



Corticon Basic Guided Journey

Table of Contents

Introduction to BRMS.....	7
What is a BRMS?.....	8
Introduction to Progress Corticon.....	8
Roles in the lifecycle of a Decision Service.....	9
Overview of Corticon Studio.....	9
Tasks required to create and test rule models.....	12
What is a Vocabulary?.....	13
How business terms are represented in the Corticon Vocabulary.....	14
The Vocabulary tree.....	14
Tasks to define a Vocabulary.....	15
Create a Vocabulary file.....	15
Add an entity.....	16
Add attributes to an entity.....	16
Types of associations.....	18
Add associations to an entity.....	18
Rule Operators.....	19
What is a Rulesheet?.....	21
Tasks to define rules in a Rulesheet	22
Create a Rulesheet file.....	23
Define a rule statement.....	23
Define rules in the Rulesheet editor.....	24
Syntax for values in rule column cells.....	24
Syntax errors in a Rulesheet.....	25
Link a rule with a rule statement.....	26
How to analyze rules in a Rulesheet.....	26
What is a Ruletest?.....	29
How a Ruletest works.....	30
Tasks to test rules in a Rulesheet using a Ruletest.....	31
Create a Ruletest file.....	31
How to specify input for a Ruletest.....	32
How to specify expected results.....	32
How to run a Ruletest.....	33
Compare the output with expected results.....	34
How to embed dynamic data in a rule statement.....	35

How to link a rule statement to multiple rules.....	35
Enhance rules.....	37
Action-only rules.....	37
Comparison operators.....	39
Equations and calculations.....	40
Value sets and ranges.....	41
Boolean conditions.....	43
Logical structures.....	44
How to check for null values.....	48
What is a Ruleflow?.....	51
How to create Ruleflows.....	52
Create a Ruleflow file.....	52
Overview of the Ruleflow editor.....	53
Add Rulesheets to the Ruleflow.....	53
Connect Rulesheets in the Ruleflow editor.....	54
Set Ruleflow properties.....	54
Test Ruleflows using Ruletests.....	55
Enable and disable Rulesheets in a Ruleflow.....	55
Advanced Ruleflow tips and tricks.....	56

Copyright

© 2021 Progress Software Corporation and/or 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.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

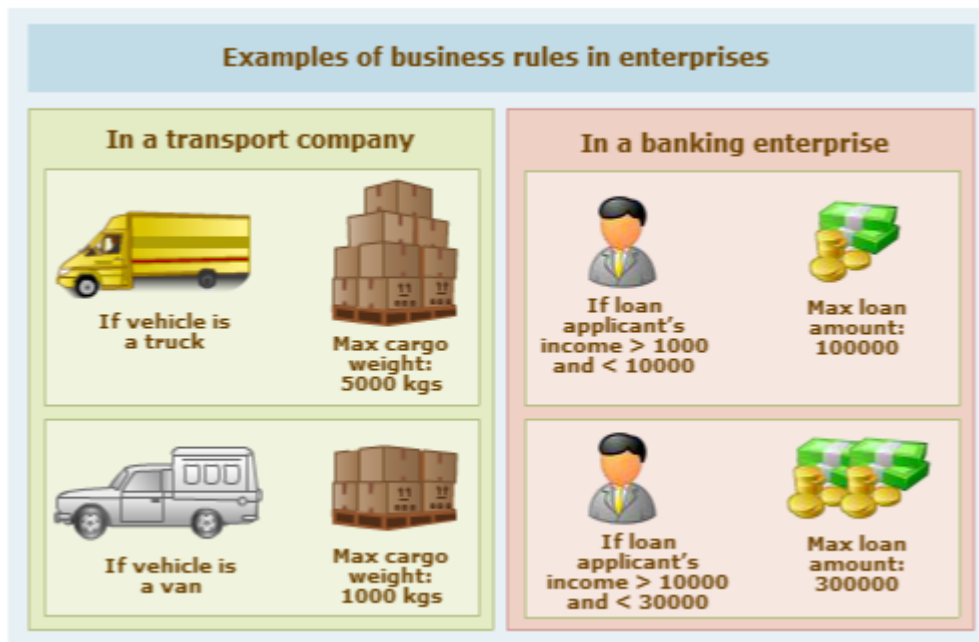
Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, 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.

October 2020

Last updated with new content: Release 6.2

Introduction to BRMS

Every enterprise has rules, policies, and regulations that govern different aspects of its business operations.



For example, a transport carrier company may have rules about how much cargo each of its vehicles can carry, or a bank may have rules about how much money it can lend to an individual.

In the past, enterprises implemented business rules manually, by employees. This approach tended to be inefficient, prone to errors, and lacked transparency. To eliminate these problems enterprises began to automate business rules, enabling employees and systems to make business decisions more quickly and with fewer errors.

Before the advent of BRMS, automating business rules meant developing your own rules application. This was time consuming, resource-intensive, and costly. Often the application would fall short of realizing the business analyst's requirements. Additionally, any changes required modifying and testing application code.

A better approach is to use a BRMS such as Corticon that can enable enterprises to automate business rules quickly, transparently and flexibly.

For details, see the following topics:

- [What is a BRMS?](#)
- [Introduction to Progress Corticon](#)
- [Tasks required to create and test rule models](#)

What is a BRMS?

A Business Rule Management System (BRMS) like Corticon enables enterprises to model, test, and automate complex business rules without writing code.

A BRMS has several advantages:

- Increased agility—enterprises using a BRMS show greater responsiveness to change and reduced time-to-market in implementing new rules or modifying existing rules.
- Reduced cost—automated decision-making is faster and more efficient.
- Better quality—logical rule models reduce human errors, ensure that there is consistency in decision-making, and encourage the use of best practices.
- Increased visibility into the application development process—requirements that are captured as rule models by business analysts become the implementation, which reduces surprises during user-acceptance testing.

Business rules that are automated using a BRMS can be accessed as services by internal or external systems and can be invoked as part of an automated business process.

Introduction to Progress Corticon

Progress Corticon is a BRMS. It enables enterprises to model, test, and automate business rules. Progress Corticon offers the following components:

- Progress Corticon.js Studio - the development environment to create and test rule models for JavaScript. Corticon.js Studio offers the ability to package rules as JavaScript to be run wherever JavaScript runs.
- Progress Corticon Studio—the development environment to create and test rule models for Java.
- Progress Corticon Server—the runtime environment that hosts deployed rule models (created in Corticon Studio), exposing them as Decision Services to the external world. Decision Services can be exposed as Web services (SOAP or REST), Java services, or .NET services.

Roles in the lifecycle of a Decision Service

There are two roles in the lifecycle of a Decision Service—rule modelers and integration developers.

Rule modelers create rule models in Corticon Studio or Corticon.js Studio depending on their deployment strategy. They define business terms, the rule logic, and the order of execution of rules.

Integration developers package and deploy rules as Decision Services to Corticon Server or for JavaScript deployment and make them accessible to external systems.

Generally, only rule modelers use Corticon Studio. Integration developers might use certain features of Corticon Studio to connect to databases and publish rule models, but they largely work with Corticon Server and the tools provided with it or specific JavaScript deployments.

This content provides an entry point into Corticon for rule modelers as well as integration developers.

Overview of Corticon Studio

Corticon Studio and Corticon.js are integrated development environments for developing business rules. They have the following features:

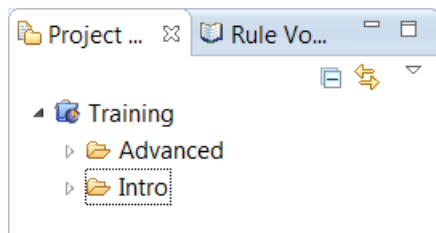
- Workspaces, projects, and folders to organize rule-modeling components
- Capabilities to create rule-modeling components
- Views and perspectives to see and open rule-modeling components
- Editors to develop and modify rule-modeling components

Start Corticon Studio by choosing Progress > Corticon 6.x Studio.

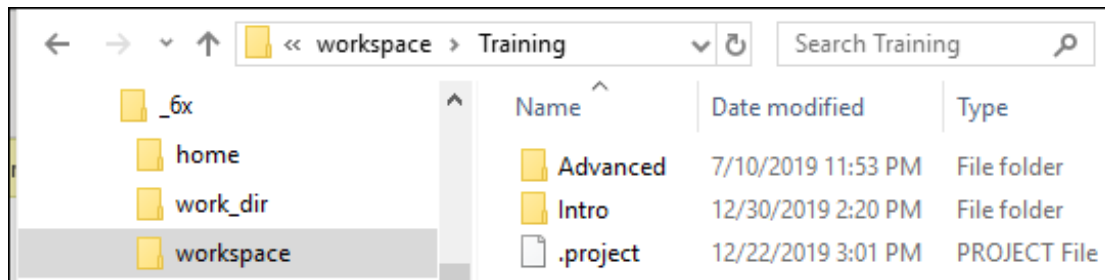
Start Corticon.js Studio by choosing Progress > Corticon.js 1.x Studio.

Workspaces, projects, and folders

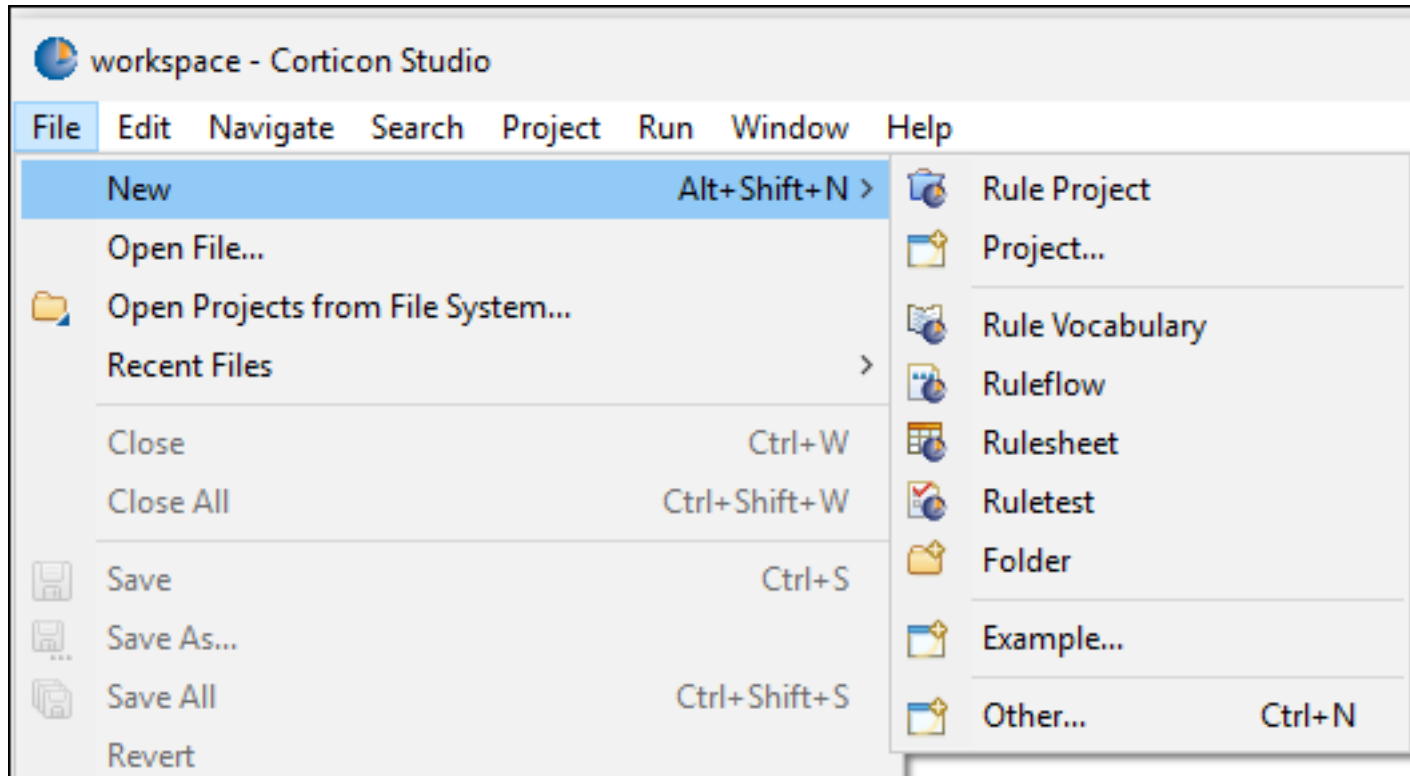
Workspaces, projects, and folders provide a way to organize rule-modeling components. A workspace is a 'root' directory that acts as a container for rule projects created in Corticon Studio.



The Training project is a Corticon rule project and it is a container for rule-modeling components. A Corticon rule project can further contain folders to separate rule-modeling components. When you create workspaces, rule projects, or folders, the relevant directory and folder structures are created as locations on your computer, as shown in bottom image.



Rule-modeling components

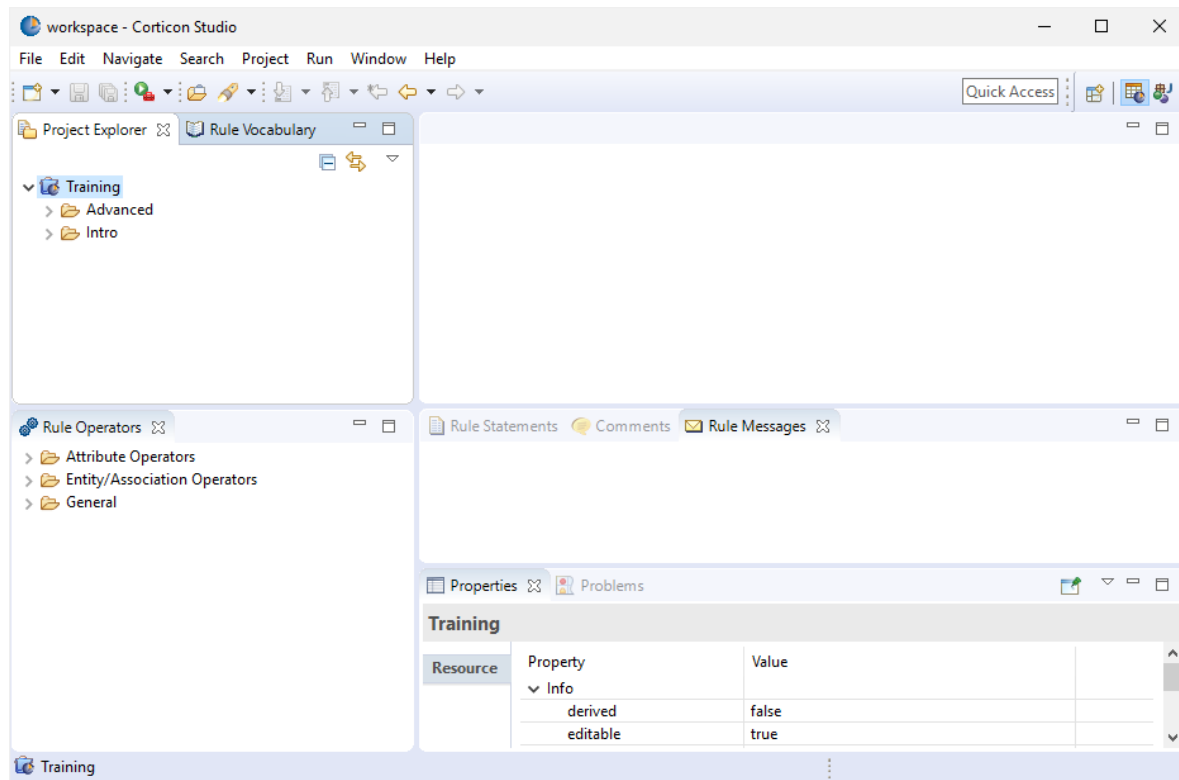


Corticon Studio provides capabilities to create the following rule-modeling components:

- Rule Vocabulary—enables you to define business terms used by the rule model.
- Rulesheet—enables you to model rule logic using terms from the Vocabulary.
- Ruleflow—enables you to organize one or more Rulesheets into a sequence that can be deployed as a Decision Service on Corticon Server.
- Ruletest—enables you to test Rulesheets and Ruleflows.

Together, the Vocabulary, Rulesheets, and Ruleflows form a rule model in Corticon.

Views and perspectives



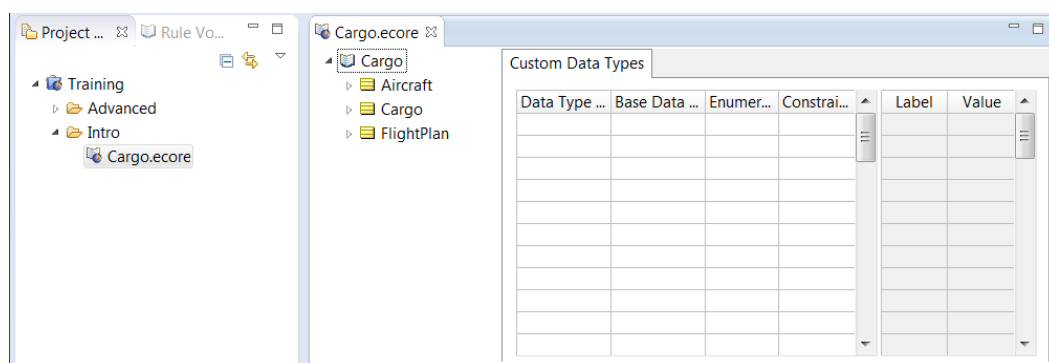
The user interface of Corticon Studio is composed of several panels, each with one or more Views as tabs. Each of which displays rule components, details of rule components, or other features used in rule modeling. They provide an easy way to access and use rule components.

For example, the Project Explorer view at the top left of this image enables you to access rule components contained in rule projects. To open a rule component you simply double-click it from the Project Explorer view.

A perspective is an arrangement or layout of panels and views. By default, Corticon Studio opens in the Corticon Designer perspective, which provides all the views you require to model your rules.

You can customize a perspective by clicking and dragging views within Corticon Studio. You can close views by clicking the Close icon on the view.

Editors



When you open a rule-modeling component (by creating it, or by double-clicking it in the Project Explorer view), it opens in an editor. Each rule-modeling component opens in its own, unique, editor, which enables you to define and develop the rule component.

In this example, a Vocabulary file (Cargo.ecore) is opened in a Vocabulary editor.

Tasks required to create and test rule models

To model and test rules in Corticon Studio, you must do the following:

- Import or create a Vocabulary that contains business terms.
- Model your rule logic in Rulesheets.
- Test your rule logic using Ruletests.
- Create a Ruleflow that can be deployed as a Decision Service.

What is a Vocabulary?

A Corticon Vocabulary is a rule-modeling component that enables you to define all the business terms that you require in your rules.



For example, a transport company may have a rule that determines how much cargo each type of vehicle can carry. There are two key business terms used in this rule—cargo and vehicle. You could define these terms as entities in your Vocabulary.

A Vocabulary is similar to a data model such as a Unified Modeling Language (UML) model or an Entity-Relationship model. The terms for the Vocabulary could come from a number of sources—database tables, forms used in business operations, policy and procedure documents, etc.

When you build a Vocabulary, you not only define the terms, you also define relationships between those terms. For example, a single vehicle can carry many cargo containers, implying a one-to-many relationship. You would define this as an association in your Vocabulary.

For details, see the following topics:

- [How business terms are represented in the Corticon Vocabulary](#)
- [Create a Vocabulary file](#)
- [Add an entity](#)
- [Add attributes to an entity](#)
- [Types of associations](#)
- [Add associations to an entity](#)
- [Rule Operators](#)

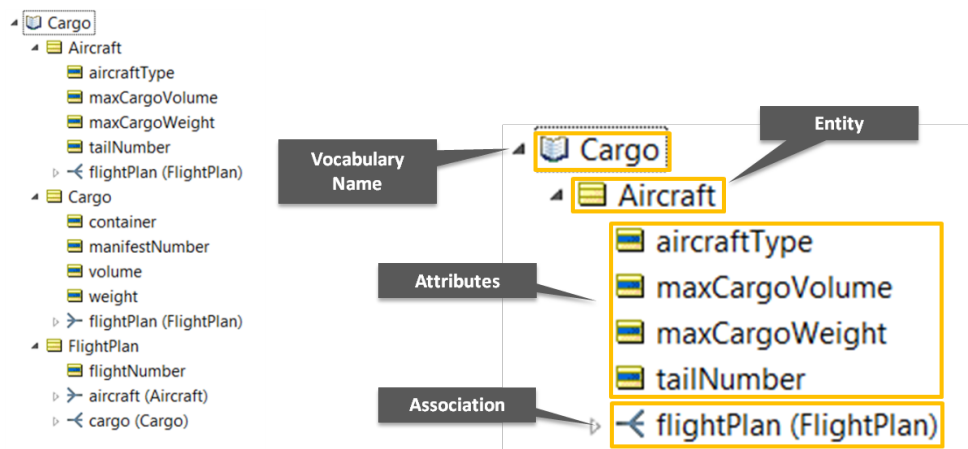
How business terms are represented in the Corticon Vocabulary

Business terms are represented in the Vocabulary as either entities or attributes.

An attribute is like a data field that holds a value. An entity is a collection of attributes. For example, the term cargo may have data fields such as cargoID, weight, volume, etc. So, cargo should be defined as an entity in a Vocabulary and the terms cargo ID, weight, and volume should be defined as attributes that belong to the entity cargo.

Relationships between entities are represented as associations in a Vocabulary. For example, since a vehicle can carry multiple cargo containers, you could create a one-to-many association between the vehicle entity and the cargo entity in a Vocabulary.

The Vocabulary tree



A Vocabulary is represented as a hierarchical tree.

The top-most level, or root node of the tree, is the name of the Vocabulary. It is denoted by an 'open book' icon (Cargo in the example).

One level under the Vocabulary name are entities. Entities are denoted by a 'golden box' icon (Aircraft, Cargo, FlightPlan). Each entity name must be unique in the Vocabulary.

A level below each entity are its attributes. Each attribute has a name, a data type, and a few other properties. It is denoted by a golden box icon with a green bar in the middle (aircraftType, maxCargoVolume, etc).

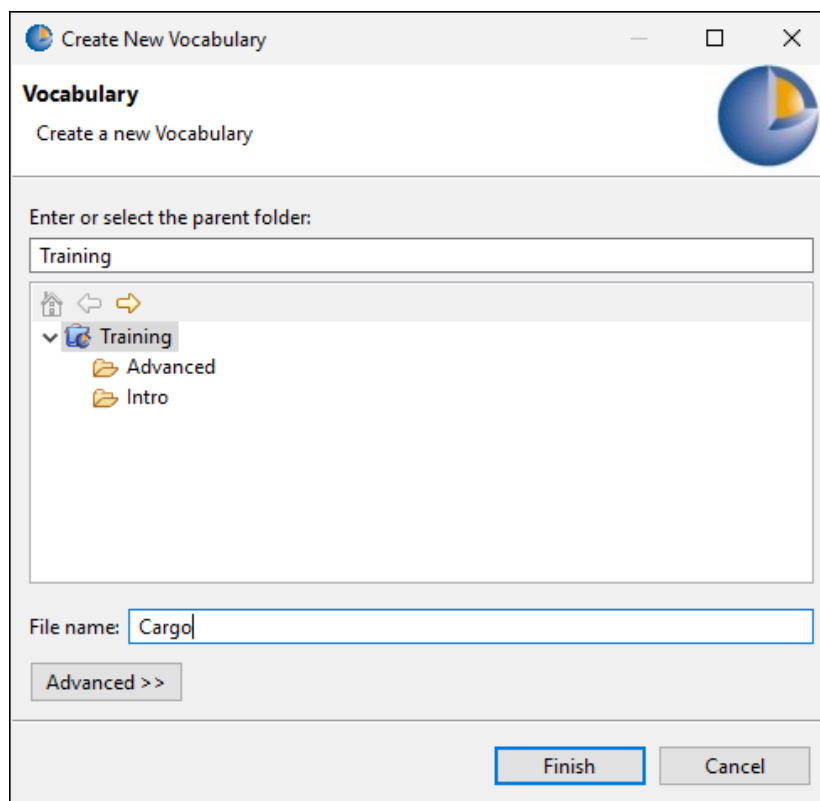
Finally, associations are at the same hierarchical level as attributes. The associated entity is displayed in parentheses. The association icon changes based on the type of association. Since the Cargo entity has a many-to-one relationship with the FlightPlan entity, the icon is multiple bars converging into a single horizontal bar. The FlightPlan to Cargo association shows a reverse of the icon (a single bar diverging into multiple bars).

Tasks to define a Vocabulary

To define a Vocabulary:

- Create a new Vocabulary or import an existing Vocabulary.
- Add entities to the Vocabulary.
- Add attributes to entities.
- Add associations to entities.


Create a Vocabulary file



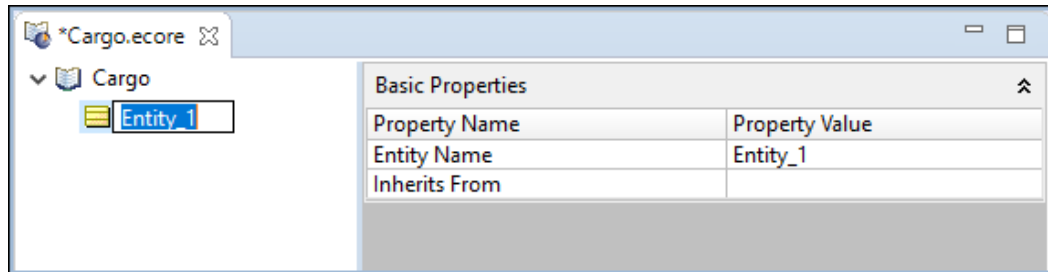
You must create a Vocabulary as part of a rule project. The Vocabulary file opens in the Vocabulary editor in and has the extension .ecore.

Follow these steps to create a Vocabulary file:

1. Select File > New > Rule Vocabulary. The Create New Vocabulary wizard opens.
2. In the Create New Vocabulary wizard:
 - a. Select the rule project and, optionally, a folder within the rule project in the Enter or select the parent folder area.
 - b. Enter a name for the Vocabulary file in the File name field.
 - c. Click Finish.

The Vocabulary file is created and appears under the rule project in the Rule Project Explorer view. It is represented by the icon . The file automatically opens in the Vocabulary editor.

Add an entity



Before you add an entity, you must determine the name that you want to give to it. Every entity name must be unique within the Vocabulary. It cannot contain any spaces. Certain terms such as size, sum, and today are part of a predefined set of Rule Operators.

Note: The Operator Vocabulary contains predefined operators such as =, >, and <, that are required to build business rules.

They cannot be used as entity names. Refer to the “Vocabularies” section of the Quick Reference Guide for more information on restricted names.

Follow these steps to add an entity in the Vocabulary editor:

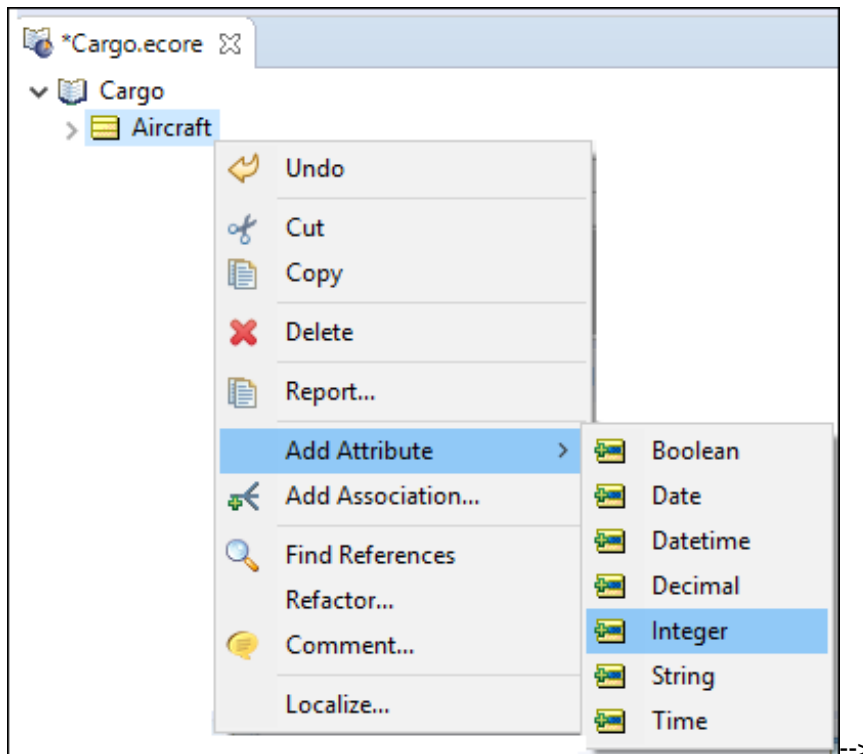
1. In the Vocabulary editor, right-click the Vocabulary name (with an ‘open book’ icon next to it) and select Add Entity.
2. An entity with the default name Entity_1 is created. The entity name is automatically editable. *Hint:* If it is not editable, double-click the default name Entity_1.
3. Delete Entity_1 and enter the name of the business term Aircraft as the entity name.

Add attributes to an entity

After you create an entity, you can add attributes to it. An attribute is similar to a data field or a variable, whose values are populated at runtime.

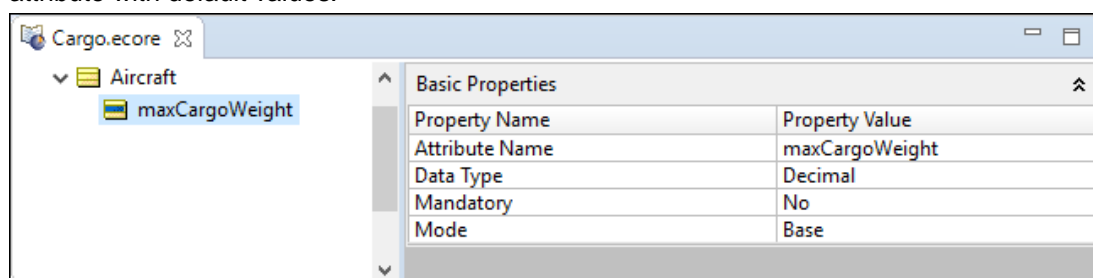
Follow these steps to add attributes to an entity in the Vocabulary editor:

1. Right-click the entity name (with the 'golden box' icon next to it) and then select Add Attribute.
2. Choose from seven data types displayed in the side list—Boolean (True or False), Decimal, DateTime, Date, Integer, String, Time.



3. For this example, choose Decimal. An attribute with the default name Attribute_1 is created. The attribute name is automatically editable. Hint: If it is not editable, double-click the default name Attribute_1.
4. Delete Attribute_1 and enter the attribute name. You can give any name that is meaningful to you. However, note that attribute names must be unique for each entity. For this example, type in MaxCargoWeight. (no spaces!)

After you add an attribute, the property editor panel of the Vocabulary editor displays all the properties for the attribute with default values.



Each attribute has the following properties:

- Attribute Name—automatically populated when you define the attribute.
- Data Type—assigned when you created the attribute, you can change the data type in the drop-down list—Boolean (True or False), Decimal, DateTime, Date, Integer, String, Time. Note: You can also create custom data types using the Custom Data Types tab. This is covered in the course Advanced Decision Modeling with Progress Corticon Studio.

- **Mandatory**—by default this property is set to No. Select Yes if the attribute value cannot be null and must be populated at runtime. Retain No if the attribute value can be null.
- **Mode**—by default, this property is set to Base. The Base setting enables the attribute to be used by systems outside Corticon. In most cases, you should retain the default setting. For more information about this property, refer to the Corticon documentation.

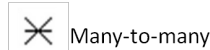
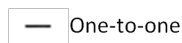
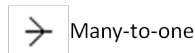
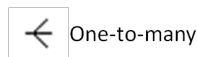
You can add a comment to any Vocabulary attribute, entity, association, and domain. Comments are listed by date and can be set as Note, TODO, ChangeLog, or NeedsReview.

Try adding a comment to Cargo.weight:

1. Right-click on weight in the Cargo entity, and then choose Comment.
2. In the Comment windows, type “Weight in whole kilograms.” and click OK.

Types of associations

An association describes the relationship between two entities. In a Corticon Vocabulary, you can define the association under either entity. By default, the association that you define appears under both entities in the Vocabulary tree.



There are four types of associations:

- **One-to-many**—for example, if a vehicle can contain many cargo containers, from the perspective of the Vehicle (source) entity, the relationship with the Cargo (target) entity is one-to-many. A one-to-many association's icon is a single bar diverging into multiple branches.
- **Many-to-one**—for example, from the perspective of the Cargo (source) entity, the relationship with the Vehicle (target) entity is many-to-one. A many-to-one association's icon shows multiple branches converging into a single bar.
- **One-to-one**—for example, if a vehicle can have only one driver, the relationship is one-to-one. This is true from both perspectives. A one-to-one association's icon is a single bar.
- **Many-to-many**—for example, multiple workers may be assigned to load multiple vehicles through the workday. Each worker may load multiple vehicles, and each vehicle may have multiple workers assigned to load it. A many-to-many association's icon shows multiple branches converging to a point from both sides.

Add associations to an entity

When you define an association, you can add it under either of the two entities in the relationship. You add an association to an entity just as you add an attribute. You must right-click the entity and select Add Association.

This brings up the Association dialog box in which you set the association's properties. In most cases, you only need to look at these properties:

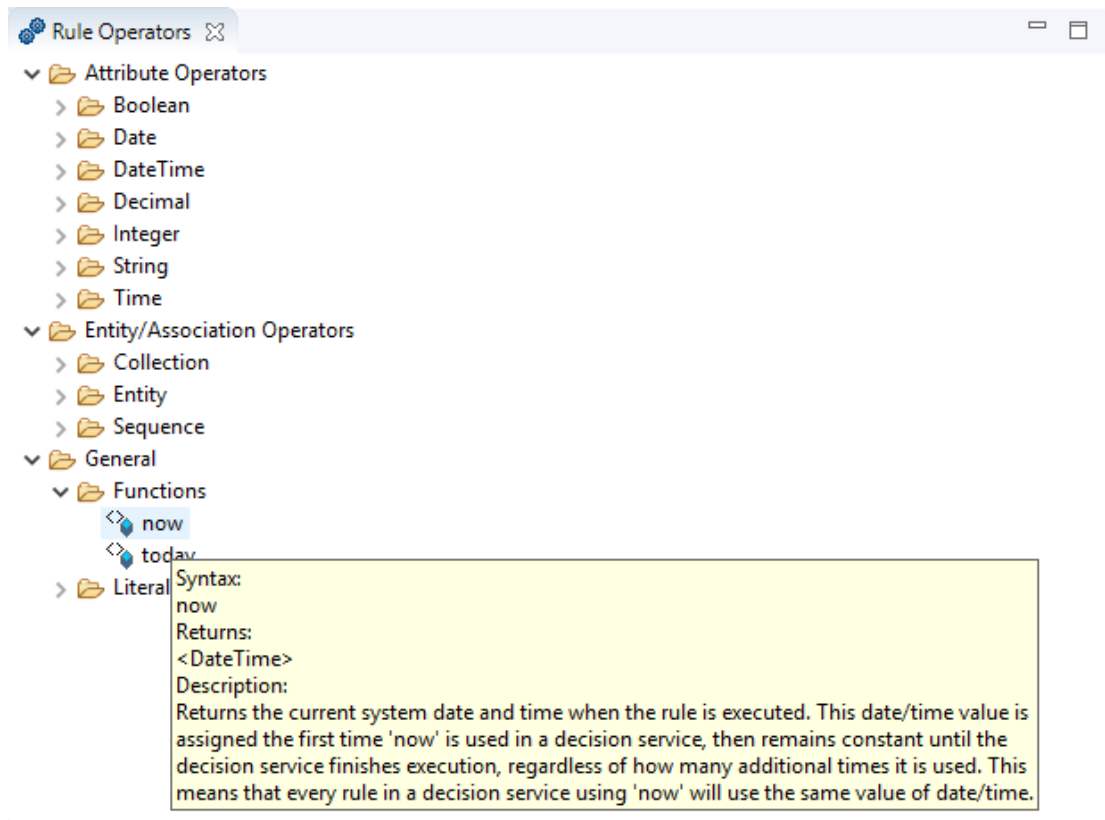
- Source Entity Name—by default, the entity to which you are adding the association is selected as the source entity. You should retain this default setting.
- Source—select One or Many depending on the role you want the source entity to play in the relationship.
- Target Entity Name—select the target entity from the drop-down list.
- Target—select One or Many depending on the role you want the source entity to play in the relationship.

When you complete setting the properties, click OK.

The association will be displayed in the Vocabulary tree under both entities with the relevant association icon.

Rule Operators

When you use your Vocabulary to build rules, you also make use of rule operators such as =, <, and >. Corticon Studio provides a rich set of predefined rule operators in the Rule Operators view. By default, this view is located at the bottom-left of the Corticon Designer perspective.

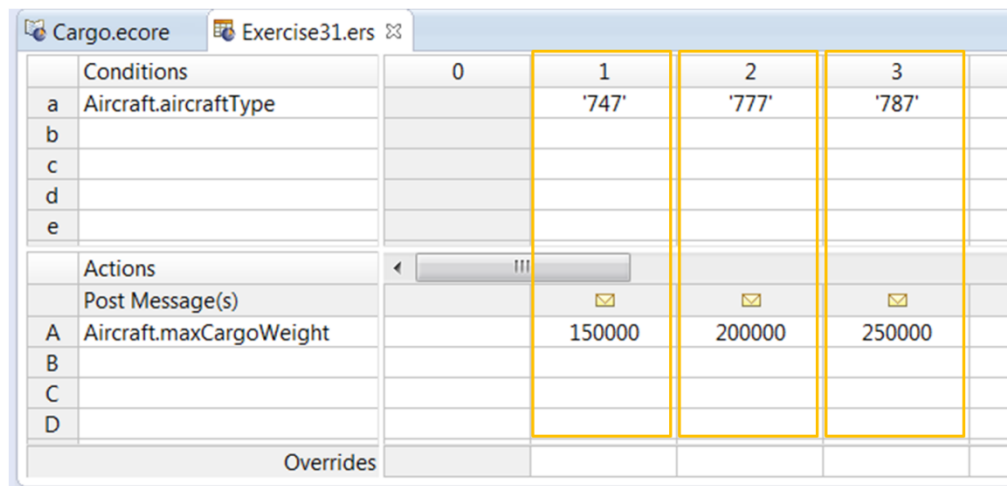


The Rule Operators view organizes operators into folders based on their function and purpose. For example, the Attribute Operators folder contains subfolders for different data types. The folder for each data type contains rule operators that you can use when defining rules that use attributes of that data type.

You can learn about each rule operator in the Rule Operators view by hovering the mouse pointer over it. A tooltip appears, describing the rule operator, as shown in this image. The Rule Language Guide in the documentation set provides details of all rule operators available.

What is a Rulesheet?

You define your rule logic in a Corticon Rulesheet. A rule is like an 'if-then' statement. Each rule consists of one or more conditions (if) that are associated with one or more actions (then).



Conditions		0	1	2	3
a	Aircraft.aircraftType		'747'	'777'	'787'
b					
c					
d					
e					
Actions					
	Post Message(s)		✉	✉	✉
A	Aircraft.maxCargoWeight		150000	200000	250000
B					
C					
D					
Overrides					

This Rulesheet has three rules. The Rulesheet editor has the following parts:

- **Conditions**—where you define the conditions for each rule. For example, `Aircraft.aircraftType = 747`. The condition value could be a single value (747), a set of values (747, 777, 787), or a range of values (`weight=100000..200000`).
- **Actions**—where you define the actions that need to be triggered when the conditions are satisfied. For example, `Aircraft.maxCargoWeight=150000`.
- **Rule columns**—the highlighted columns in this image. Each column represents a rule. It associates a set of conditions with a set of actions. For example, column 1 defines the rule—if the aircraft is a 747, then its maximum cargo weight is 150,000.

The terms `Aircraft.aircraftType` and `Aircraft.maxCargoWeight` come from the Rule Vocabulary. Each Rulesheet must be linked to a Rule Vocabulary.

Corticon evaluates all the conditions in each rule. If all the conditions in the rule are satisfied, the actions in the rule are triggered.

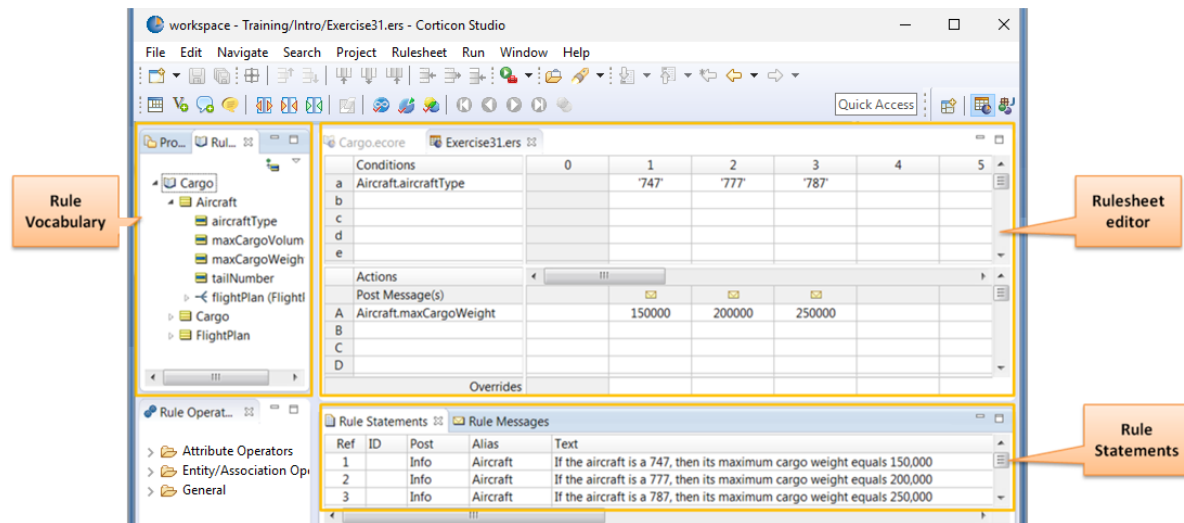
Note: If an action does not execute for some reason, Corticon still tries to execute the other actions in the rule.

For details, see the following topics:

- [Tasks to define rules in a Rulesheet](#)
- [Create a Rulesheet file](#)
- [Define a rule statement](#)
- [Define rules in the Rulesheet editor](#)
- [Link a rule with a rule statement](#)
- [How to analyze rules in a Rulesheet](#)

Tasks to define rules in a Rulesheet

To define a Rulesheet you will use the following views:



Perform the following tasks to define rules in a Rulesheet:

1. Create a Rulesheet file.
2. Describe each rule informally in a rule statement. Rule statements help document the purpose of a rule. They can also be posted as messages to a client application when the rule fires.
3. Define the rules by dragging terms from the Rule Vocabulary view to the Conditions and Actions sections in the Rulesheet editor and defining values for each rule in the rule columns.
4. Link each rule statement with its rule.


Create a Rulesheet file

You create a Rulesheet file just like you create a Vocabulary file. A Rulesheet file has the extension .ers.

Note: A Rulesheet file must be linked to a Vocabulary file. When you create a Rulesheet file, you create it for a rule project. If a Vocabulary exists in the same rule project (before you create the Rulesheet file), Corticon Studio and Corticon.js links it to the Rulesheet by default.

Follow these steps to create a Rulesheet file:

1. In Corticon Studio, select File > New > Rulesheet.
2. In the Create New Rulesheet wizard:
 - a. Enter a name for the Rulesheet in the File name field.
 - b. Specify the rule project or folder in which the Rulesheet must be created in the Enter or select the parent folder field.
 - c. Click Finish.

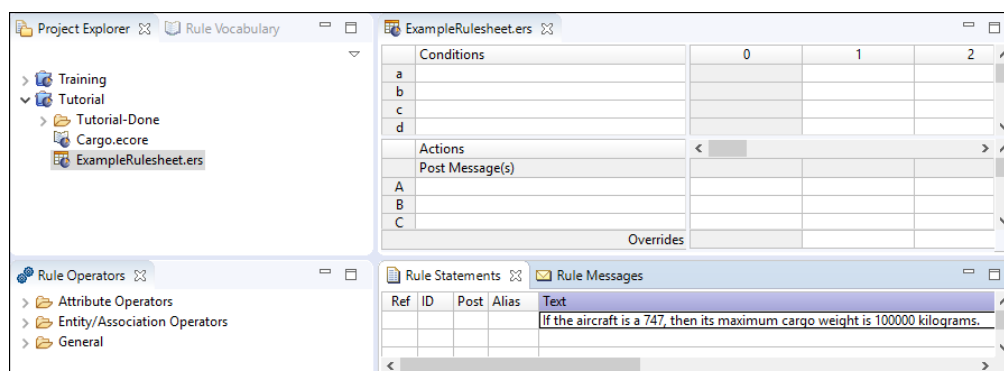
The newly created Rulesheet file appears in the Project Explorer view. It is represented by the icon . The Rulesheet also opens by default in the Rulesheet editor. In addition, the Rule Vocabulary view and the Rule Statements view become active.

Define a rule statement

A rule statement serves the following needs:

- Writing the rule in plain or informal language enables you to articulate the rule quickly and prepares you for the next step—defining the rule formally. You can also use the rule statement to identify Vocabulary terms that you need in the rule.
- The rule statement can be made a part of the output message sent by Corticon when the rule fires. The rule statement describes the rule, so users can understand the action or actions that are triggered by the rule by reading the rule statement in the output message.
- Finally, a rule statement is a way to document a rule so that stakeholders can understand its purpose and logic by reading the rule statement.

Rule statements appear in the Rule Statements view.



Note: You can define multiple rule statements for a single rule. For example, one rule statement can document the rule, while another is sent as part of the output message when the rule fires.

Follow these steps to create a rule statement:

1. Open the Rulesheet. The Rule Statements view becomes active.
2. The Rule Statements view comprises several rows and columns. To create the rule statement, double-click a cell in the column named Text and type the rule statement.

Define rules in the Rulesheet editor

A Rulesheet typically contains multiple rules. This Rulesheet has two rules.

	Conditions	0	1	2	3	4	
a	Aircraft.aircraftType		'747'	'777'	-		
b							
c							
d							
	Actions	< >					
	Post Message(s)						
A	Aircraft.maxCargoWeight		150000	200000			
B							
C							
	Overrides						

To define rules in the Rulesheet, you:

- Drag and drop Vocabulary terms for conditions and actions as follows:
 1. Select the Vocabulary term from the Rule Vocabulary view.
 2. Drag and drop the term to the next empty cell in the Conditions or Actions pane. Each Vocabulary term must go into a separate row.
- Specify condition and action values for each rule in the rule columns as follows:
 1. Double-click the cell that corresponds with the Vocabulary term.
 2. Enter the value.

When you specify a value in a rule column cell, the equality operator is implied. For example, when you enter 747 in column 1 as shown here, it means (if) Aircraft.aircraftType = '747'.

Note: The values that you enter must conform to the rules of syntax enforced by Corticon.

Syntax for values in rule column cells

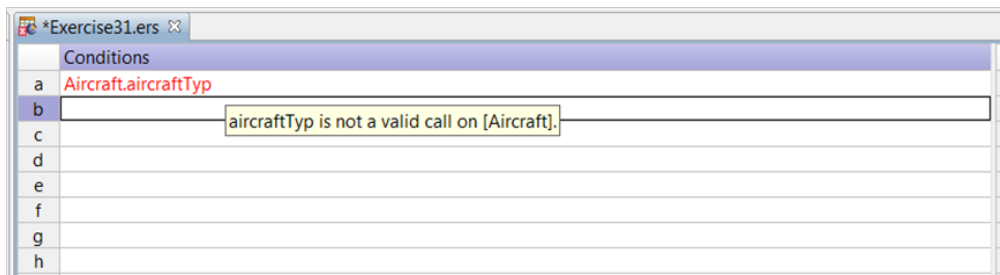
The values that you specify must conform to the following syntax:

- If the Vocabulary term's data type is String, Date, DateTime, or Time, the corresponding value in the rule column must be enclosed in single quotes.
 - CORRECT: 'apple'

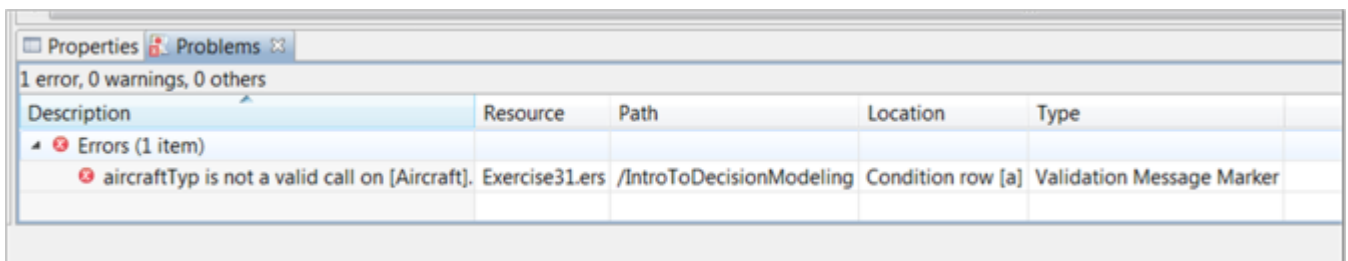
- INCORRECT: "apple", apple
- If the Vocabulary term's data type is Integer, the corresponding value must not be enclosed in quotes and it must not contain decimal points or delimiters.
 - CORRECT: 4, -4
 - INCORRECT: 4.0, '4'
- If the Vocabulary term is Decimal, the corresponding value can contain a decimal point, but the decimal point is optional. Commas are not allowed, and the value must not be enclosed in quotes.
 - CORRECT: 10, -10.0, 10.5
 - INCORRECT: 10,25, 1,025.00
- If the Vocabulary term is Boolean, the corresponding value can only be T, F, true, or false. The value must not be enclosed in quotes. The case does not matter.
 - CORRECT: t, F, FALSE
 - INCORRECT: 'false'

Syntax errors in a Rulesheet

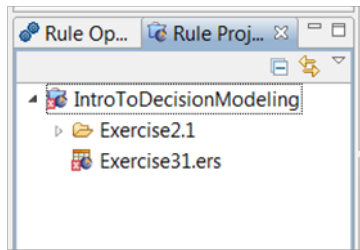
If you make any syntax errors while defining a rule, Corticon Studio detects the error and highlights it in red. If you hover your mouse pointer over the error, a tooltip appears, describing the problem. This enables you to understand the cause of the error and resolve it.



A description of the error is also displayed in the Problems view. It describes the causes of errors in all rule-modeling components that are open.

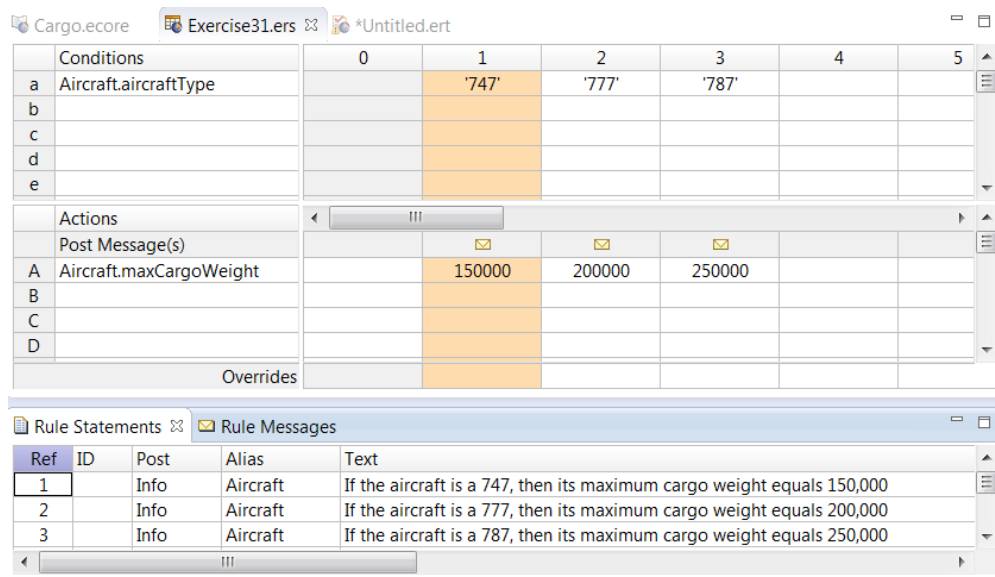


The project folder and the Rulesheet in the Rule Project Explorer view are also marked with error symbols.



Link a rule with a rule statement

You must explicitly link each rule statement with its rule. To link a rule statement with its rule, enter the number of the rule column in the rule statement's Ref cell.



If you only want to document the rule, you do not need to configure any other properties.

If you want to post the rule statement as part of the output message, configure these properties:

- **Post**—The Post property enables you to indicate the severity of the rule violation. You can select from three levels—Info, Warning, and Violation. Note that these severity levels are user-defined and have no special meaning in Corticon. You can define them to match your requirements.
- **Alias**—Specify the Vocabulary entity in the rule condition that you want displayed with the output message when there is a rule violation.

After you link a rule statement with a rule, when you select the rule statement in the Rule Statements view, the corresponding rule column is highlighted in orange. Similarly, if you select a cell or a group of cells within the rule column, the corresponding rule statement is highlighted in orange. This indicates that the rule statement and the rule are linked.

How to analyze rules in a Rulesheet

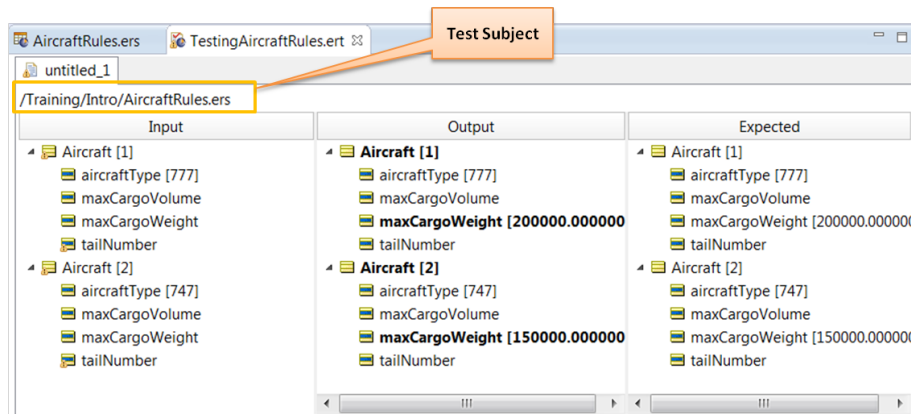
Corticon provides tools that help you analyze rules in your Rulesheet by:

- Checking for conflicts
- Checking for completeness
- Identifying dependencies and loops

These features are covered in the course [Advanced Guided Journey](#).

What is a Ruletest?

A Ruletest simulates a business scenario.



The Ruletest input data is evaluated by the rules.

AircraftRules.ers				
Conditions		0	1	2
a	Aircraft.aircraftType		'747'	{ '777', '787' }
b				
c				
d				
e				
f				
q				
Actions				
Post Message(s)			✉	✉
A	Aircraft.maxCargoWeight		150000	200000
B				
C				

If the data satisfies all the conditions in a rule, the rule fires and some output containing the results of the rule execution is produced.

You can define different sets of input data to test how the rules behave in different scenarios. You can also use a Ruletest to compare the output of a rule execution with expected results.

A Ruletest stores this information in a Ruletest file, enabling you to save use-cases that are of interest, change rules, and run the test again to see how the modified rules behave when applied to the same use-cases.

Take a close look at the images. Here is an example of a Rulesheet (AircraftRules.ers) with two rules and a Ruletest (TestingAircraftRules.ert) that has been executed.

The Rulesheet has rules that define a value for Aircraft.maxCargoWeight based on the value of Aircraft.aircraftType received in input data.

The Ruletest tests the Rulesheet. The Ruletest editor has four parts:

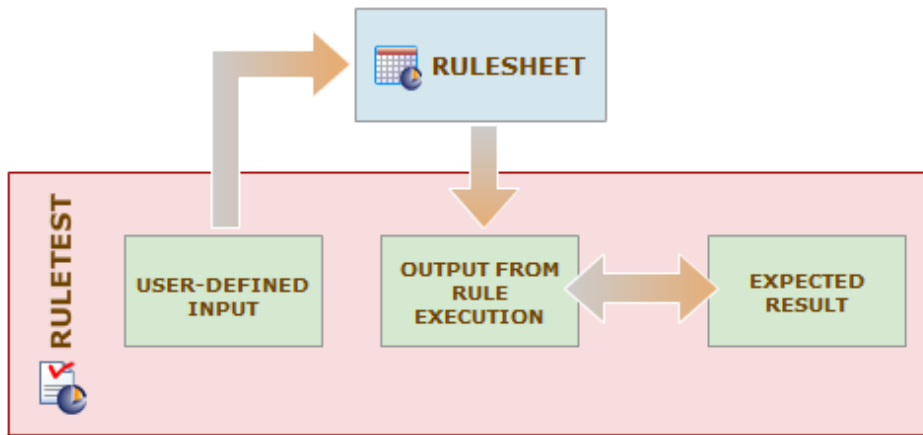
- Test Subject—specifies which Rulesheet or Ruleflow is being tested (in this case AircraftRules.ers).
- Input—where you define input data to be processed by the rules in the Rulesheet. In this example, there are two instances of the Aircraft entity with different values for Aircraft.aircraftType.
- Output—where Corticon Studio displays the result of a Ruletest execution. As you can see, a value for Aircraft.maxCargoWeight has been assigned based on the rules in the AircraftRules.ers Rulesheet.
- Expected—where you can optionally define the result that you expect.

For details, see the following topics:

- [How a Ruletest works](#)
- [Tasks to test rules in a Rulesheet using a Ruletest](#)
- [Create a Ruletest file](#)
- [How to specify input for a Ruletest](#)
- [How to specify expected results](#)
- [How to run a Ruletest](#)
- [Compare the output with expected results](#)
- [How to embed dynamic data in a rule statement](#)
- [How to link a rule statement to multiple rules](#)

How a Ruletest works

When you run a Ruletest the input values are used to generate output that you can compare to your expected results.



These steps details how the rules generate the output values.

1. Input data is processed by the rules in the Rulesheet.
2. If the input data satisfies all the conditions in one or more rules, those rules fire. The Ruletest then displays output that could include the same data as the input but with changed values, and/or additional data and values.
3. If the input data does not trigger any rules, the Ruletest displays the unchanged input data as the output.
4. If you specify expected results for a test, the Ruletest automatically compares the output with the expected results and displays any differences through color-coding.

Tasks to test rules in a Rulesheet using a Ruletest

Perform the following tasks to test rules using a Ruletest:


1. Create a Ruletest file.
2. Specify inputs for the test.
3. Optionally, specify expected results to compare with the output of a test execution.
4. Run the test.

Create a Ruletest file

A Ruletest file must be created under a rule project and must refer to a Rulesheet. A Ruletest file has the extension .ert.

Follow these steps to create a Ruletest file:

1. In Corticon Studio, select File > New > Ruletest. The Create New Ruletest Wizard opens.
2. In the Create New Ruletest wizard's Create a new Ruletest page:
 - a. Ensure that the Enter or select parent folder field contains the path under which you want the Ruletest created.
 - b. Enter the name of the Ruletest in the File name field.
 - c. Click Next. The Select Test Subject page opens.
3. In the Select Test Subject page:
 - a. Ensure that the correct Rulesheet is selected in the Local section.
 - b. Click Finish.

The newly created Ruletest should now appear in the Rule Project Explorer view. It is represented by the icon . The Ruletest should also open by default in the Ruletest editor.

How to specify input for a Ruletest

In the business world, a Corticon Decision Service may receive input in different formats such as XML, JSON, Java, or .NET objects. However, in a Ruletest, you must specify input data in the Corticon data format.

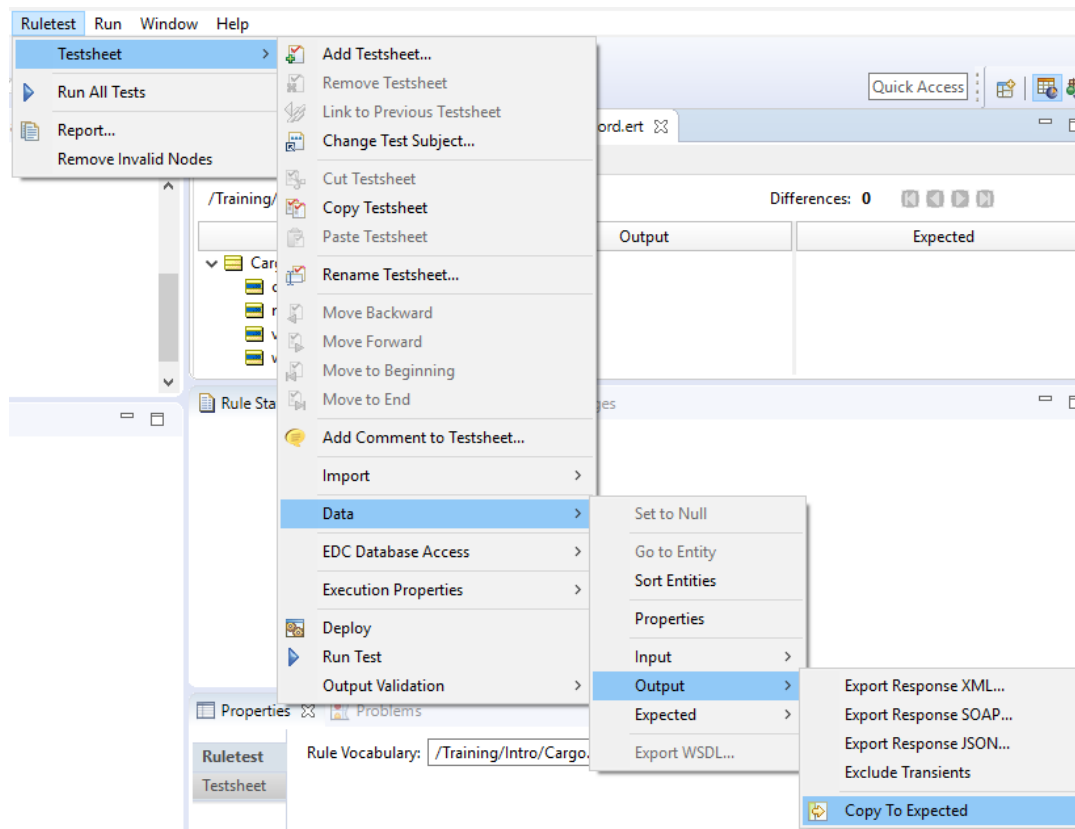
You specify inputs for a Ruletest by:

1. Dragging and dropping entities from the Rule Vocabulary view to the Input pane. Each time you drag and drop an entity to the Input pane, you create an instance of the entity. This adds all the attributes in the entity to the Input pane. However, you can delete attributes that you do not require.
2. Specifying values for attributes. To do this, double-click the attribute name and enter the value. The syntax for specifying values is similar to specifying values for attributes in Rulesheets, with one difference—you must not enter any values in quotes, even if the data type of the attribute is String, DateTime, Date or Time. If you enter quotes, the Ruletest treats it as part of the value. Note: If you break any of these rules while specifying values for attributes in the Input pane, Corticon Studio indicates a mismatch with a warning icon.

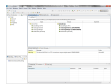
You can specify different entities as well as multiple instances of the same entity. When you specify multiple instances of the same entity, each entity is assigned an identity number. The identity number is 1 for the first instance of the entity, and is incremented by 1 for each additional instance that you drag and drop.

How to specify expected results

After you define input, you can optionally define expected results in the Expected pane. To do this, drag and drop elements from the Rule Vocabulary view to the Expected panel and specify values, just like you do when you are defining input data.



Note: After you run the Ruletest, you can copy data in the Output pane to the Expected pane. This is useful if you want to make some changes to the rules and see if the output differs from the output of the previous test. To do this, select Ruletest > Testsheet > Data > Output > Copy To Expected or click the Copy To Expected button:



How to run a Ruletest

When you run a Ruletest, Corticon:

- Compiles the Rulesheet. This happens only if the Ruletest is being run for the first time on the Rulesheet, or if the Rulesheet has been modified since the last test execution.
- Applies the rules in the Rulesheet to the input data.

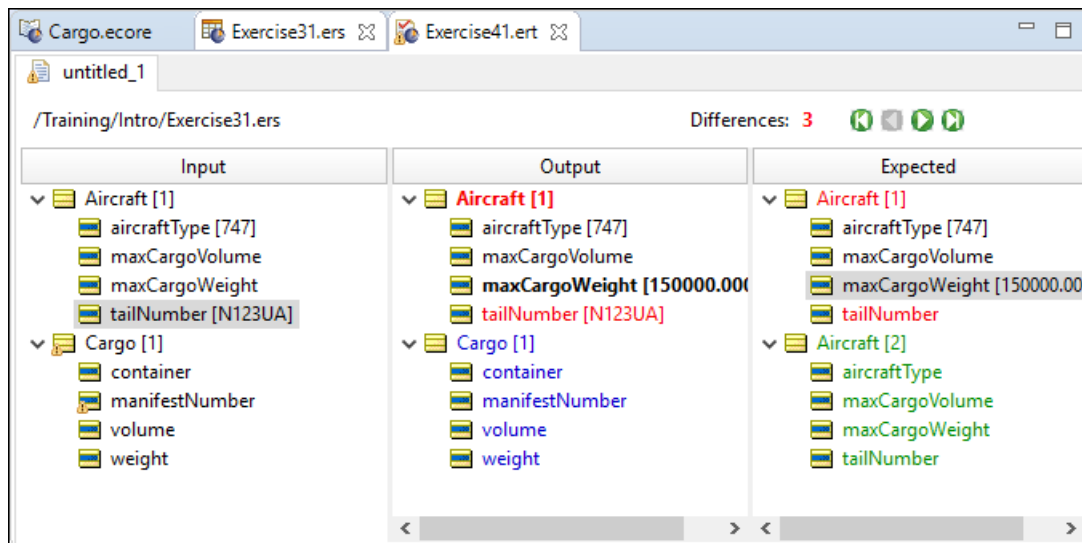
Follow these steps to run the Ruletest:

1. Ensure that the Ruletest is open.
2. Click the Run Test icon.

The Output panel displays the result of the Ruletest execution. Any elements that are modified or added (as a result of the actions in the rule) are highlighted in bold text. If any rule statements are linked to the rule and are configured for posting, they are displayed in the Rule Messages view.

Note: If no rule fires, the Output panel displays the same elements and values as in the Input panel, without any highlights. The Rule Messages view remains blank.

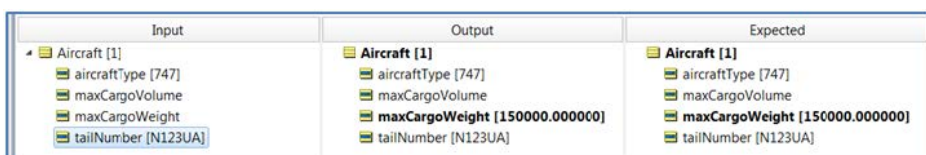
Any differences between the Output pane and the Expected pane are highlighted through color codes as shown in this image.



Compare the output with expected results

The Ruletest displays differences between the Output pane and the Expected pane as follows:

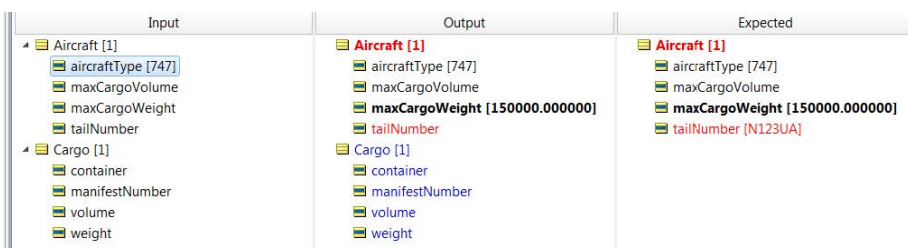
- If there are no differences, both sets of data are shown in black in both panes.



- If one or more attribute values differ for the same entity instance, the entity and the attributes are shown in red in both panes.



- If additional entity instances are produced in the output (that do not exist in the Expected pane), they are shown in blue in the Output pane.

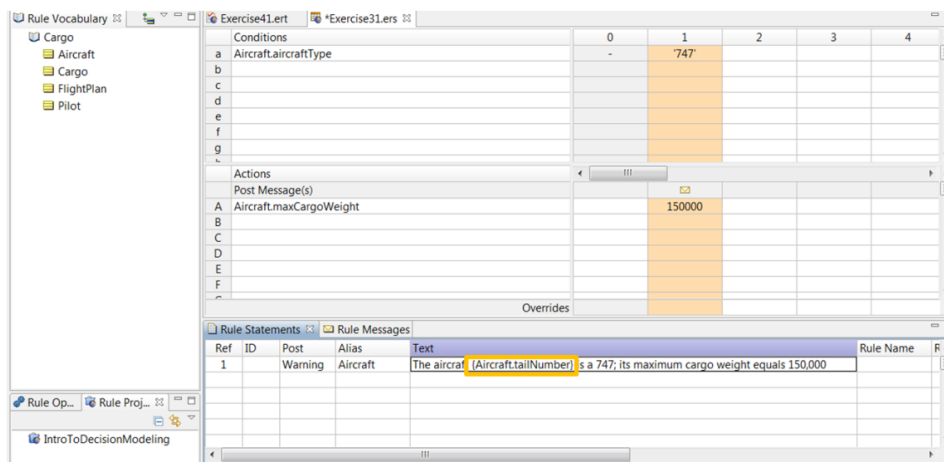


- If the Expected pane contains entity instances that are not produced in the Output pane, they are displayed in green in the Expected pane.

Input	Output	Expected
<ul style="list-style-type: none"> Cargo [1] <ul style="list-style-type: none"> container manifestNumber volume weight 	<ul style="list-style-type: none"> Cargo [1] <ul style="list-style-type: none"> container manifestNumber volume weight 	<ul style="list-style-type: none"> Aircraft [1] <ul style="list-style-type: none"> aircraftType [747] maxCargoVolume maxCargoWeight [150000.000000] tailNumber [N123UA]

How to embed dynamic data in a rule statement

Rule statements in a Rulesheet can be used to generate rule messages. Rule messages need not contain only static information. They can also contain dynamic data such as the values of attributes (in request messages received by Corticon or created/modified as the result of a rule execution at runtime).

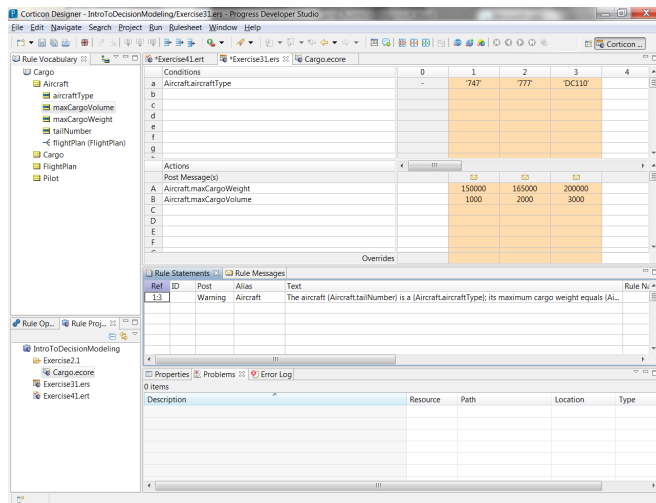


For example, the rule statement 'If an aircraft is of the type 747, the maximum cargo weight is 150,000' could be modified to include the aircraft's tail number when a rule message is generated: 'The aircraft N123UA is of the type 747; its maximum cargo weight is 150,000'. Here, the value N123UA is the value of the attribute tailNumber in a request message received by Corticon at runtime.

You can embed dynamic data in rule messages by specifying the attribute name within the rule statement. To specify an attribute name in a rule statement, open the Rulesheet. In the Rule Statements view, specify the attribute name by dragging and dropping it from the Rule Vocabulary view to the appropriate place within the rule statement. The attribute name is enclosed in curly braces indicating that its value will be populated at runtime.

How to link a rule statement to multiple rules

Using dynamic data, you can create a single rule statement that describes multiple rules. You can then link the rule statement to each of the rules. When any of the rules fire, attribute names will be replaced by actual values in the rule message.



For example, if you have three rules, as shown. Each of these rules specifies the maximum cargo weight for a certain type of aircraft. A single rule statement that embeds the aircraft tail number, the aircraft type, and the maximum cargo weight, can be used to cover all three rules.

There are a number of ways to link a rule statement to multiple rules. Here are two common methods:

- `<ColumnNumber>:<ColumnNumber>`— this specifies a range of rule columns (for example 1:3, where 1 and 3 are inclusive). When any of these rules fires, the rule statement is posted as a rule message. This is the method shown in the image above.
- `{<ColumnNumber>, <ColumnNumber>, ...}`—this specifies a list of rule columns (for example {1, 3}). When any of the rules in the list fires, the rule statement is posted as a rule message.

Enhance rules

Comprehensive business rules require a rule modeling tool that supports the necessary comparison operators, equations, calculations and logic operation required to capture the complexity of your business scenarios. In this content you explore the following ways to enhance your rules.

For details, see the following topics:

- [Action-only rules](#)
- [Comparison operators](#)
- [Equations and calculations](#)
- [Value sets and ranges](#)
- [Boolean conditions](#)
- [Logical structures](#)
- [How to check for null values](#)

Action-only rules

As you develop rule models in Corticon, you may encounter situations where you require a rule that always fires regardless of conditions, such as a rule that sets a default value for an attribute. For example, building on the transportation example, the transportation company may require a rule that sets the maximum cargo weight for an aircraft, regardless of the aircraft type. You can address situations like this by defining action-only rules.

The screenshot shows a Rulesheet editor with two tabs: 'Action-only.ers' and '*Action-only.ert'. The main area is a table with columns labeled 0, 1, and 2. The first column is divided into 'Conditions' and 'Actions' sections. The 'Conditions' section has rows labeled a through i, all of which are grayed out. The 'Actions' section has rows labeled A through E. Row A is selected and contains the action 'Aircraft.maxCargoWeight' with a default value of '100000' in column 0. Columns 1 and 2 are empty for all rows.

	0	1	2
Conditions			
a			
b			
c			
d			
e			
f			
g			
h			
i			
Actions			
Post Message(s)			
A Aircraft.maxCargoWeight	100000		
B			
C			
D			
E			

An action-only rule is a non-conditional rule—it does not contain any conditions, it contains only actions and it always fires as the following Rulesheet demonstrates.

The screenshot shows the 'untitled_1' Rulesheet with a tab for '*Action-only.ert'. The main area is a table with columns labeled Input, Output, and Expected. The 'Input' column shows three aircraft objects: Aircraft [1], Aircraft [2], and Aircraft [3]. Each aircraft object has four attributes: aircraftType, maxCargoVolume, maxCargoWeight, and tailNumber. The 'Output' column shows the results of the rule execution for each aircraft. The 'Expected' column is empty. The output for each aircraft is a list of attributes: aircraftType, maxCargoVolume, maxCargoWeight [100000.000000], and tailNumber. The maxCargoWeight attribute is highlighted in red in the output, indicating a change from its default value.

Input	Output	Expected
<ul style="list-style-type: none"> Aircraft [1] <ul style="list-style-type: none"> aircraftType [747] maxCargoVolume maxCargoWeight tailNumber 	<ul style="list-style-type: none"> Aircraft [1] <ul style="list-style-type: none"> aircraftType [747] maxCargoVolume maxCargoWeight [100000.000000] tailNumber 	
<ul style="list-style-type: none"> Aircraft [2] <ul style="list-style-type: none"> aircraftType [777] maxCargoVolume maxCargoWeight tailNumber 	<ul style="list-style-type: none"> Aircraft [2] <ul style="list-style-type: none"> aircraftType [777] maxCargoVolume maxCargoWeight [100000.000000] tailNumber 	
<ul style="list-style-type: none"> Aircraft [3] <ul style="list-style-type: none"> aircraftType [787] maxCargoVolume maxCargoWeight tailNumber 	<ul style="list-style-type: none"> Aircraft [3] <ul style="list-style-type: none"> aircraftType [787] maxCargoVolume maxCargoWeight [100000.000000] tailNumber 	

You define an action-only rule in column 0 of a Rulesheet. The cells that correspond to the Conditions panel in column 0 are grayed out. This indicates that you cannot specify any conditions in column 0. However, the cells that correspond to the Actions panel in column 0 are editable, enabling you to specify actions that always fire, regardless of conditions.

You can use column 0 to:

- Specify equations and calculations.
- Assign default values to attributes.
- Specify any action that you want performed whenever the Rulesheet is processed.

Note: Each Action row in column 0 is a separate rule and can have its own rule statement.

To define an action-only rule, drag and drop the required attribute to an action row. Then, enter the default value for the attribute in corresponding cell in column 0.

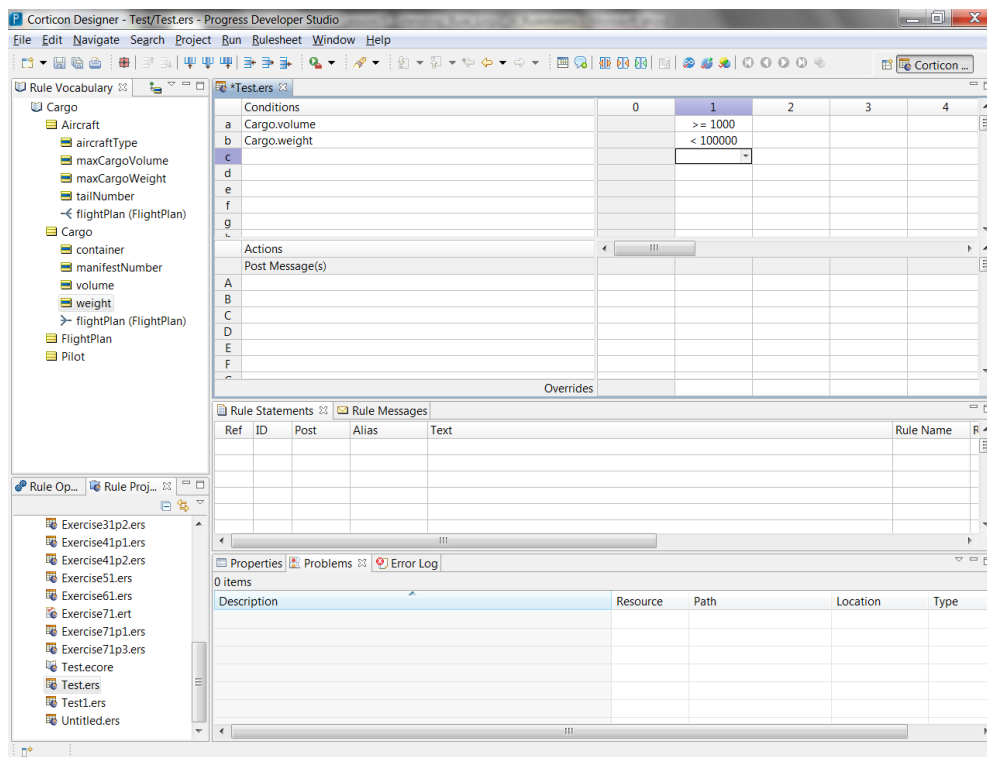
Comparison operators

You define rule conditions by specifying a Vocabulary term in a conditions cell and entering a value in a corresponding rule column cell. When you do this, the condition automatically assumes the equality '=' operator.

Conditions	0	1	2	3	4
a Aircraft.aircraftType	-	'747'			
b					
c					
d					
e					
f					
g					
l					
Actions	!!!				
Post Message(s)					
A Aircraft.maxCargoWeight		150000			
B					
C					
D					
E					
F					
G					
Overrides					

For example, when you specify Aircraft.aircraftType in a cell in the Conditions pane and enter 747 in a corresponding rule column cell, you define the condition Aircraft.aircraftType = '747'.

However, as you model rules, you may require different types of operators. For example, you may have to specify a condition that checks if an attribute value is greater than or less than a certain value (for example, cargo volume > 1000 or cargo weight < 100000). These types of operators, called comparison operators, are part of the library of rule operators that Corticon provides.



The types of comparison operators that you can use in a condition depend on the data type of the attributes in the condition. For instance, for an Integer data type you can use operators such as >, <, >=, and <=. To check which operators are applicable for each data type, refer to the Attribute Operators category in the Rule Operators view in Corticon.

You can use a comparison operator in a rule by double-clicking the rule column cell and entering the operator followed by the value.

Equations and calculations

As you model rules, you may run into situations where you need to derive the value of an attribute from a calculation. For example, in a supermarket where a customer gets a discount of 2% if the total price of items in their shopping cart is greater than or equal to 100 €. To facilitate this, you would need a rule that checks if the total price of all items is greater than 100 and if so, deduct the discount from it to arrive at a new total. This rule would look like this.

grocerStore.ecore Equations.ers *TestShoppingCart.ert		0	1
Conditions			
a	ShoppingCart.total		≥ 100
b			
c			
d			
e			
f			
g			
h			
Actions			
Post Message(s)			
A	$\text{ShoppingCart.total} = \text{ShoppingCart.total} - (\text{ShoppingCart.total} * 0.02)$	<input type="checkbox"/>	<input checked="" type="checkbox"/>
B			

As you can see, the equation is defined in the first action row. When you define an equation like this in an action row, the corresponding cells in rule columns turn into checkboxes. You must select the checkbox in the appropriate column (in this example, column 1), for the action to be triggered.

When the Rulesheet is processed at runtime, Corticon populates the attribute values on the right side of the equation, performs the calculation, and assigns the result of the expression to the attribute on the left side of the equation.

*Equations.ert		
untitled_1		
/Training/Intro/Equations.ers		
Input	Output	Expected
<ul style="list-style-type: none"> ShoppingCart [1] <ul style="list-style-type: none"> total [110.000000] 	<ul style="list-style-type: none"> ShoppingCart [1] <ul style="list-style-type: none"> total [107.800000] 	

Value sets and ranges

You can use values in rule columns in two ways:

- [Value sets](#)
- [Value ranges](#)

Value sets

When you define values for a condition or an action spanning multiple rule columns, you build a value set. A value set is a set of distinct values for a condition or an action. For example, in the following image, the condition Aircraft.aircraftType has two values (747 and 777) in its value set. The action Aircraft.maxCargoWeight also has two values (100000 and 125000) in its value set.

Conditions		0	1	2	3
a	Aircraft.aircraftType	-	'747'	'777'	
b					
c					
d					
Actions					
Post Message(s)					
A	Aircraft.maxCargoWeight		100000	125000	
B					
C					

As you build a value set by defining values across different rule columns, you will notice that each cell in the same row becomes a drop-down list when you click it. The drop-down list displays all the values that you enter in the value set.

Corticon Studio automatically completes a value set for a condition. For instance, if you have a condition with the single value < 200, Corticon Studio automatically adds >=200 to the value set.

Note: You can view the complete value set for a condition by hovering the mouse pointer over the condition. A tooltip appears displaying the complete value set.

This value is not a part of any rule and is not displayed in the rule columns by default. It is used when you run a completeness check on the Rulesheet.

Note: A completeness check calculates the set of all possible mathematical combinations of all values in all conditions. It then compares this set of possible combinations to those already specified in the Rulesheet.

How to use other in a condition value set

After you enter three values for a condition's value set, a value named **other** is added to the list of choices in the drop-down list, as shown in this image.

Conditions		0	1	2	3	4
a	Aircraft.aircraftType	-	'747'	'777'	'787'	
b						
c						
d						
Actions						
Post Message(s)						
A	Aircraft.maxCargoWeight		100000	125000	150000	
B						

Other is a special type of operator. It helps you recognize that other values may be unaccounted for, and gives you a simple way to detect them in rules. You can think of other as the operator that provides the ELSE clause of an IF-THEN-ELSE rule. It is available only for value sets that you build in condition rows.

In this example, the value set for Aircraft.aircraftType has three values—747, 777, and 787. However, there may be a case where data is received with an aircraft type outside this value set. To address such a case, you can use other in a rule that sets the maximum cargo weight of any aircraft that is not a 747, 777, or a 787.

Note that you can manually enter other at any point while defining a value set (you do not have to first specify three values in the value set). Other is not a String value, so single quotes are not required.

Value ranges

A common requirement while defining rules is to specify conditions that check for a range of values in a single rule column cell. For example, a rule that checks if the cargo volume is between 1000 and 2000 (a range of values), and assigns a container type.

Conditions		0	1	2
a	Cargo.volume		(1000..2000)	3000..4000 ▾
b				
c				
d				
e				
f				
g				
h				
Actions		◀ III ▶		
Post Message(s)				
A	Cargo.container		'standard'	'oversize'
B				

You specify a range of values by entering the starting value, followed by two dots or periods, followed by the ending value—for example 1000..2000—as shown in the image. You must enclose the value range in parentheses or square brackets as follows:

Syntax example	Description
1000..2000	No square brackets or parentheses. This specifies a range between 1000 and 2000 and includes the values 1000 and 2000.
(1000..2000)	Enclosed in parentheses. This specifies a range between 1000 and 2000 but excludes the values 1000 and 2000.
[1000..2000]	Enclosed in square brackets. As in the first example in this table, this specifies a range between 1000 and 2000 and includes the values 1000 and 2000.
(1000..2000]	An opening parenthesis and a closing square bracket. This specifies a range between 1000 and 2000, excludes 1000, but includes 2000.
[1000..2000)	An opening square bracket and a closing parenthesis, (the reverse of the previous example in this table). This specifies a range between 1000 and 2000, includes 1000, but excludes 2000.

Note: You must use a square bracket and a parenthesis together. You cannot start or finish a range with a square bracket or a parenthesis and leave the other end of the range empty. For instance, specifying 1000..2000) results in an error.

Boolean conditions

A Boolean condition is any condition that returns True or False. For example, you may want to specify a condition that checks if the weight of a cargo container is less than 5000. You can define a Boolean condition using a comparison operator as follows.

	Conditions	0	1	2
a	Cargo.weight < 5000		T	-
b	Cargo.volume > 10000		-	T
c				
	Actions	III		
	Post Message(s)			
A	Cargo.container		'standard'	'large'
B				
C				

You define a Boolean condition as follows:

Specify the condition expression in the Conditions row.

All the cells in that row (across all rule columns) will display three choices in a drop-down list—T (true), F (false), and hyphen '-'. Select true or false based on your rule logic. Corticon automatically inserts a hyphen in other cells indicating that they do not play a role in the Boolean condition.

Boolean conditions versus value sets

Some conditions can be expressed as Boolean conditions or through value sets. For example, a condition that checks if the aircraft type is 747 can be expressed as a Boolean condition or through a value set with a single value, as shown in this image.

Conditions	0	1
a Aircraft.aircraftType		'747'
b		
c		
d		
e		
f		
Actions	III	
Post Message(s)		
A Aircraft.maxCargoWeight		100000
B		
C		
D		

=

Conditions	0	1
a Aircraft.aircraftType='747'		T
b		
c		
d		
e		
f		
Actions	III	
Post Message(s)		
A Aircraft.maxCargoWeight		100000
B		
C		
D		

So when should you use a Boolean condition and when should you build a value set?

If you only have to define a rule that checks for a single value, as in the first example, you can use either approach.

However, if you have to define rules that share a common pattern and check for multiple values for an attribute, it is better to use value sets, as shown in these images.

Conditions	0	1	2	3
a Aircraft.aircraftType		'747'	'777'	'787'
b				
c				
d				
e				
Actions	III			
Post Message(s)				
A Aircraft.maxCargoWeight		100000	125000	150000
B				

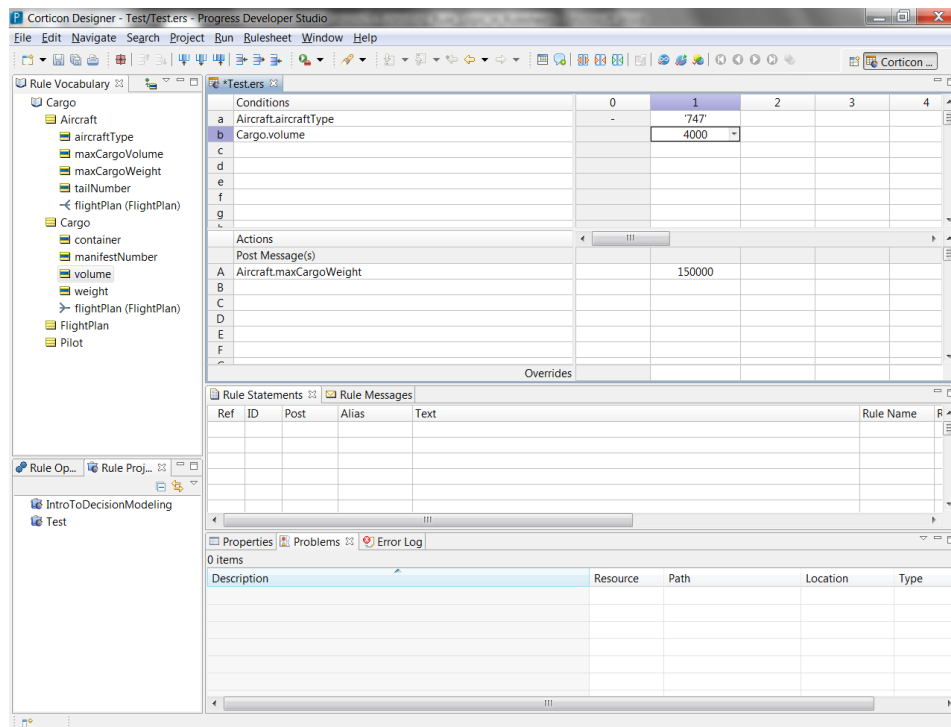
Conditions	0	1	2	3
a Aircraft.aircraftType='747'		T	F	F
b Aircraft.aircraftType='777'		F	T	F
c Aircraft.aircraftType='787'		F	F	T
d				
e				
Actions	III			
Post Message(s)				
A Aircraft.maxCargoWeight		100000	125000	150000
B				

Logical structures

You can define logical structures in rules:

- [Logical AND](#) on page 45
- [Logical OR](#) on page 45
- [Logical NOT](#) on page 46

Logical AND



A common requirement when you define rules is to specify AND logic in rules. For example, consider a situation where a rule must assign a maximum cargo weight for an aircraft only when the aircraft type is a 747 AND the cargo volume is 4000. You achieve this by defining two conditions and entering their values in the same rule column, as shown in this image. The rule fires only when all the conditions are satisfied.

Logical OR

Just like AND, a common requirement is to use OR logic in a Rulesheet. You can specify OR logic in two ways:

- Through multiple rules—you specify OR logic using multiple rules when the OR logic requires two or more conditions that are different, as shown here:

Conditions		0	1	2
a	Cargo.volume		> 2000	-
b	Cargo.weight		-	> 100000
c				
d				
e				
f				
Actions				
Post Message(s)				
A	Cargo.container		'oversize'	'oversize'
B				

After you define the rules, you must save the file. When you save the file, Corticon Studio automatically detects the OR logic and adds hyphens in any empty cells that correspond to the defined conditions.

- Through OR'd value sets—if the OR logic needs to check for a list of values for a single attribute, you can enclose those values in curly braces in a rule column cell as shown here:

OR_Logic_ValueSets.ers				
Conditions		0	1	2
a	Aircraft.aircraftType		{'747', '777', '787'}	{'DC-10', 'MD-10'}
b				
c				
d				
e				
Actions				
Post Message(s)				
A	Aircraft.maxCargoWeight		100000	150000
B				

- You can also use comparison operators in an OR'd value set. For example if you specify {<200, >600}, any value that is less than 200 or greater than 600 will satisfy the condition.
- Note: You can specify an OR'd value set either by double-clicking a rule column cell and entering the value set or by pressing the CTRL key and selecting multiple values from the drop-down list (in this case, Corticon Studio automatically formats the OR'd value set).

Logical NOT

You can also use negating logic in Rulesheets. This is useful when you want to define a rule that fires when the attribute is 'NOT' a specified value. For example, consider a rule that assigns a maximum cargo weight to any aircraft that is not a 747. In this case, the condition must be defined in such a way that the rule checks for cases where the value of the aircraft type attribute is not 747.

There are several ways to accomplish this:

- Specify a Boolean condition expression with an equality operator (Aircraft.aircraftType='747') that returns False.

Negating_Logic.ers			
Conditions		0	1
a	Aircraft.aircraftType='747'		F
b			
c			
d			
e			
f			
g			
Actions			
Post Message(s)			
A	Aircraft.maxCargoWeight		100000
B			

- Specify a Boolean condition expression with an inequality operator (Aircraft.aircraftType < >'747') that returns True. The inequality operator (< >) is available for all data types.

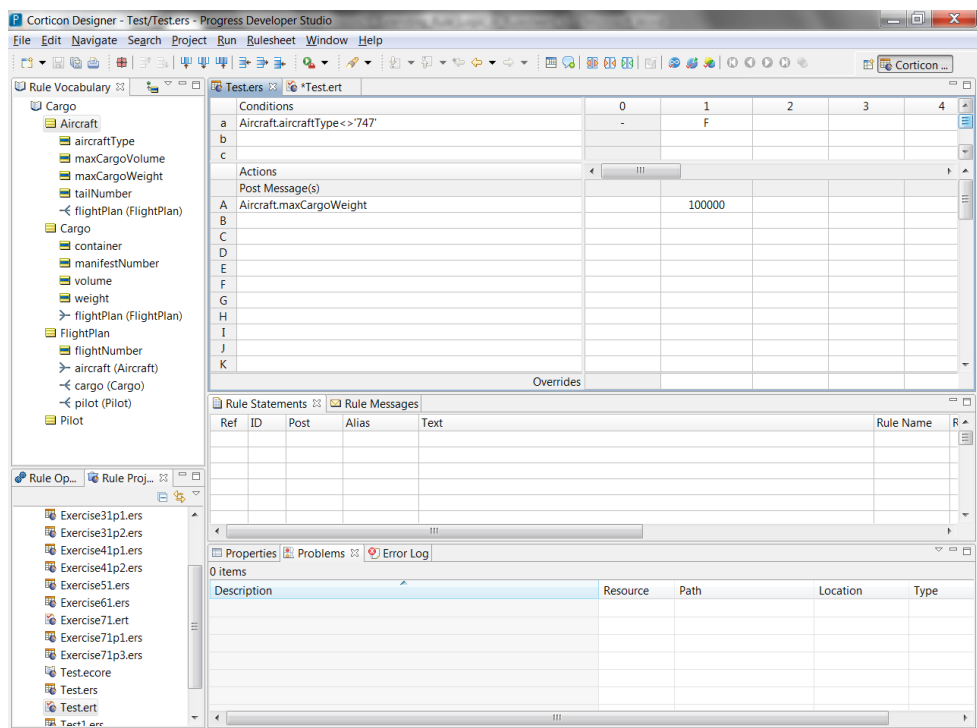
Negating_Logic.ers		
Conditions		
	0	1
a	Aircraft.aircraftType<>'747'	T
b		
c		
d		
e		
f		
g		
Actions		
Post Message(s)		
A	Aircraft.maxCargoWeight	100000
B		

- Specify a condition (Boolean or value-set based) that checks if the value is not something. To do this, you must use the not operator, which is a Boolean operator available in the Attribute Operators > Boolean subcategory in the Rule Operators view.

Negating_Logic.ers		
Conditions		
	0	1
a	Aircraft.aircraftType	not '747'
b		
c		
d		
e		
f		
g		
Actions		
Post Message(s)		
A	Aircraft.maxCargoWeight	100000
B		

In all of these examples, negating logic is expressed in different ways. You should choose whichever way is most intuitive to you and any stakeholders who may want to read and understand the rules in your Rulesheet.

Avoid multiple negatives



Note that you should use negating logic carefully. While a Corticon Rulesheet enables you to define double and triple negatives, defining multiple negatives is not a good practice. As in natural language, double and triple negatives are difficult for human readers to grasp, so any stakeholder who reads your rules may find them hard to understand.

For example, as shown in the image, you can create a rule in Corticon that checks if the aircraft type is a 747 using two negatives—`Aircraft.aircraftType <> '747'` and `F`. While this works the same way as `Aircraft.aircraftType = '747'`, it is harder to read.

How to check for null values

Conditions		0	1	
a	Aircraft.aircraftType		null	
b				

Conditions		0	1	
a	Aircraft.aircraftType = null		T	
b				

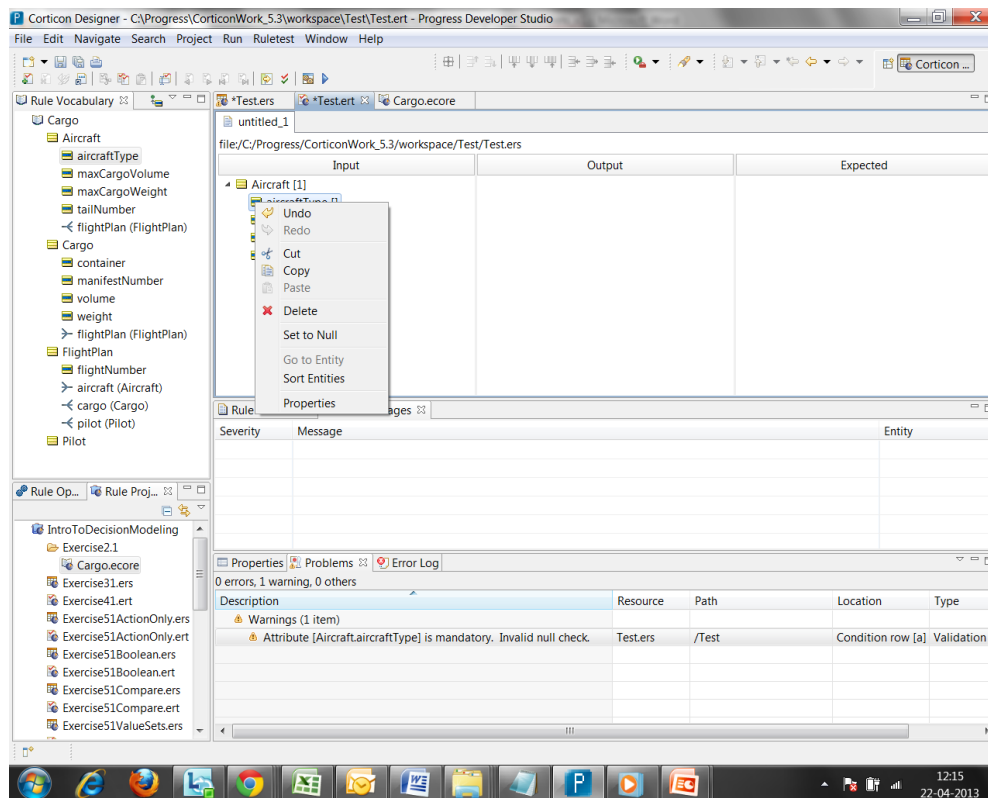
You can also specify a condition that checks for null in attribute values. Null means that no value has been provided for the attribute in the input data. Note that this is different from the value '0' which has significance in mathematical calculations, or an empty string that does not contain any characters, but is still treated as a value.

Checking for a null value is useful when you want to define a rule that assigns a default value if the value is null. Another common use is to send back information in the response rule message if the value received is null.

You check for a null value by specifying null without any quotes in the condition expression, as shown in this image.

When you create an attribute, you can specify whether the attribute value is mandatory or a null value is acceptable. If you attempt to define a condition that checks for null when the attribute value is mandatory, you will see a warning message.

Nulls in Ruletests



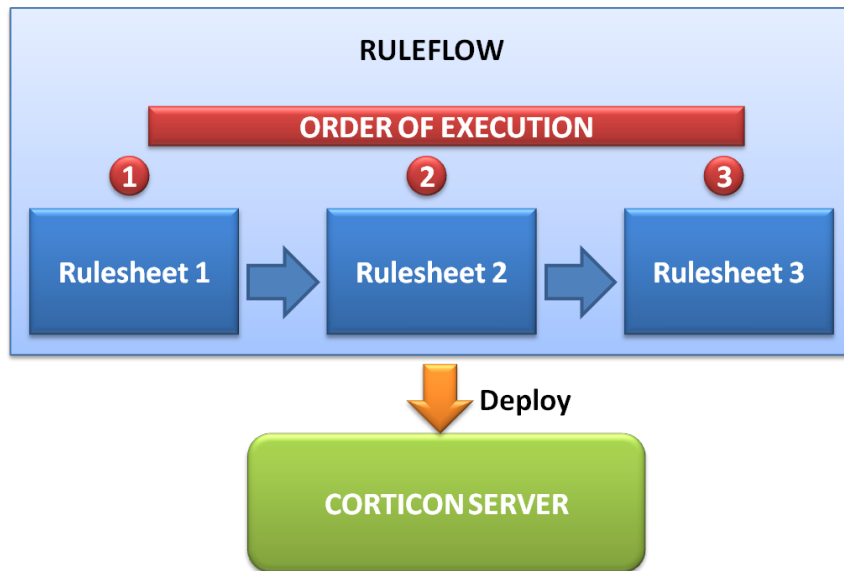
If you have to check for null in a rule condition, you must also understand how to specify null in Ruletests because you may want to test the rule using the Ruletest.

By default when you drag and drop an entity into the Input pane in the Ruletest editor, all the attributes in the entity have no value and are null. If you accidentally double-click a String attribute, the attribute value changes from null to an empty string (denoted by empty square brackets) even if you do not enter anything.

To ensure that the attribute value is null, right-click the attribute and select Set to Null.

What is a Ruleflow?

A Ruleflow enables you to connect two or more Rulesheets in a sequence.



When the Ruleflow is processed at runtime, the Rulesheets are executed one by one in that sequence. The output of one Rulesheet becomes the input of the next Rulesheet.

Note that a Rulesheet can be used in multiple Ruleflows. This enables you to use Rulesheets as reusable modules of rule logic. Any change in the rule logic only has to be made once in the Rulesheet and it is propagated across all Ruleflows that refer to it.

A Ruleflow is deployed to Corticon Server as a Decision Service. Rulesheets cannot be deployed directly. In cases where you need to deploy only one Rulesheet, you must add it to a Ruleflow and then deploy the Ruleflow.

For Corticon.js, a Ruleflow is packaged and deployed wherever the JavaScript application is running.

For details, see the following topics:

- [How to create Ruleflows](#)
- [Test Ruleflows using Ruletests](#)
- [Advanced Ruleflow tips and tricks](#)

How to create Ruleflows

Perform the following tasks to create Ruleflows:


- Create a Ruleflow file.
- Add Rulesheets to the Ruleflow using the Ruleflow editor.
- Connect Rulesheets in the Ruleflow editor.
- Set Ruleflow properties.

Create a Ruleflow file

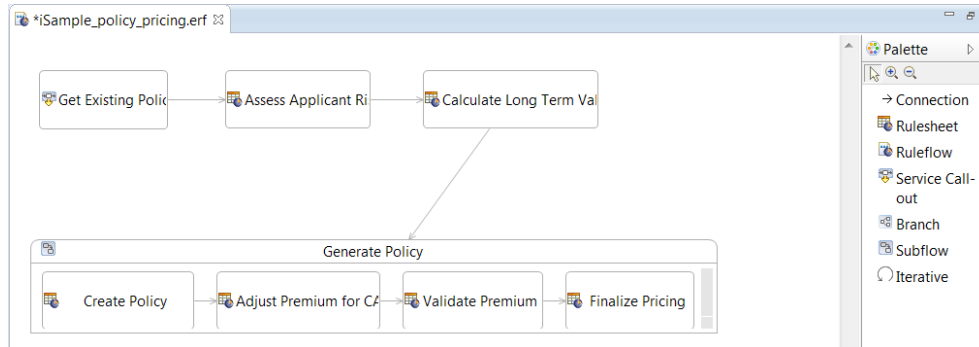
You must create a Ruleflow file under a rule project and associate it with a Vocabulary. All Rulesheets that are to be added to a Ruleflow must be associated with the Ruleflow's Vocabulary. A Ruleflow file has the extension .erf.

Follow these steps to create a Ruleflow file:

1. In Corticon Studio, click File > New > Ruleflow. The Create New Rule Flow wizard opens.
2. In the Create New Rule Flow wizard:
 - a. Ensure that the Enter or select parent folder field contains the path of the location where you want the Ruleflow created.
 - b. Enter a name for the Ruleflow in the File name field.
 - c. Click Next.
 - d. Ensure that the Vocabulary with which you want to associate the Ruleflow is selected.
 - e. Click Finish.

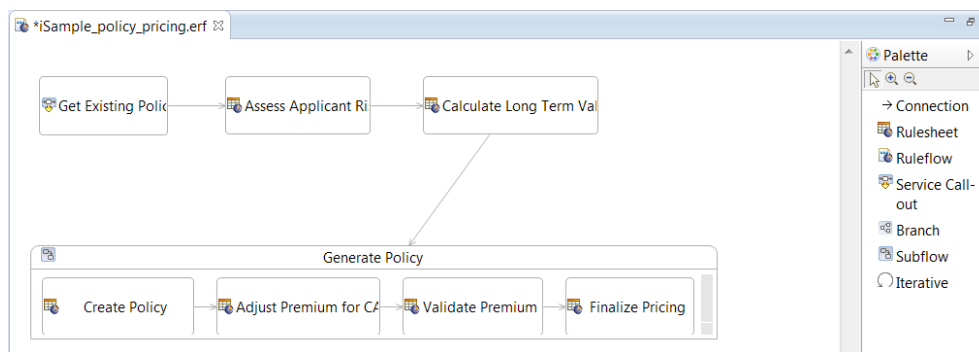
The newly created Ruleflow should now appear in the Project Explorer view. It is represented by the icon . The Ruleflow should also open by default in the Ruleflow editor.

Overview of the Ruleflow editor



The Ruleflow editor consists of two sections:

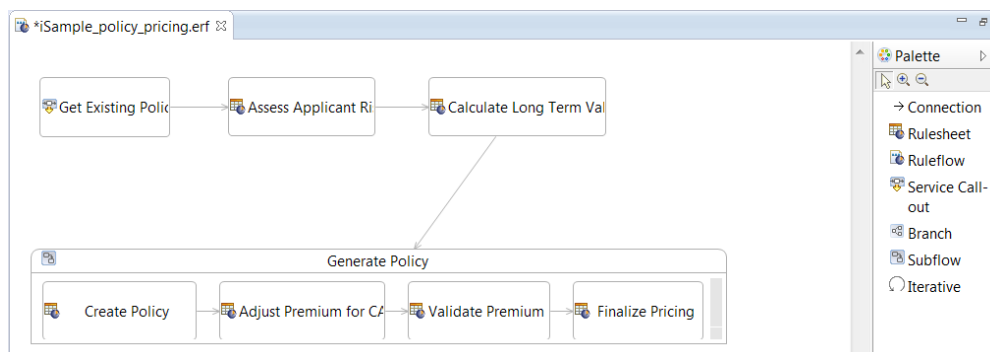
- A modeling area where you model a flowchart-like diagram of connected Rulesheets. The flowchart displays the sequence in which Rulesheets execute.
- A Palette that enables you to perform actions such as connect Rulesheets, create iterations, and create subflows.



Add Rulesheets to the Ruleflow

Before you start working with the Ruleflow editor, you must identify which Rulesheets you want to add, and the sequence in which the Rulesheets must be executed at runtime.

To add Rulesheets to the Ruleflow, you drag each Rulesheet from the Rule Project Explorer view to the modeling area in the Ruleflow editor. After you add a Rulesheet to the Ruleflow editor, it is represented by a box, as shown.

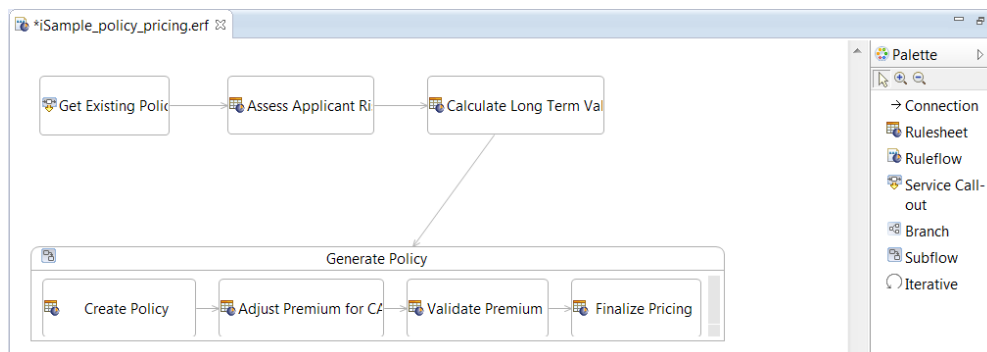


Since the Rulesheets are sequenced in a flowchart-like diagram, you can choose to arrange the Rulesheet boxes vertically, horizontally, or in a combination as shown here. However, it is advisable to drag and position the first Rulesheet in the sequence first, the second next, and so on to make it easier to connect them.

Note: Each Rulesheet box bears the Rulesheet's filename by default. However, you can change the name by triple-clicking the Rulesheet box and modifying it.

Connect Rulesheets in the Ruleflow editor

After you add all the Rulesheets that you require, you must connect them to define the order of execution.



You connect one Rulesheet to another by clicking Connection in the Palette and then dragging the connection arrow from one Rulesheet box to the next.

Set Ruleflow properties

When you open a Ruleflow file, the Properties view in Corticon is populated with different Ruleflow properties.

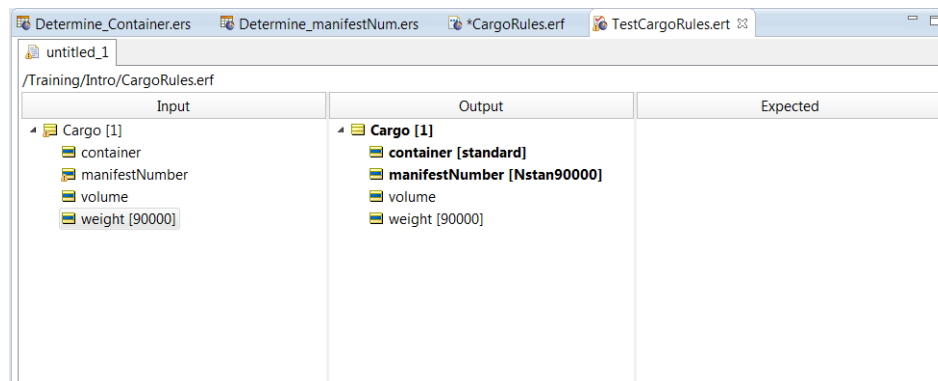
Ruleflow properties are categorized in three tabs:

Tab	Description
-----	-------------

Ruleflow Activity	<p>The properties in this tab are:</p> <ul style="list-style-type: none"> • Rule Vocabulary—the Vocabulary file associated with the Ruleflow. • Major Version, Minor Version, Version Label, Effective Date, Expiration Date—enables you to maintain and manage multiple versions of the Ruleflow. • Total Number of Rules—the number of rules included in the Ruleflow. This is a non-editable field; it is the sum of the number of rule columns and action-only rules in each Rulesheet that is added to the Ruleflow.
Comments	A text field that enables you to specify comments (for example, version-related comments) for the Ruleflow.
Rulers & Grid	Provides various options to display rulers and a grid in the modeling area of the Ruleflow editor.

Test Ruleflows using Ruletests

You can test a Ruleflow using a Ruletest just as you use it to test a Rulesheet. When you test a Ruleflow in a Ruletest, the input data that you define is processed by the first Rulesheet in the Ruleflow's sequence. After the first Rulesheet completes executing, its output becomes the input to the next Rulesheet in the sequence, and so on, until the last Rulesheet completes executing.




The Output pane then displays the final output of the Ruleflow. The Rule Messages view displays all rule messages that are created as a result of rules firing in each Rulesheet.

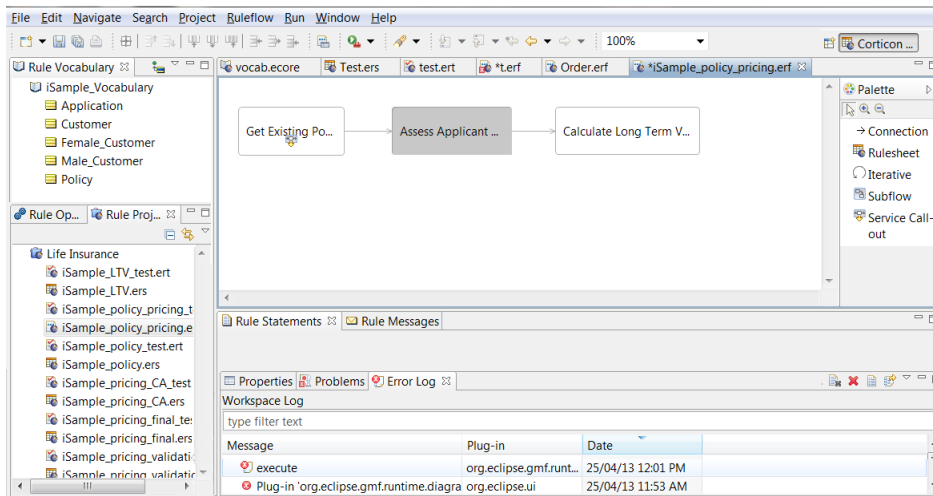
To test a Ruleflow using a Ruletest, you must select the Ruleflow as the Ruletest's test subject either while creating the Ruletest or by modifying the Test Subject property in the Ruletest editor.

Enable and disable Rulesheets in a Ruleflow

As you test Ruleflows, you may find it necessary to omit specific Rulesheets from the Ruleflow to see how the Ruleflow performs in different use cases.

When you omit a Rulesheet from the sequence, the order of execution skips a step—the output from the previous Rulesheet becomes the input to the next Rulesheet.

To omit a Rulesheet from the sequence of execution, select the Rulesheet box in the Ruleflow editor and click on the Disable icon  in the toolbar.



The disabled Rulesheet box turns grey, indicating that it has been disabled, as shown in this image.

To enable a Rulesheet box, click the same icon.

Advanced Ruleflow tips and tricks

Ruleflows provide a number of powerful features that are not in the scope of this course, including:

- Nested Ruleflows—you can reduce the complexity of large Ruleflows by breaking them into smaller, 'child' Ruleflows and adding them to the parent Ruleflow, referred to as "a Ruleflow in a Ruleflow."
- Conditional Branching—you can create branches in a Ruleflow, where the value of an attribute determines which branch the data is routed to.
- Subflows—you can configure an Iteration for a Subflow to enable looping behavior.
- Versioning—you can assign Major and Minor Version number that the Server responds correctly to requests for the different versions.
- Effective Dates—you can have identically named Ruleflows with slight variations that respond to requests only when in the specified date range.
- Service Call-outs—you can access Datasources to enrich your rules, and update databases.

These topics are discussed in the course Advanced Rule Modeling in Progress Corticon Studio.