where you don't want a newly-added rule. In this case, you can just delete the rule.

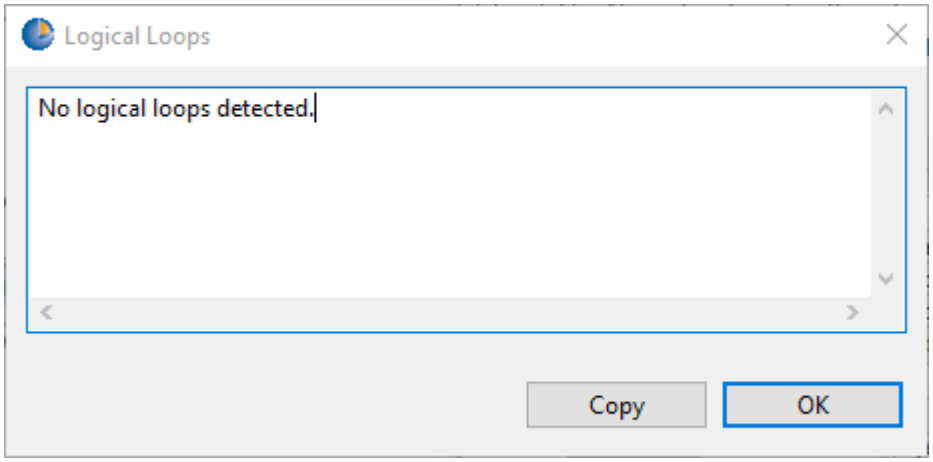
Save your Rulesheet.

Check for logical loops

A third form of logical error is circular logic or loops. To check for this, select Rulesheet > Logical Analysis > Check for Logical Loops.



You see the following result.



This is a very simple Rulesheet and contains no logical loops. Click OK in the dialog box to dismiss it. Note: While unintended logical loops should be fixed, sometimes logical loops are a useful technique for implementing rule logic that requires recursive reasoning.

11

Test rule execution

The Analyze phase helped to ensure the logical integrity of our rules. The Test phase helps to ensure that our rules give us the correct business results. Let's move on to testing.

First, we're going to define a test case for each one of our rules by defining some input values and expected results. Corticon Studio enables us to define our expected results, and then highlights any differences between the actual test results and the expected results. The table below defines one test case for each rule in the Rulesheet.

Cargo.ecore | *Cargo.ers

	Conditions	0	1	2	3
a	Cargo.weight		<= 20000	-	{> 20000, null}
b	Cargo.volume		-	> 30	{<= 30, null}
c					
d					
e					
f					
	Actions				
	Post Message(s)				
A	Cargo.container		standard	oversize	heavyweight
B					
C					
D					
E					
	Overrides			1	

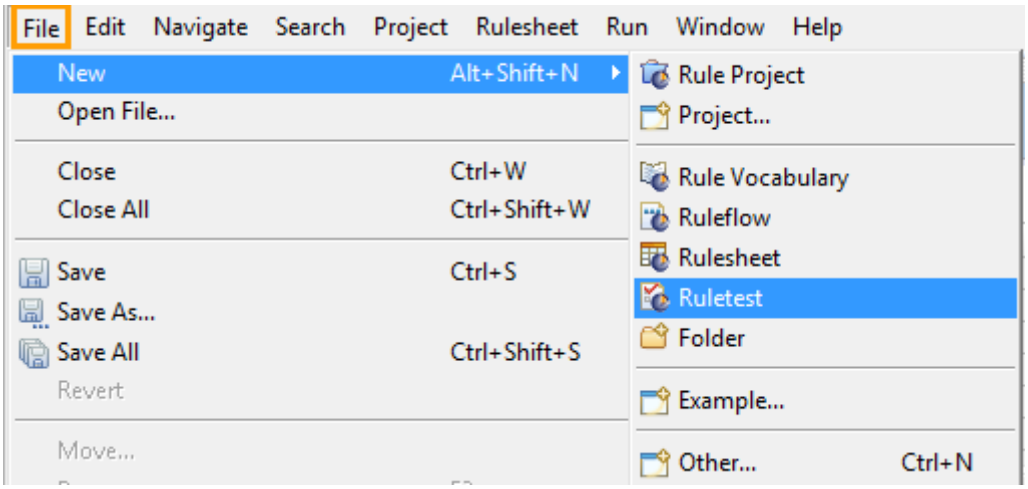
	Test Case 1	Test Case 2	Test Case 3
Input Values			
Cargo.weight	1000	1000	30000
Cargo.volume	10	40	20
Expected Results			
Cargo.container	standard	oversize	Heavyweight
Rule that should fire	Rule 1	Rule 2	Rule 3

For Test Case 2, we expect Rule 2 to override Rule 1. Even though the cargo weight is less than 20,000, which also satisfies the condition of Rule 1, the cargo volume is greater than 30, so Rule 2 overrides Rule 1 and is triggered.

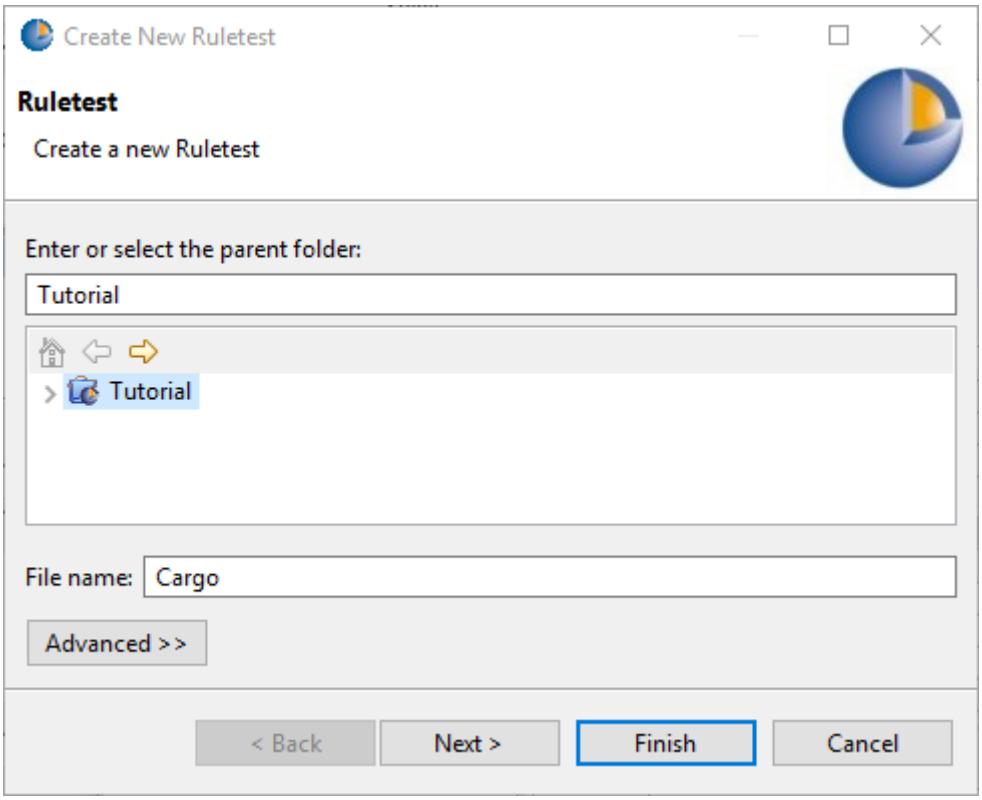
Create a Ruletest file

To begin testing, we need to create a Ruletest file:

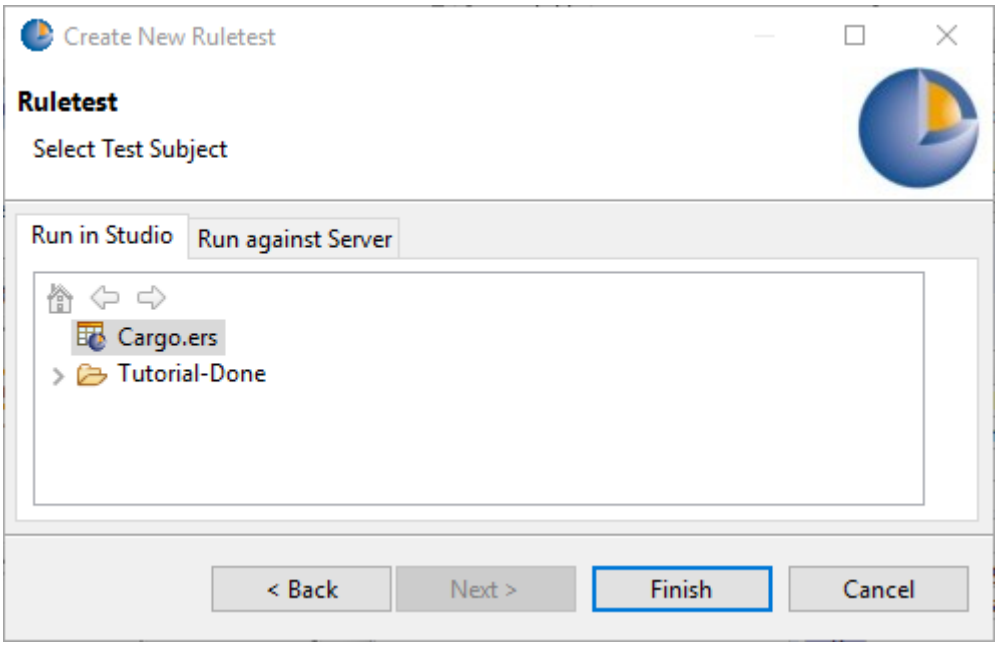
1. Select File > New > Ruletest.



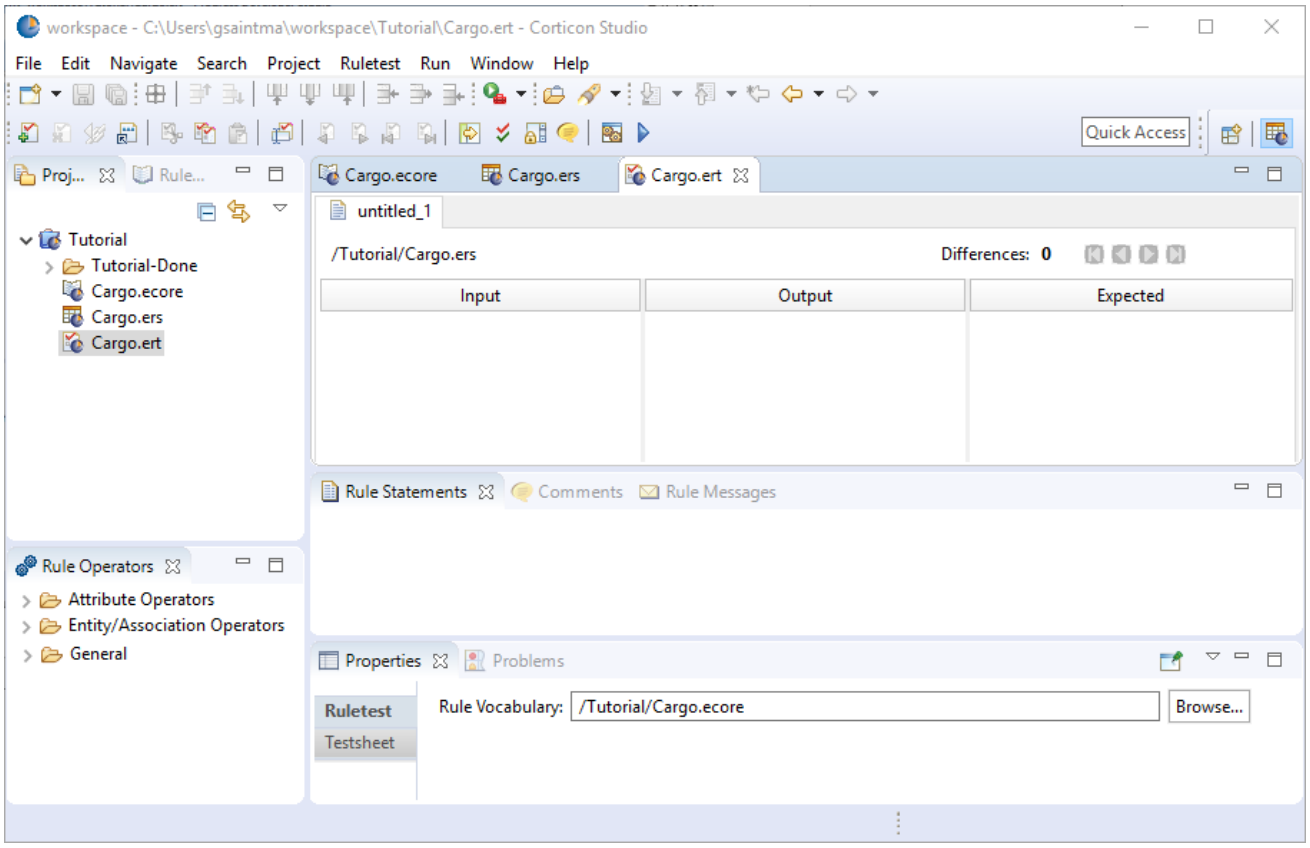
2. In the Create New Ruletest wizard, check that the parent folder is Tutorial, and then enter Cargo as the file name. Click Next.



3. Ensure that Cargo.ers is selected as the Test Subject, and then click Finish.



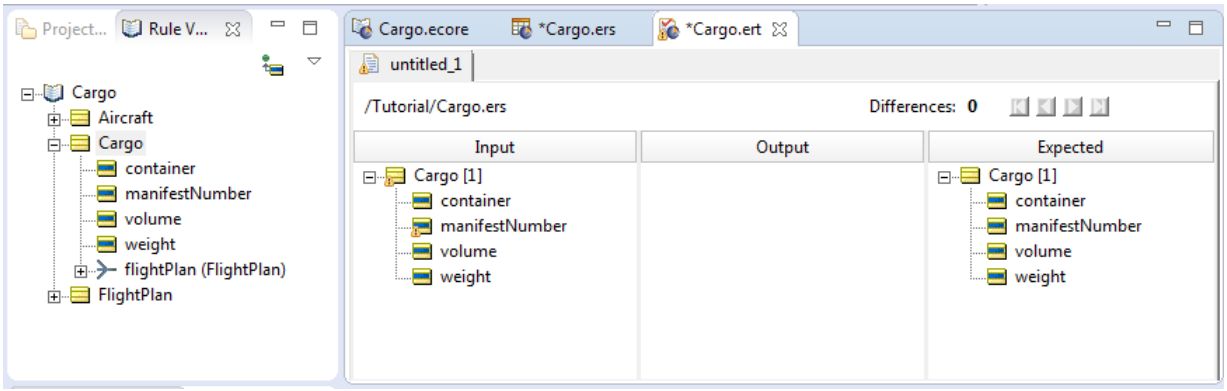
The Ruletest file opens in Corticon Studio on a new testsheet, untitled_1, as shown.



As you can see, your new Ruletest is associated with the appropriate Rulesheet. This ensures that your Ruletest scenario will test the right rules.

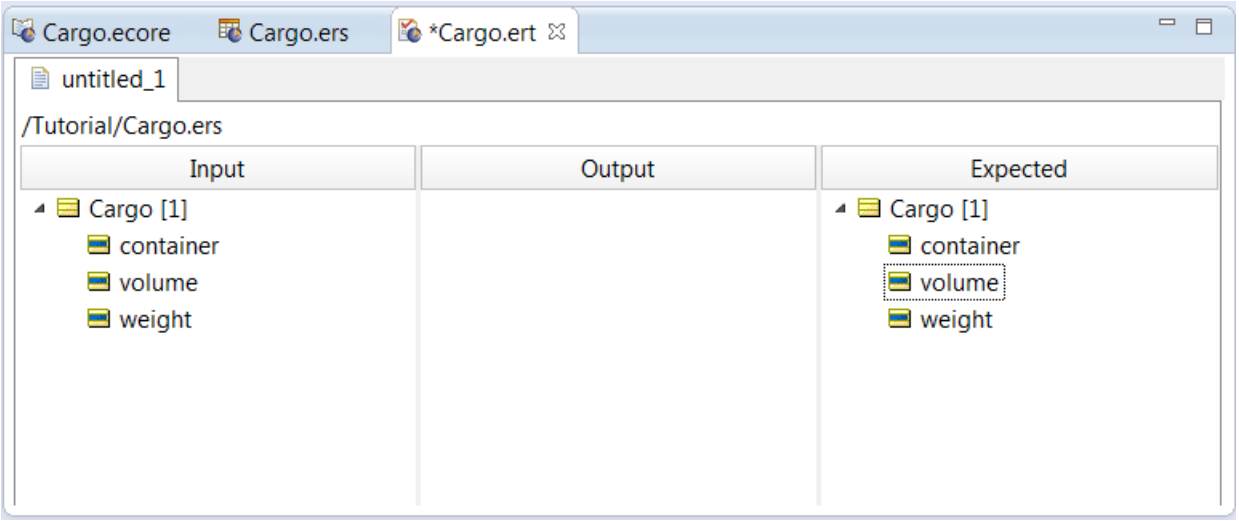
Set up test cases

1. Drag the Cargo entity from the Vocabulary and drop it anywhere in the Input as well as the Expected panes of the Ruletest.



Note: Cargo [1] on the Input and Expected panes represents a single 'instance' or 'example' of the Cargo entity. Changes you make to any Corticon Studio file will cause an asterisk character to appear in the file's tab. This is a reminder to save the file.

2. Remove unneeded attributes by selecting them and pressing the Delete key. In our case, we remove manifestNumber from both Input and Expected as it is not needed to test these rules.

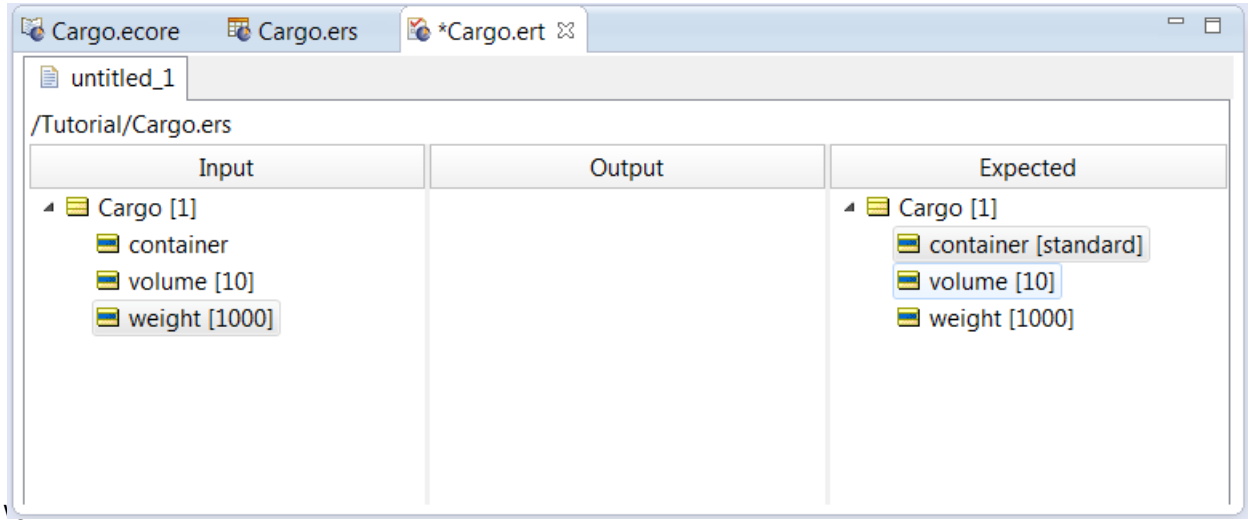


3. Add test data for Test Case 1 to the TestSheet, as in this table.

	Test Case 1	Test Case 2	Test Case 3
Input Values			
Cargo.weight	1000	1000	30000
Cargo.volume	10	40	20
Expected Results			
Cargo.container	standard	oversize	Heavyweight
Rule that should fire	Rule 1	Rule 2	Rule 3

To do this, double-click the terms weight and volume in the Input column to enter their values. Then do the

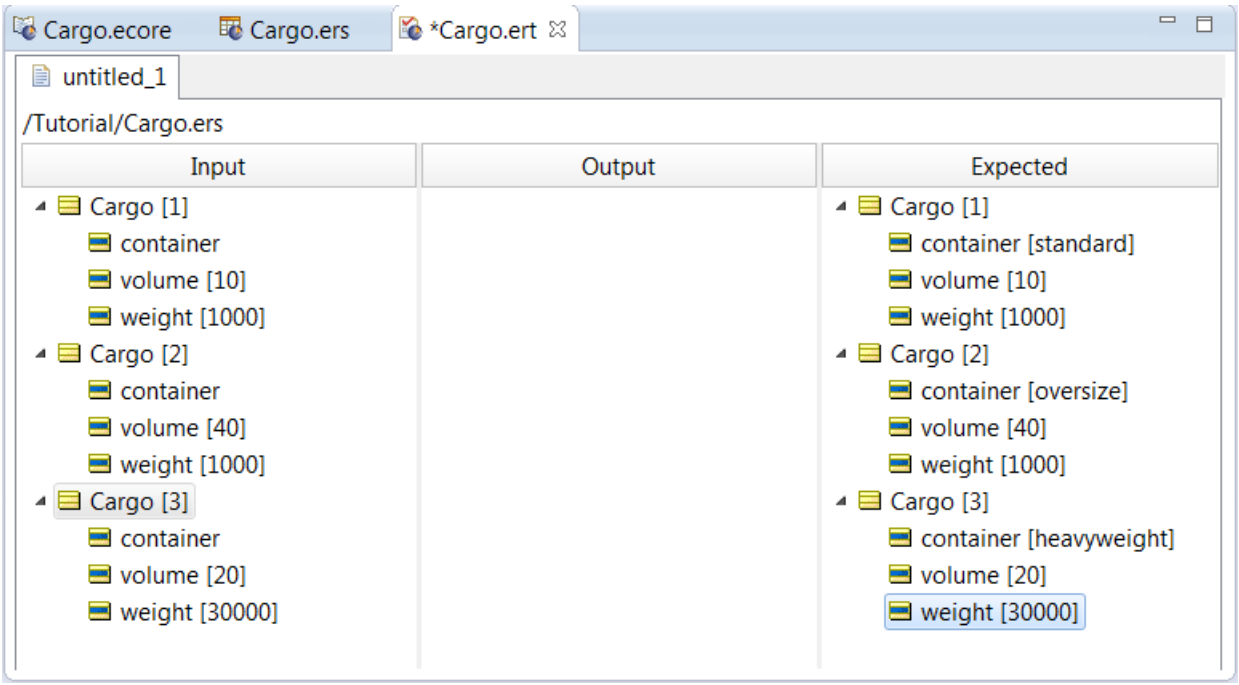
same in the Expected column, entering the container value as



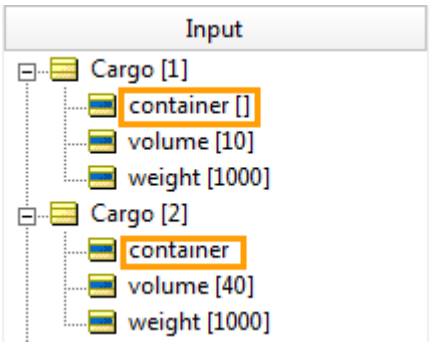
4. Enter the remaining test data based on the test cases in the table.

	Test Case 1	Test Case 2	Test Case 3
Input Values			
Cargo.weight	1000	1000	30000
Cargo.volume	10	40	20
Expected Results			
Cargo.container	standard	oversize	Heavyweight
Rule that should fire	Rule 1	Rule 2	Rule 3

You can duplicate the first test case by selecting Cargo [1], copying it, and then pasting it in the Input pane. Then, modify the input values based on the test cases. Repeat this for the Expected pane. Use the tab key to advance through the entry boxes. Specify values for container in the Expected pane --you can select these values from the container drop-down list. Your Ruletest should now look like this:



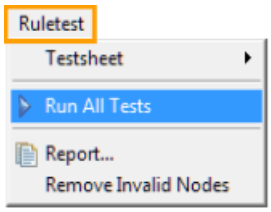
The input column expects that all its container values are null. That is, if you click on container and enter nothing you still set a value as illustrated for Cargo [1]:



To correct this, click on a container [] and then right click to choose Set to Null.

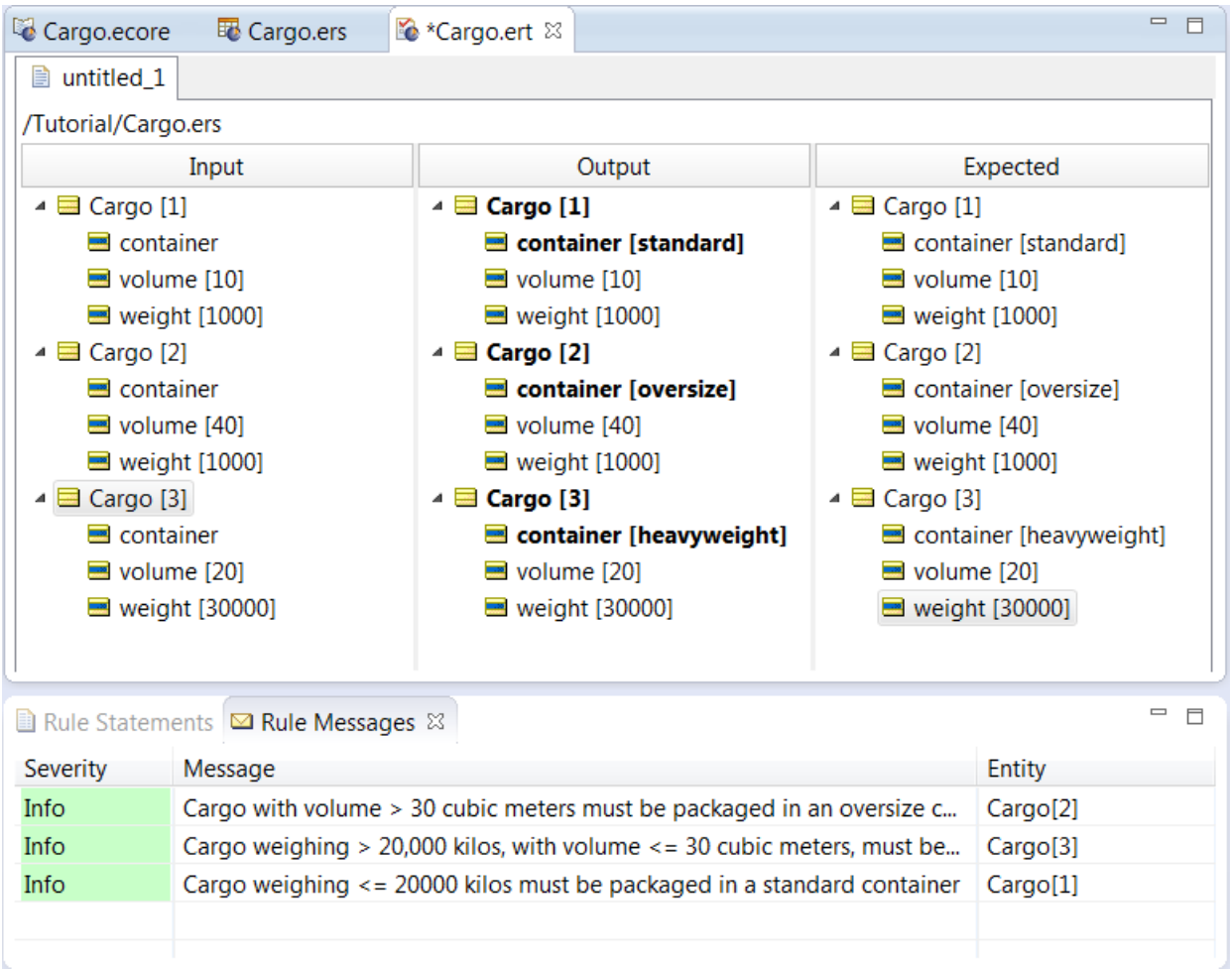
Run the Ruletest and verify the results

Run the Ruletest by selecting Ruletest > Run All Tests.



Running the Ruletest sends the data on the Input pane to the rules engine. The rules engine fires the appropriate rules and displays the results in the Output pane. The first time rules are executed, they are automatically compiled from the rule model into an optimized executable form, then deployed into the engine, which may take a few seconds. Once deployed, the rules will execute much faster on subsequent tests.

Next, check the outcome of your test in the TestSheet's Output pane.



Any attributes that are altered by the rules, including the entities to which they belong, are shown in bold text in the Output pane. Any unchanged attributes and values are displayed in normal style.

Messages shown in the Rule Messages pane are produced using the Post command in your Rule Statements. Severity indicates whether a message contains information, warnings, or violations. Message contains the text of the Rule Statement. Entity shows the entity to which this message is posted.

When we select the Cargo[1] entity within the Output pane, the first rule message is highlighted, showing the audit trail of rules that fired for that entity (in this case only one rule fired for Cargo[1]).

Use the Expected pane

Cargo.ecore | Cargo.ers | *Cargo.ert

untitled_1

/Tutorial/Cargo.ers

Input	Output	Expected
Cargo [1]	**Cargo [1]**	Cargo [1]
container	**container [standard]**	container [standard]
volume [10]	volume [10]	volume [10]
weight [1000]	weight [1000]	weight [1000]
Cargo [2]	**Cargo [2]**	Cargo [2]
container	**container [oversize]**	container [oversize]
volume [40]	volume [40]	volume [40]
weight [1000]	weight [1000]	weight [1000]
Cargo [3]	**Cargo [3]**	Cargo [3]
container	**container [heavyweight]**	container [heavyweight]
volume [20]	volume [20]	volume [20]
weight [30000]	weight [30000]	weight [30000]

Rule Statements | Rule Messages

Severity	Message	Entity
Info	Cargo with volume > 30 cubic meters must be packaged in an oversize c...	Cargo[2]
Info	Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be...	Cargo[3]
Info	Cargo weighing <= 20000 kilos must be packaged in a standard container	Cargo[1]

So far, the actual results in the Output pane match the expected results in the Expected pane. Let's look at what happens when the output does not match expected results.

Let's change one of the test cases. In the Expected pane, change the container value in Cargo[2] to standard.

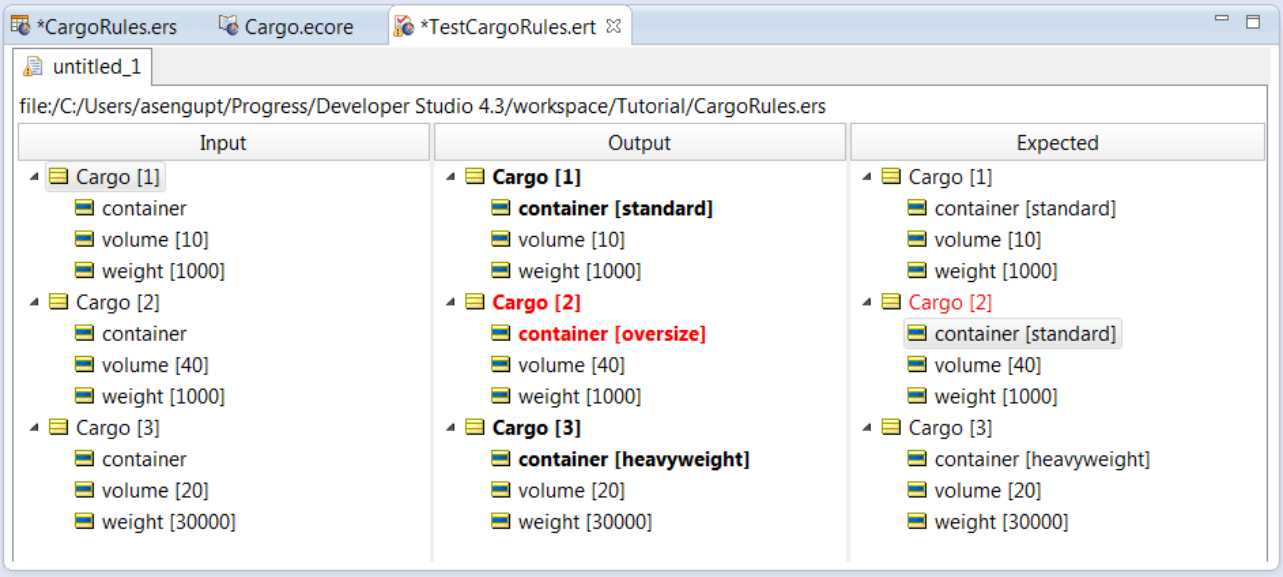
*CargoRules.ers | Cargo.ecore | *TestCargoRules.ert

untitled_1

file:/C:/Users/asengupt/Progress/Developer Studio 4.3/workspace/Tutorial/CargoRules.ers

Input	Output	Expected
Cargo [1]	**Cargo [1]**	Cargo [1]
container	**container [standard]**	container [standard]
volume [10]	volume [10]	volume [10]
weight [1000]	weight [1000]	weight [1000]
Cargo [2]	**Cargo [2]**	Cargo [2]
container	**container [oversize]**	container standard
volume [40]	volume [40]	volume [40]
weight [1000]	weight [1000]	weight [1000]
Cargo [3]	**Cargo [3]**	Cargo [3]
container	**container [heavyweight]**	container [heavyweight]
volume [20]	volume [20]	volume [20]
weight [30000]	weight [30000]	weight [30000]

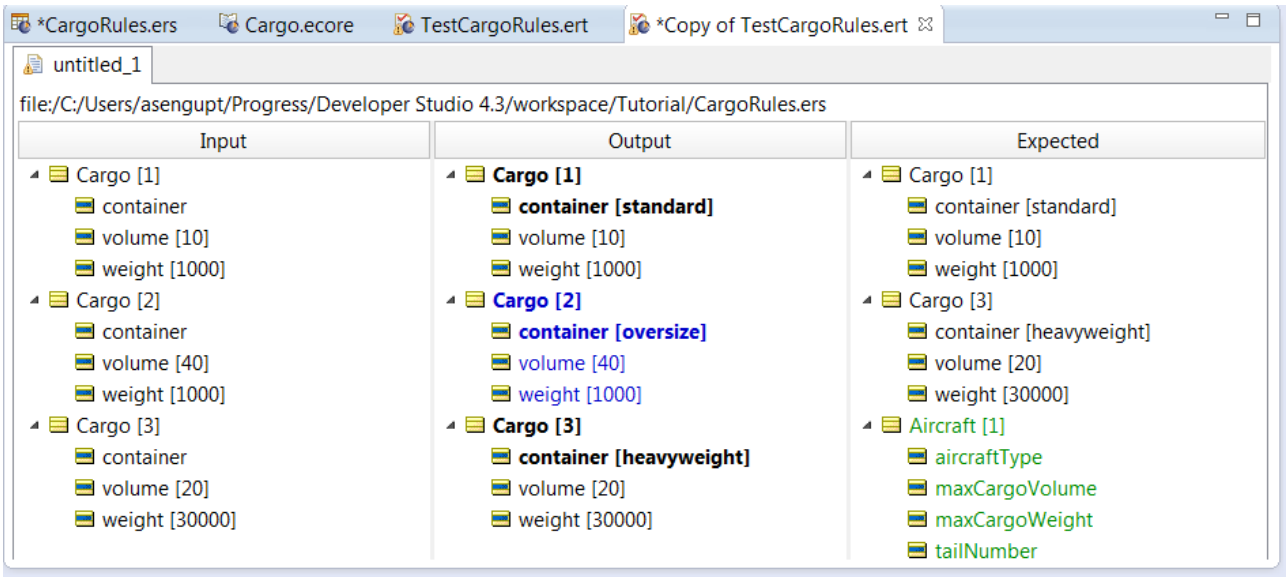
Run the test again. Your Ruletest should now look like this:



Here we see that Cargo[2] and the container attribute are colored red in both the Output and Expected panes, indicating that our Output results differ from our Expected results.

Be sure to change the container value of Cargo[2] in the Expected pane back to oversize and save the test for future use by selecting File > Save.

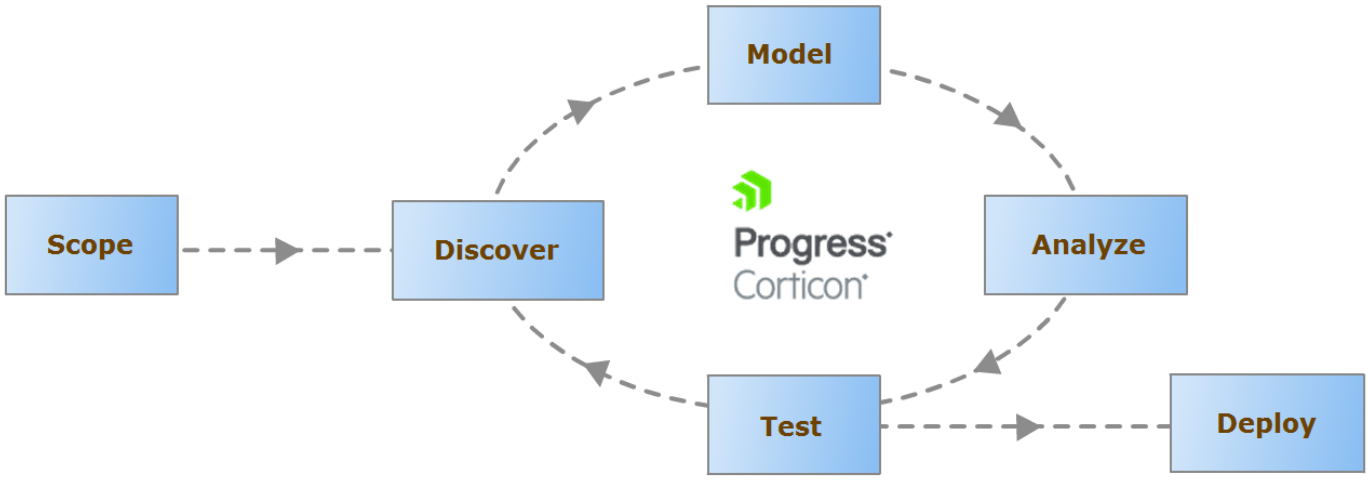
Note: Corticon Studio highlights other types of differences as well. Unexpected entities in the Output pane are highlighted in blue. Missing entities are highlighted in green in the Expected pane. Here's an example:

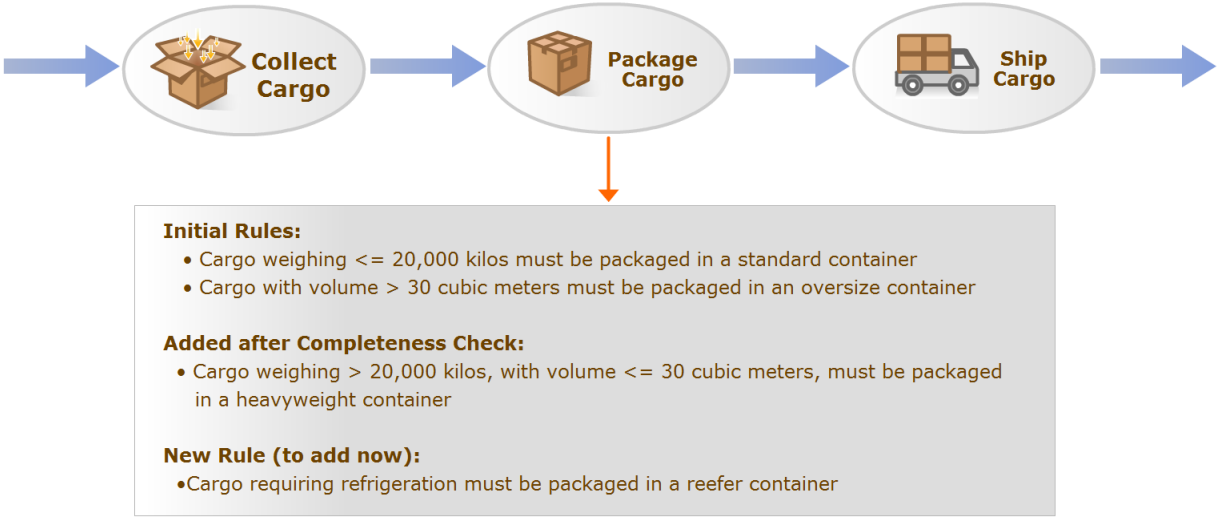


12

Iterate through the lifecycle again

Rule modeling is an iterative process. As your business needs change, you may need to make changes to your rules. Let's see how easy it is to do this in Corticon Studio by adding, analyzing, and testing a new rule in our Rulesheet.

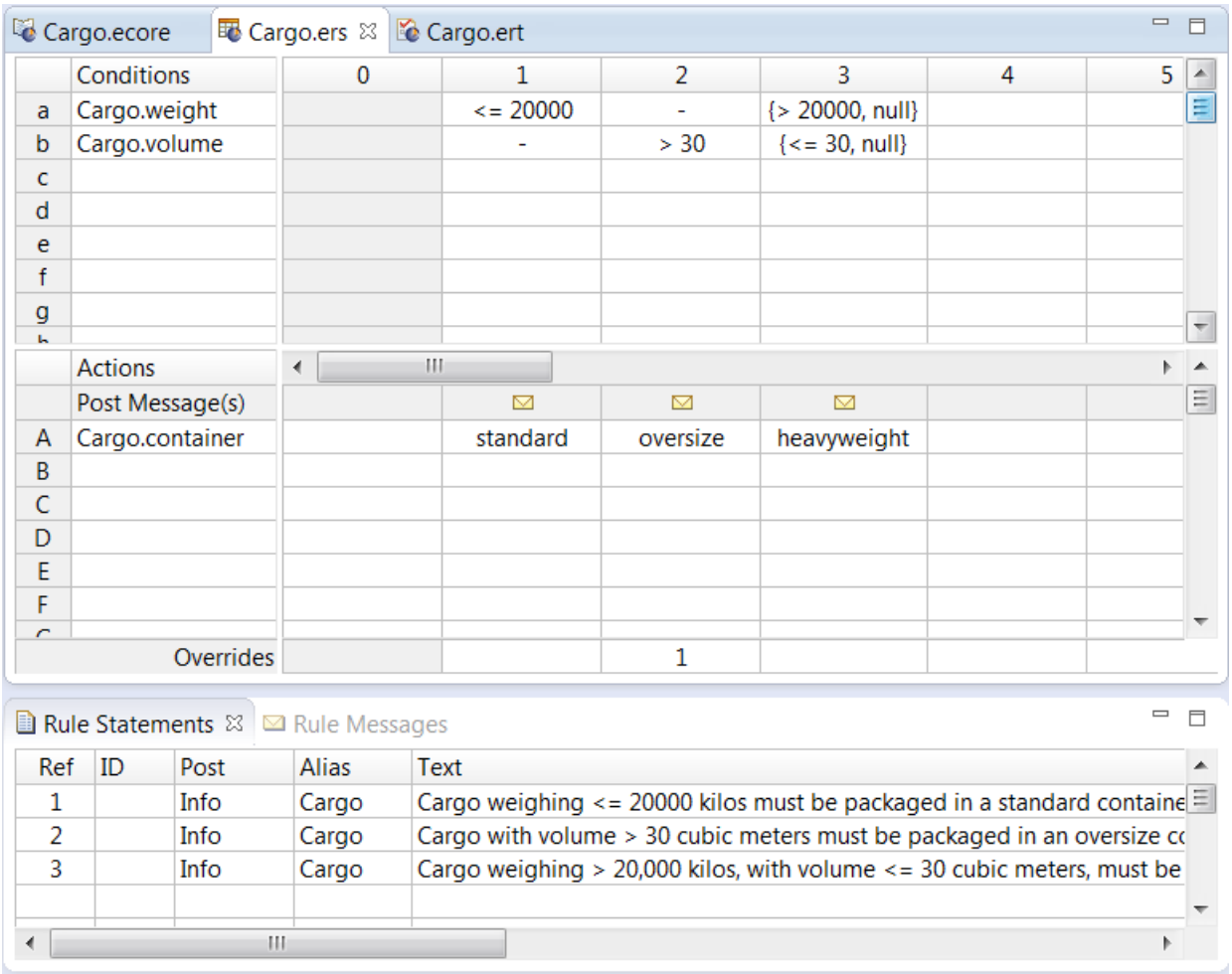




Let's begin by adding a rule statement for the new rule.

Add a new rule statement

Switch back to your Rulesheet by clicking the Cargo.ers tab. If you closed it earlier, reopen it by double-clicking the file inside the Rule Project Explorer pane.



Cargo.ecore | Cargo.ers | Cargo.ert

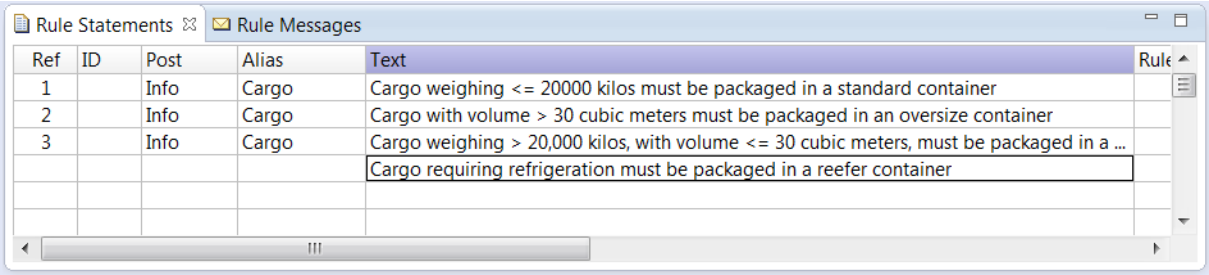
	Conditions	0	1	2	3	4	5
a	Cargo.weight		<= 20000	-	{> 20000, null}		
b	Cargo.volume		-	> 30	{<= 30, null}		
c							
d							
e							
f							
g							

	Actions						
	Post Message(s)						
A	Cargo.container		standard	oversize	heavyweight		
B							
C							
D							
E							
F							
	Overrides			1			

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
1		Info	Cargo	Cargo weighing <= 20000 kilos must be packaged in a standard containe
2		Info	Cargo	Cargo with volume > 30 cubic meters must be packaged in an oversize co
3		Info	Cargo	Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be

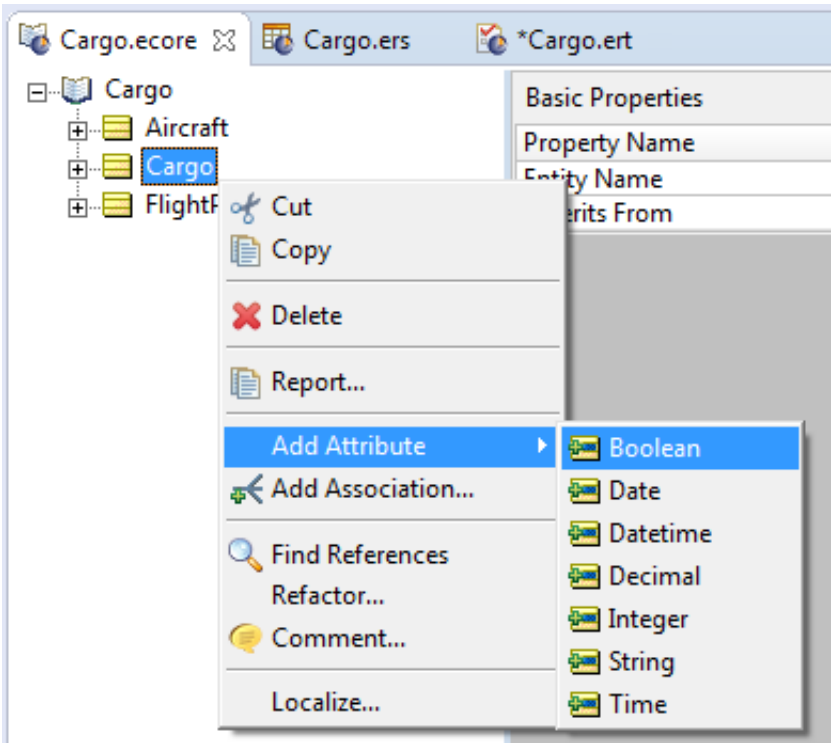
Enter the new Rule Statement—Cargo requiring refrigeration must be packaged in a reefer container. This will guide us when we model the new rule.



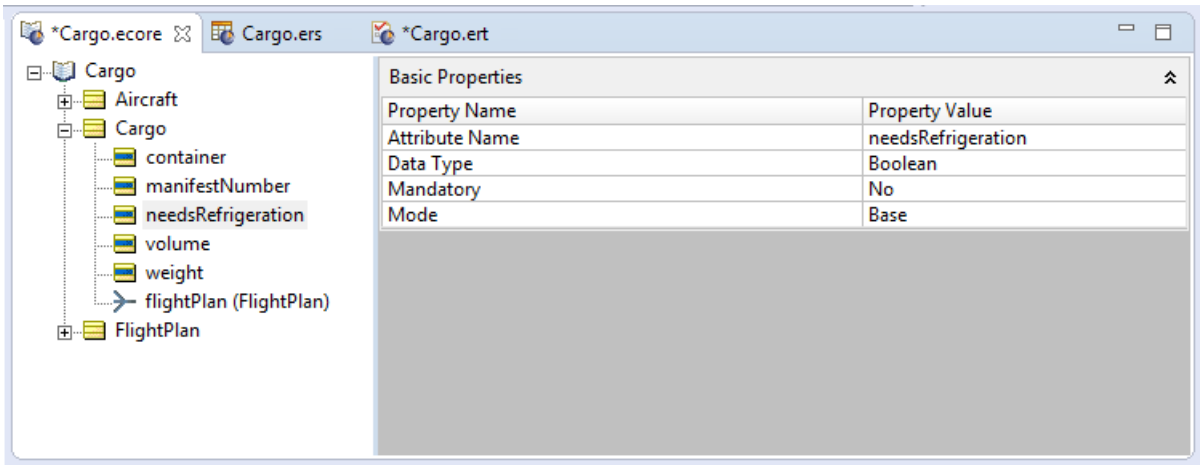
To model this rule, we need to edit the Vocabulary. First, we need to add a new attribute to the Cargo entity to define whether the Cargo requires refrigeration or not (let's call this needsRefrigeration). Second, we need add "reefer" as another possible selection option for the container attribute.

Open and edit the Vocabulary

1. Click the Vocabulary tab, Cargo.ecore tab. If you closed it, open it again by selecting Cargo.ecore from the Recent File list at the bottom of the File menu or double-click the file in the Rule Project Explorer pane. The Vocabulary opens in the Vocabulary editor.
2. Let's add a Boolean attribute named needsRefrigeration:
   a. In the Vocabulary editor in the upper right pane, right-click Cargo and choose Add Attribute and then Boolean from the submenu.

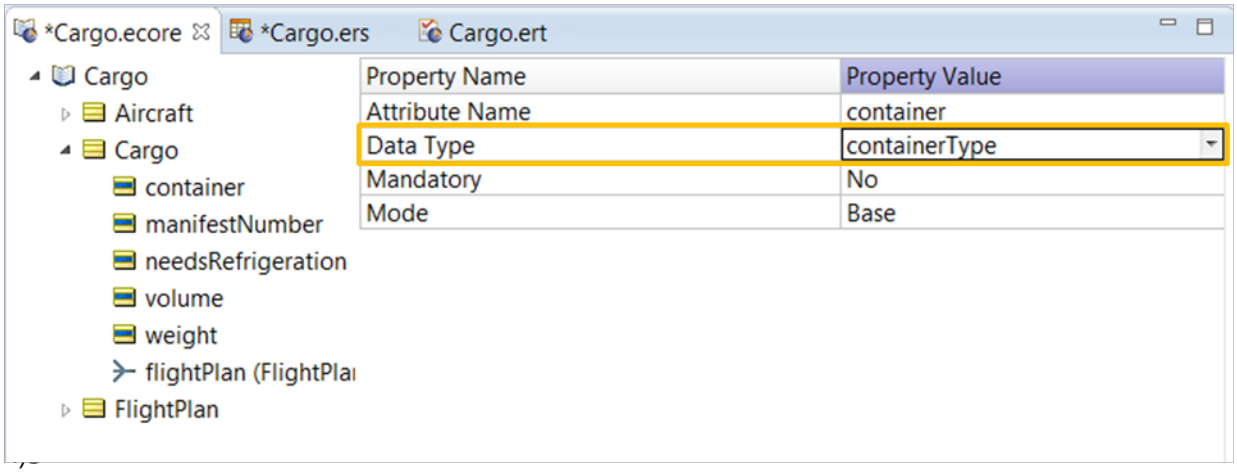


   b. Define the attribute name by double-clicking the default attribute name value (Attribute_1) and changing it to needsRefrigeration.

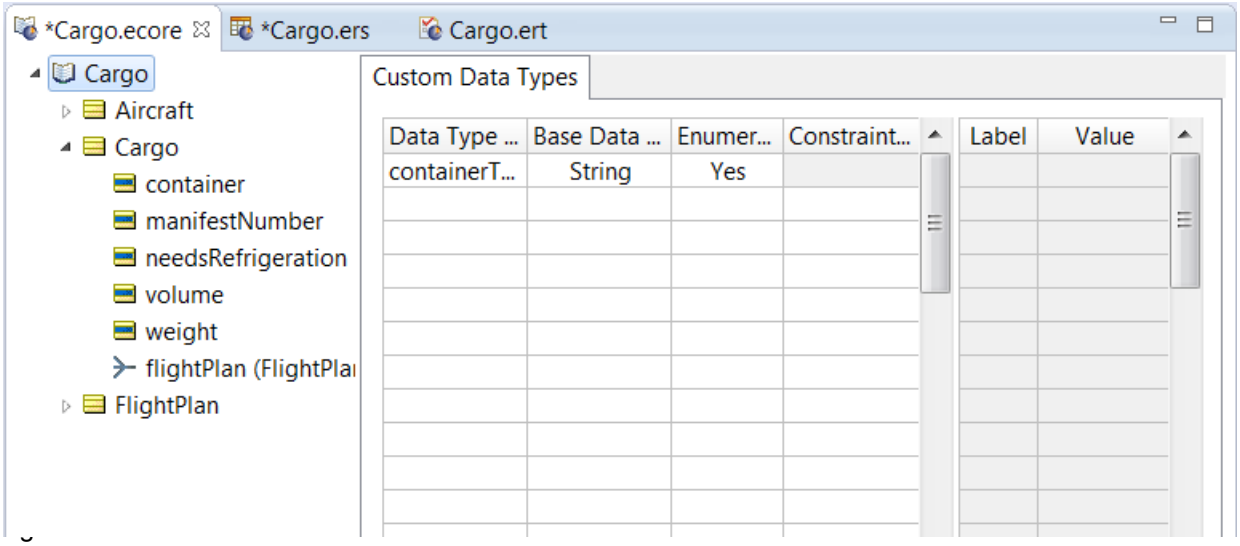


3. Add “reefer” as another allowable type of container:

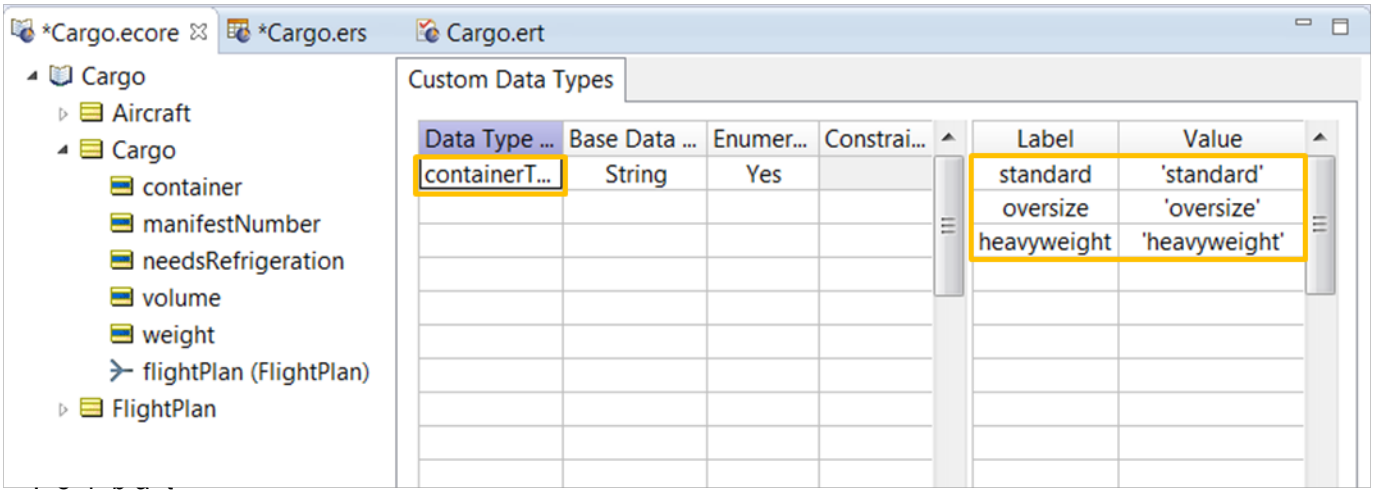
   a. Select the container attribute and note the Data Type: containerType. This is a custom data type that defines a set of allowable values for container (an enumerated



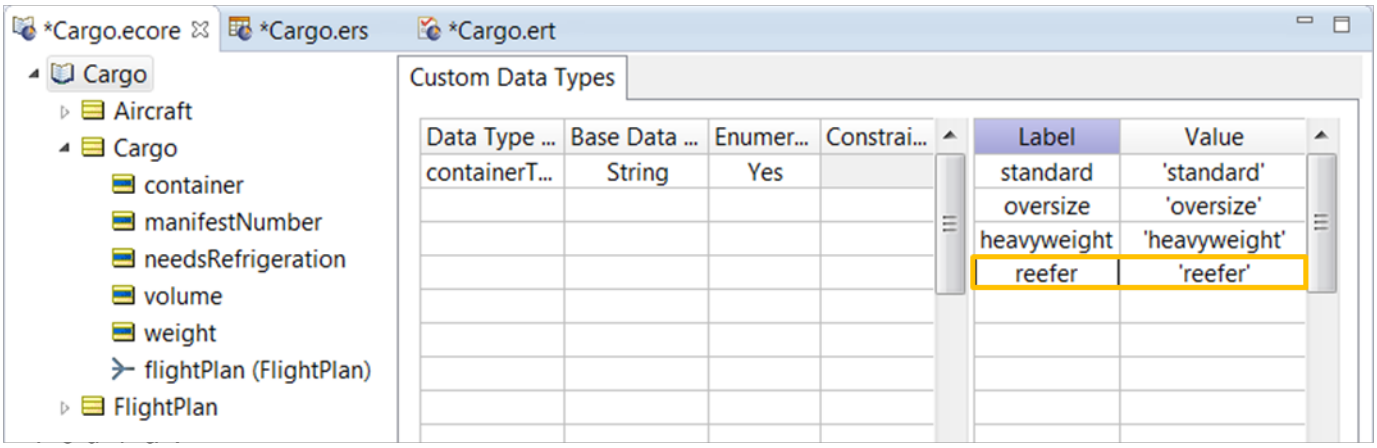
   b. To define custom data types, you must first click on the “root” node of the Vocabulary, denoted by the open book icon—in this case,



c. Click the containerType entry. This will display the enumerated set in the rightmost



d. Add “reefer” as both a Label and a



4. Save the Vocabulary by selecting File > Save and close the Vocabulary Editor pane.

Custom data types are a powerful capability that helps you define a Vocabulary that matches how you think about your business.

Model the new rule

Now that the Vocabulary contains the new terms required by the new rule, let’s model the new rule—Cargo requiring refrigeration must be packaged in a reefer container. When you are done, your Rulesheet should look like this:

Cargo.ecore | *Cargo.ers | *Cargo.ert

	Conditions	0	1	2	3	4	5
a	Cargo.weight		<= 20000	-	{> 20000, null}	-	
b	Cargo.volume		-	> 30	{<= 30, null}	-	
c	Cargo.needsRefrigeration		-	-	-	T	
d							
e							
f							
g							

	Actions						
	Post Message(s)						
A	Cargo.container		standard	oversize	heavyweight	reefer	
B							
C							
D							
E							
F							
	Overrides			1			

Be sure to define the Reference link from the Rule Statement to column 4 and add Post and Alias as shown here.

Cargo.ecore | Cargo.ers | *Cargo.ert

	Conditions	0	1	2	3	4	5
a	Cargo.weight		<= 20000	-	{> 20000, null}	-	
b	Cargo.volume		-	> 30	{<= 30, null}	-	
c	Cargo.needsRefrigeration		-	-	-	T	
d							
e							
f							
g							

	Actions						
	Post Message(s)						
A	Cargo.container		standard	oversize	heavyweight	reefer	
B							
C							
D							
E							
F							
	Overrides			1			

Rule Statements | Rule Messages

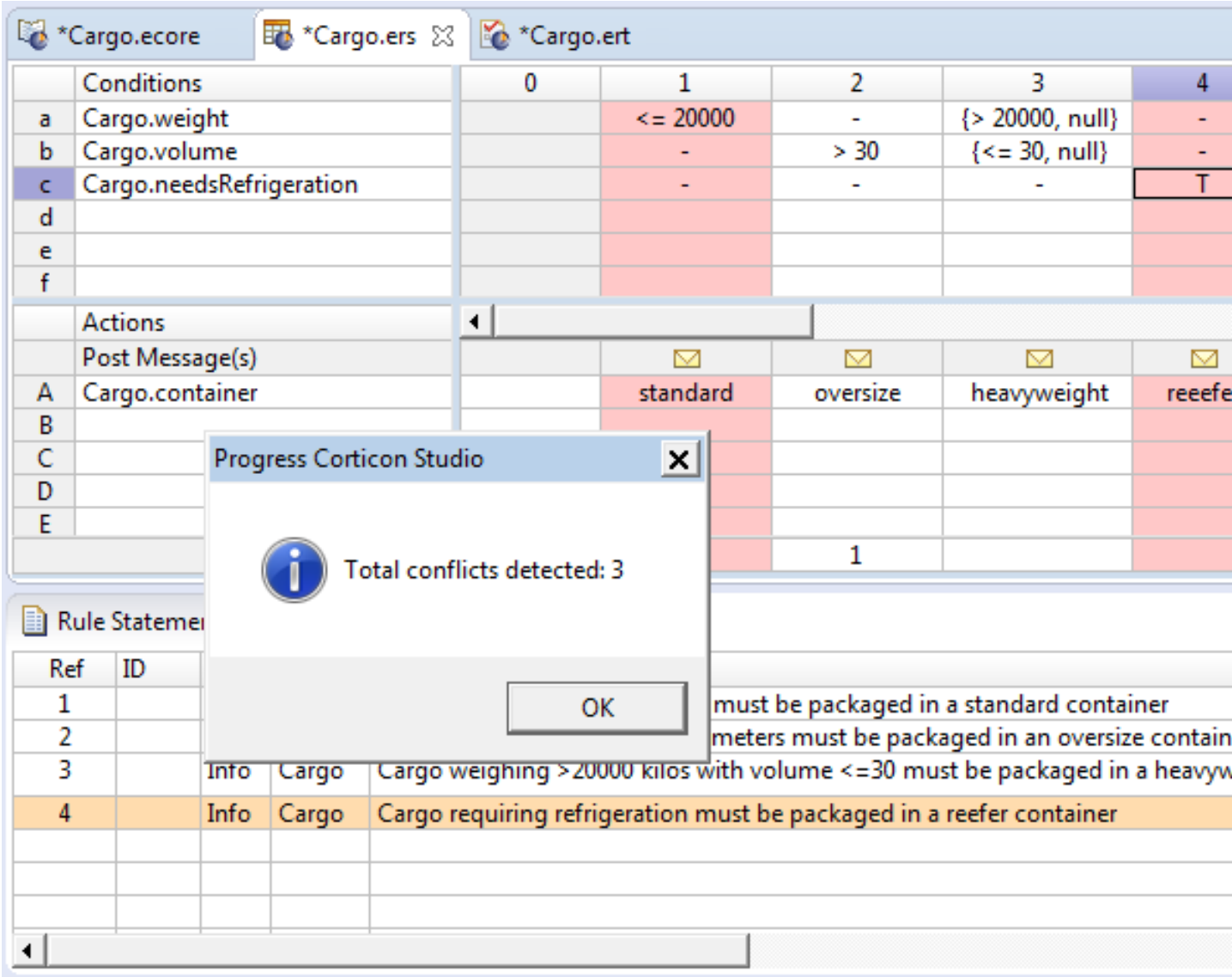
Ref	ID	Post	Alias	Text
1		Info	Cargo	Cargo weighing <= 20000 kilos must be packaged in a standard container
2		Info	Cargo	Cargo with volume > 30 cubic meters must be packaged in an oversize container
3		Info	Cargo	Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be packaged in
4		Info	Cargo	Cargo requiring refrigeration must be packaged in a reefer container

Analyze the new rule

Whenever we add or change our rules, we should re-analyze.

Let's start by checking for conflicts. You can either select Rulesheet > Logical Analysis > Check for Conflicts or click the Check for Conflicts button

that appears right below the menu bar. As you can see, adding the new rule has caused three new conflicts, one with each of our three existing rules.



You can step through multiple conflicts, and filter the Rulesheet to view only the conflicting rules, using the buttons to the right of the Conflict Check button.

Note that Rules 1, 2, and 3 each define a specific container type for cargo based on weight and volume, while Rule 4 defines the container type based on whether the cargo needs refrigeration. The conflict occurs because a cargo load that requires refrigeration may also trigger Rules 1, 2, or 3.

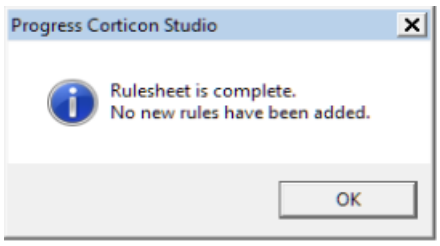
Note: Let us assume that reefer containers are only available for standard and heavyweight loads, but not for oversize loads. So, let's set Rule 4 to override rules 1 and 3, and set Rule 2 to override Rules 1 and 4. Remember, you can select multiple values in the override cell by holding down the Ctrl key.

After you have made these changes, your Rulesheet looks like this:

Cargo.ecore | Cargo.ers | *Cargo.ert

	Conditions	0	1	2	3	4
a	Cargo.weight		<= 20000	-	{> 20000, null}	-
b	Cargo.volume		-	> 30	{<= 30, null}	-
c	Cargo.needsRefrigeration		-	-	-	T
d						
e						
f						
g						
	Actions					
	Post Message(s)					
A	Cargo.container		standard	oversize	heavyweight	reefer
B						
C						
D						
E						
F						
	Overrides			{1, 4}		{1, 3}

Next, let's check for completeness by selecting Rulesheet > Logical Analysis > Check for Completeness.



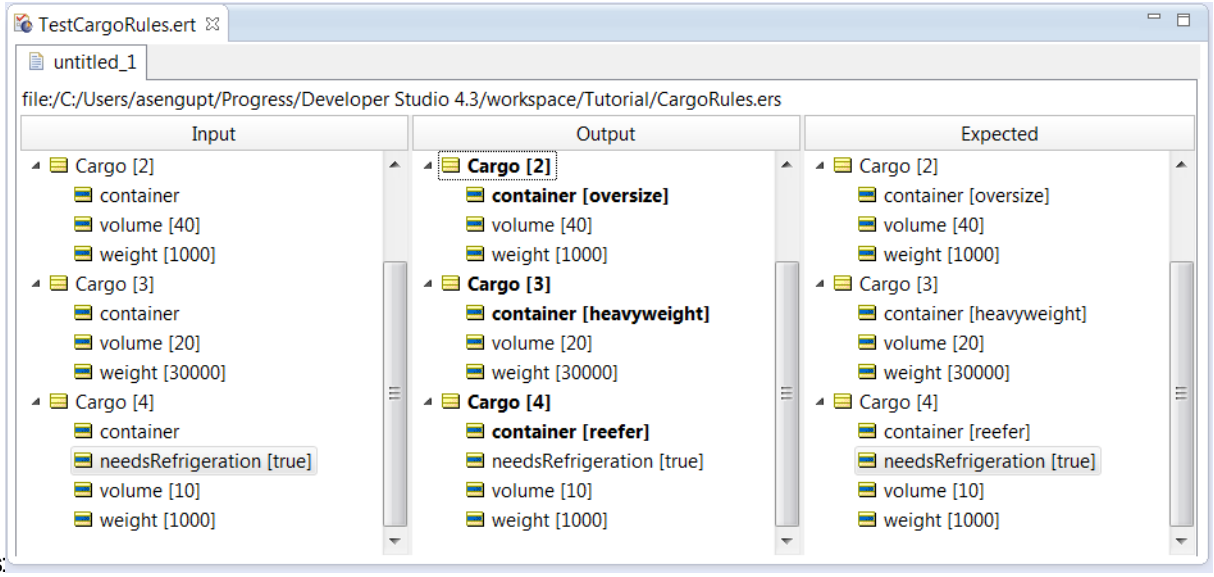
As you can see, the Rulesheet is complete. Click OK in the dialog box and save the Rulesheet file.

Run the Ruletest again

The last step is to modify your test case to test the new rule.

1. Open Cargo.ert.
2. Copy and paste Cargo[1] to create Cargo[4], in both the Input and Expected panes.
3. Drag and drop the needsRefrigeration attribute from your Vocabulary onto both Cargo[4] entities.
4. In both panes, set the value of needsRefrigeration to "true".

5. In the Expected pane, change the value of container in Cargo[4] to “reefer”.
6. Run the test. Your Ruletest should now look like this:



As you can see, the actual results match the expected results.

Congratulations! You have now completed two full iterations of the Corticon decision service development lifecycle and understand the basic concepts of rule modeling, analysis, and testing in Corticon Studio.

To learn advanced rule modeling techniques, take the Advanced Rule Modeling Tutorial.