



Corticon Tutorial

Deploying a Progress Corticon Decision Service inProcess for Java

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.

Last updated with new content: Corticon 6.2

Updated: 2021/06/04

Table of Contents

Tutorial - Deploying a Progress Corticon Decision Service in Process for Java.....	7
Install Corticon components.....	9
Set up the tutorial.....	11
Create Java classes for the Vocabulary entities.....	15
Import the JAR file into the Vocabulary.....	19
Verify that the Java classes are imported and mapped.....	23
Add the JOM JAR to the package.....	25
Package the Project	27
Create the In-Process client app project.....	31
Add Libraries to the project and the Java Build Path.....	33
Write Java Client code to deploy and access the Decision Service.....	37
Test the Decision Service.....	41

Tutorial - Deploying a Progress Corticon Decision Service in Process for Java

Rule modeling lets you create, analyze, and test your work in a Studio environment, and then deploy them to a Progress Corticon Server as a Decision Service.

There are several ways you can deploy a Decision Service. You can deploy a Decision Service in-process or as a Web Service for either Java or .NET. In this tutorial, you will learn how to deploy and access a Decision Service in-process from a Java client.

Corticon Server is a set of core Java classes that comprises all the functionality required to host and manage Decision Services. These Java classes can be deployed into an application container together with a Java application to enable the application to standalone as an in-process server.

To deploy and access a Decision Service from a Java client application, you need to perform the following tasks:

1. Create Java classes for the Vocabulary entities
2. Import the Java files into the Corticon Studio
3. Package the Ruleflow in Corticon Studio
4. Write Java client code to deploy and access the Decision Service

In this tutorial, you will learn how to perform these tasks.

This tutorial is designed for hands-on use. We recommend that you follow along in your Java editor, and Corticon Studio, using the instructions and illustrations that are provided.

About Java

You will use Corticon Studio for Java programming. In the scope of this tutorial, and run the in-process server project in the Eclipse environment. Eclipse provides solid Java development and runtime functionality, and you don't need any additional Java features.

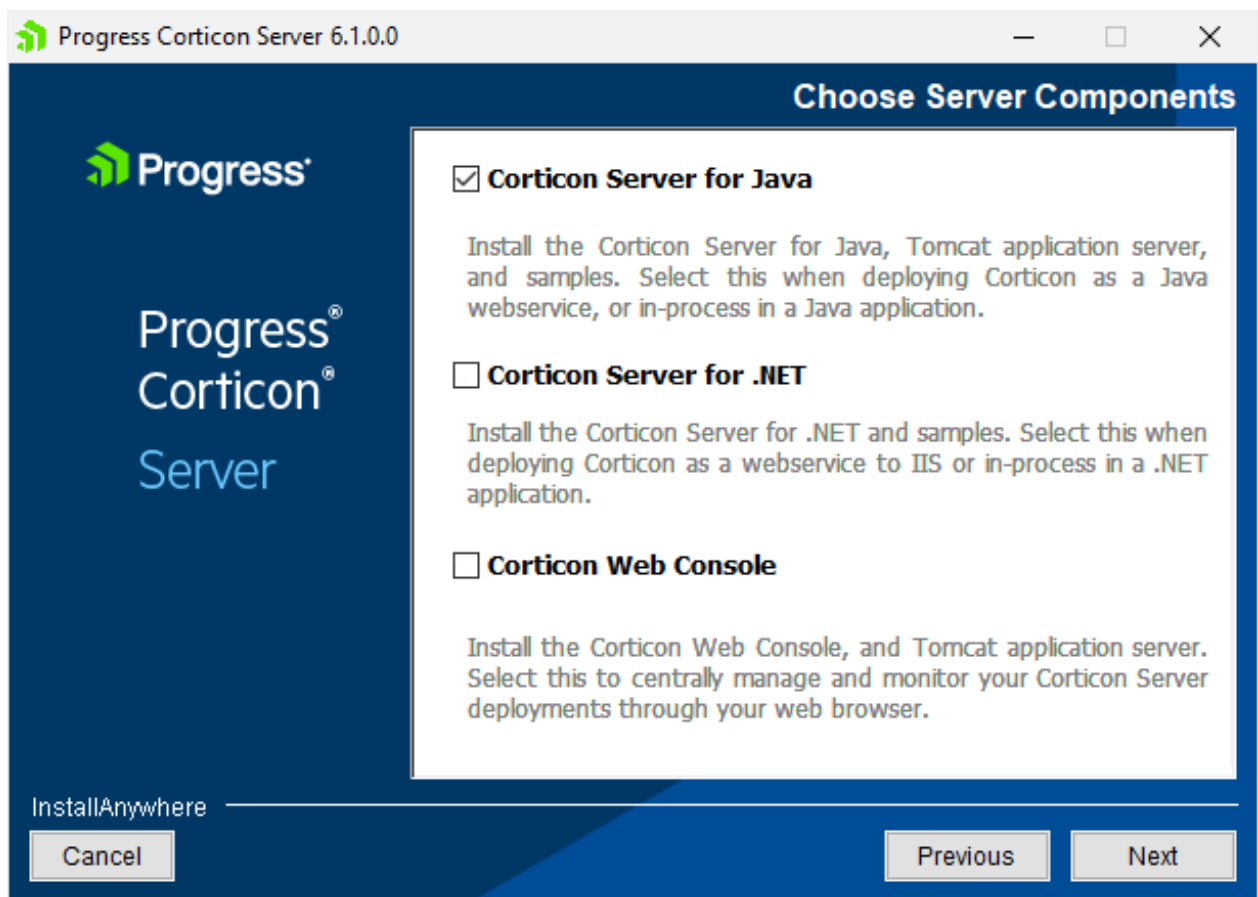
Install Corticon components

In this tutorial we will deploy the “Cargo” sample in process using Corticon Studio and the Java developer environment that is included in the Eclipse environment that Corticon Studio installed.

The only purpose of the Corticon Server for Java in this in-process example is to provide the server libraries that are used in the in-process server.

To install Corticon components:

1. Download Corticon Studio, then run it, accepting all defaults.
2. Download Corticon Server, then run it.
3. Choose just the component **Corticon Server for Java**:



Set up the tutorial

1. Choose the **Start** menu command **Progress > Corticon Studio**.
2. Choose **Help > Samples**.
3. On the **Samples** page, select **Tutorial** and click **Open**.

workspace - Corticon Studio

File Edit Navigate Search Project Run Window Help

Welcome

Samples

This page contains references to samples of the Progress products that you installed on this instance of Eclipse. You can filter by product, topic, and level of complexity.

[View Eclipse and Other Samples](#)

All Samples | Group by: Complexity

Training - This sample provides a structured dictionary for a few business processes. It contains all the necessary business terms and maps the relationships between the terms using the business rules.

Tutorial - This sample describes the business requirement of an airfreight company.

Intermediate

EDC Database Connectivity - This sample demonstrates the use of Corticon's Enterprise Database Connectivity (EDC) for accessing a database from rules.

GrandParent - This sample has been developed to simulate a parent-child relationship or a list of descendants.

Tutorial

This sample has been developed from the business requirement of an airfreight company. The company loads cargo of various sizes and weights onto its aircraft. For air safety, the company must ensure that the aircraft is not loaded with oversized and overweight cargo, that will exceed the weight limit. A part of the flight plan involves verifying that no safety violations are present. The company wants to improve the quality and efficiency of its operations by modeling and automating its business rules.

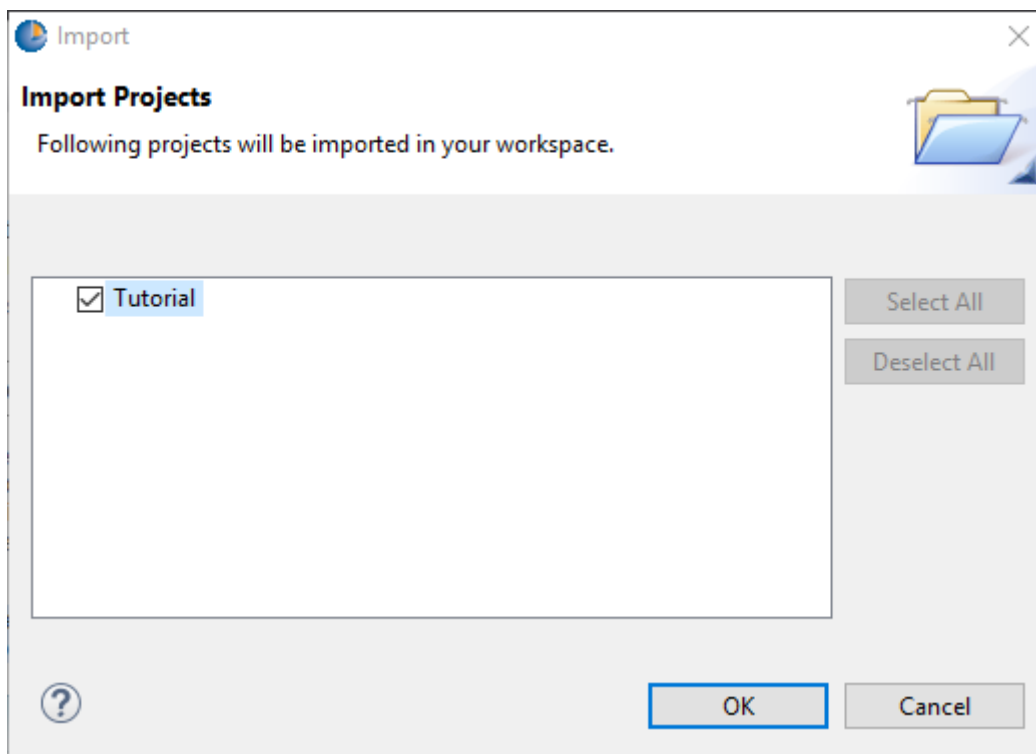
Open

Reference ID : Not available

Features : Progress Corticon Studio

Progress

4. In the **Import Projects** window, click **OK**.



The sample rule project opens in Corticon Studio.

Create Java classes for the Vocabulary entities

The data payloads of the request and response messages that are sent or received by the Java client are in the form of a collection of Java objects.

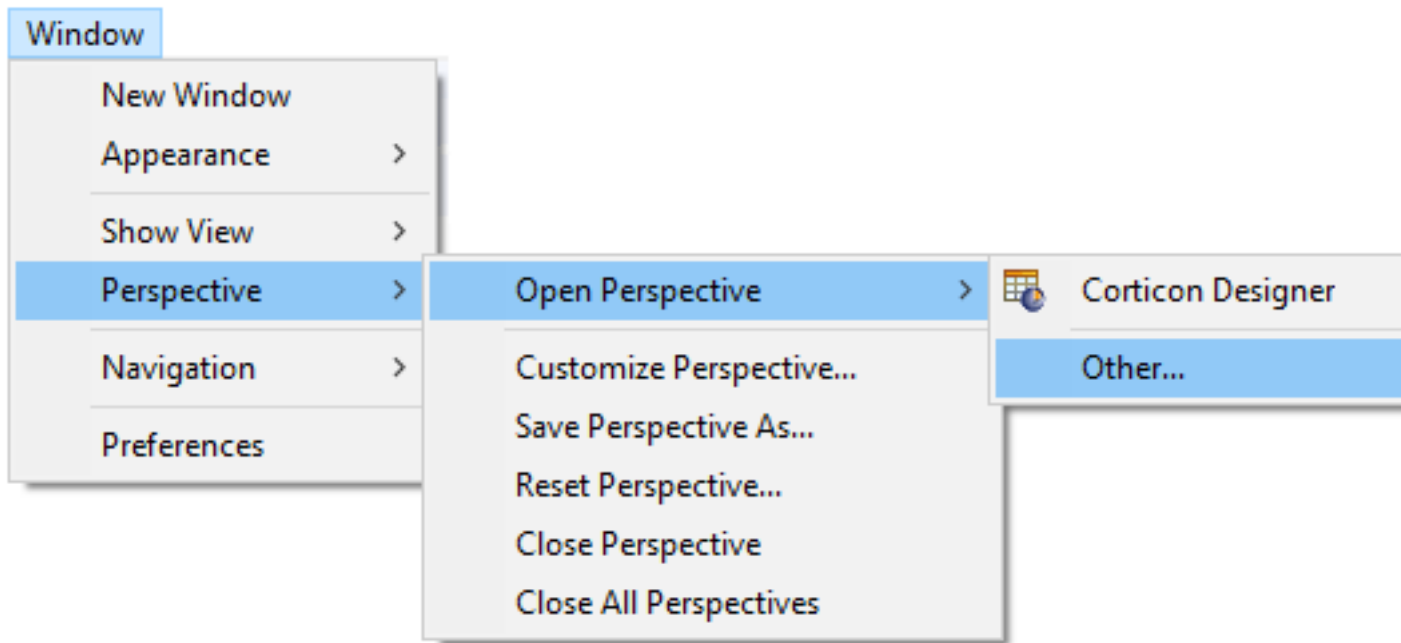
You need to create a class for each Vocabulary entity that is used by the rules you want to call. We will create get and set methods within each class file. Then we will export the code to a JAR file, and place it in a location where it can be accessed by Corticon Server at run-time.

Switch to the Java Perspective in Corticon Studio

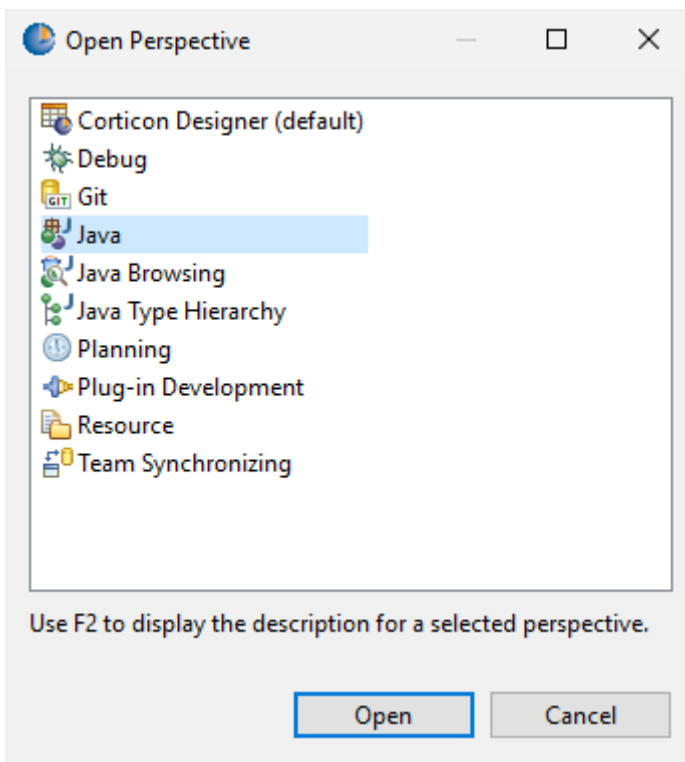
1. Click on the Java option in the top right of your Corticon Studio screen.



If you have not previously opened this perspective, choose:



2. And then choose Java:



Create Java 'get' and 'set' methods in a library

You need to create a class for each Vocabulary entity that will be used by the rules you want to call. In this tutorial, we will create a class for just one entity – Cargo.

1. Click **File > New > Java Project**.
2. Name the project **cargoLibrary**.
3. Accept all the default values, and click **Finish**.

4. In the Package Explorer, expand **cargoLibrary**, right-click on **src**, and then choose **New > Package**. Name the package **cargoLibrary**.
5. Right-click on the **cargoLibrary** package, and then choose **New > Class**. Name the package **Cargo**. (The class name and the entity name must match exactly, and they are case-sensitive.) The java editor opens, ready for your text.
6. Declare the entity's attributes as variables with the appropriate data type in get and set methods. (The variable and the attribute names must match exactly, and they are case-sensitive, as shown:

```
package cargoLibrary;
public class Cargo {
    public String container;
    public String manifestNumber;
    public Boolean needsRefrigeration;
    public long volume;
    public long weight;

    public String getContainer() {
        return container;
    }

    public void setContainer(String container) {
        this.container = container;
    }

    public String getManifestNumber() {
        return manifestNumber;
    }

    public void setManifestNumber(String manifestNumber) {
        this.manifestNumber = manifestNumber;
    }

    public Boolean getNeedsRefrigeration() {
        return needsRefrigeration;
    }

    public void setNeedsRefrigeration(Boolean needsRefrigeration) {
        this.needsRefrigeration = needsRefrigeration;
    }

    public long getVolume() {
        return volume;
    }

    public void setVolume(long volume) {
        this.volume = volume;
    }

    public long getWeight() {
        return weight;
    }

    public void setWeight(long weight) {
        this.weight = weight;
    }
}
```

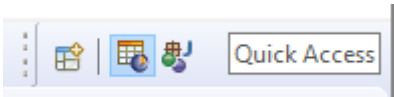
7. Save the file.
8. Select the project **cargoLibrary**.
9. Choose **File > Export**, and then **Java > JAR file**.
10. Select **cargoLibrary** and its related files

11. Name the JAR file **Cargo**.
12. Save the JAR file at a temporary location. It is a good idea to save it in the project.

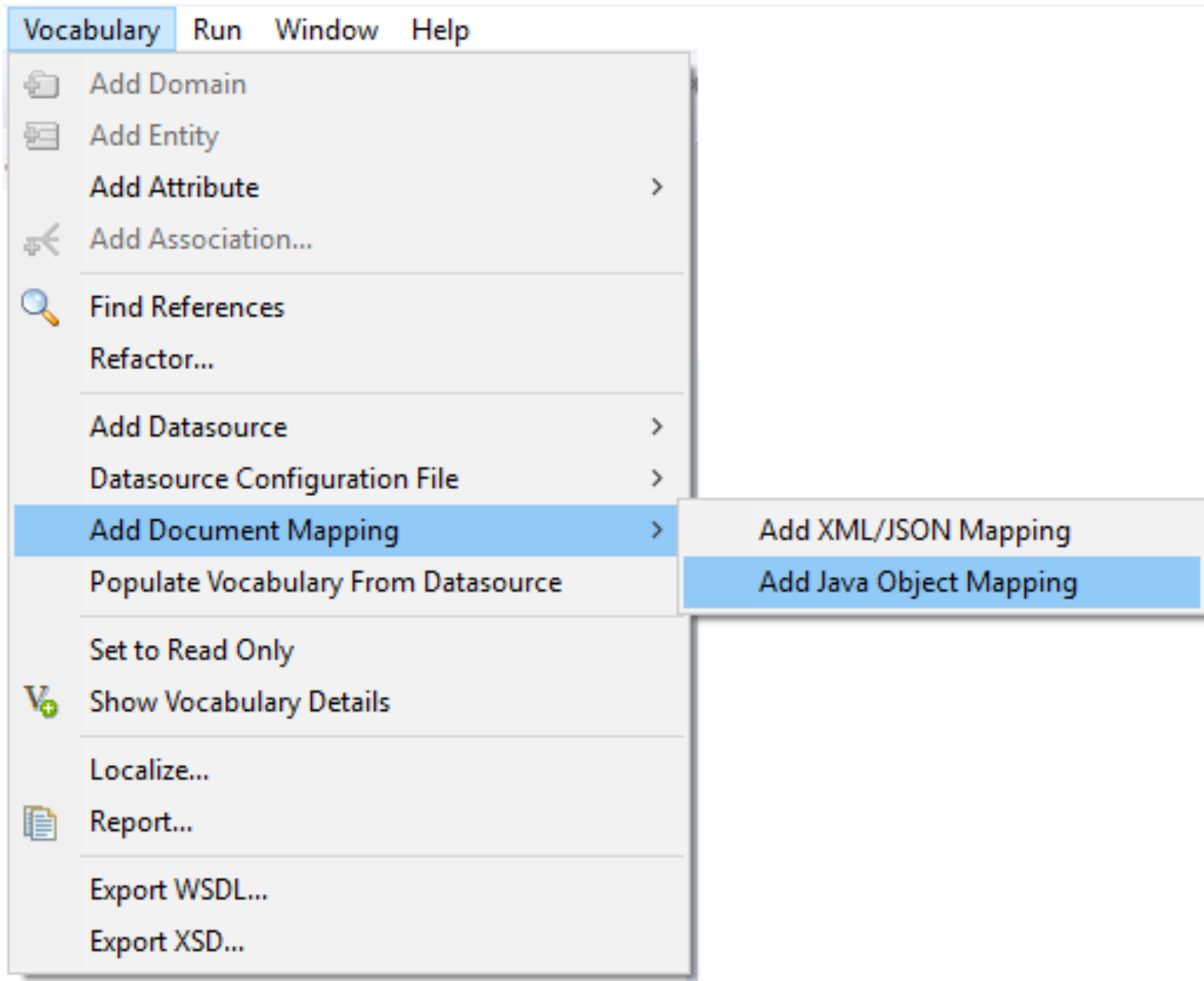
Import the JAR file into the Vocabulary

The next step is to map the Java classes and variables you just created to the Vocabulary entities in Corticon Studio. You need to import the JAR file that you just created.

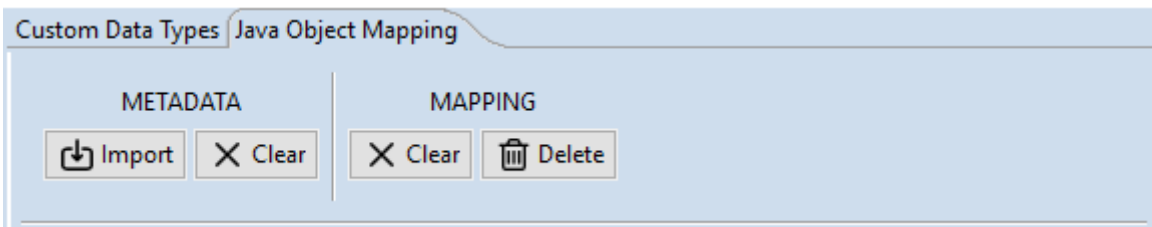
1. Switch perspective by clicking the **Corticon Designer** button at the top right of the screen:



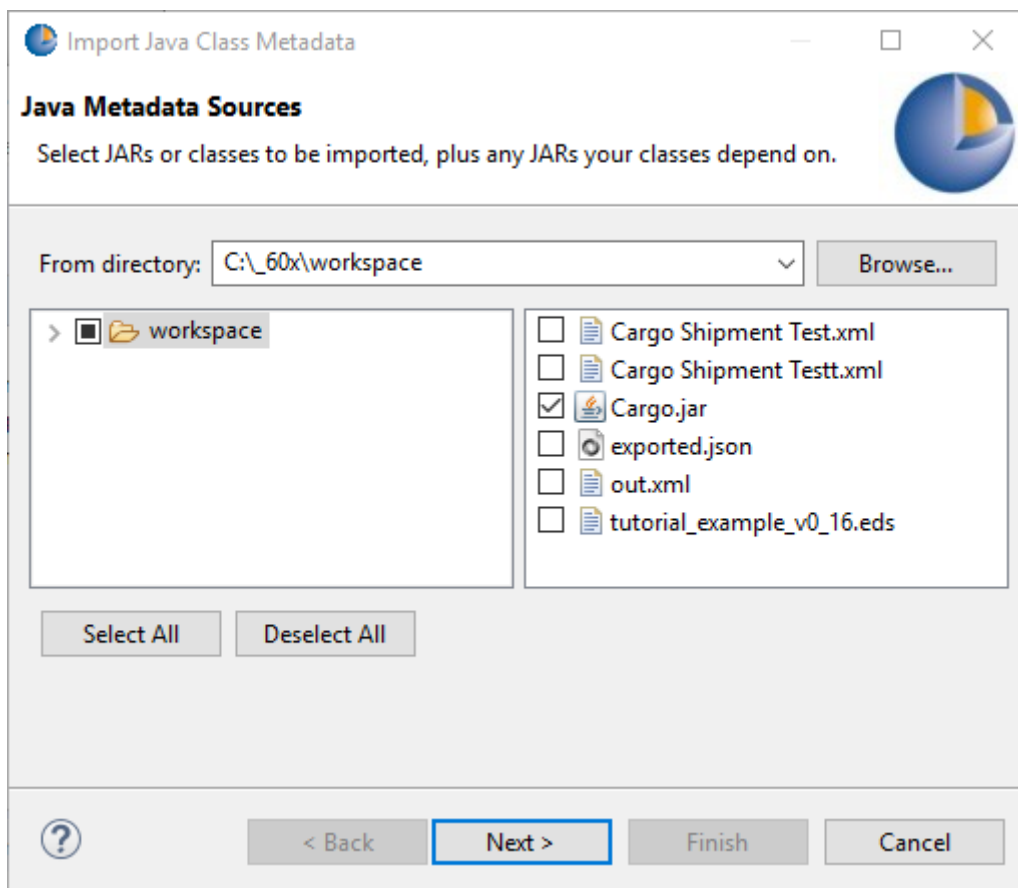
2. In the Project Explorer, expand **Tutorial > Tutorial-Done**, and then open Cargo.ecore.
3. Enable Java Object Mapping by choosing:



4. On the **Java Object Mapping** tab added to the Vocabulary root, click **METADATA Import**:

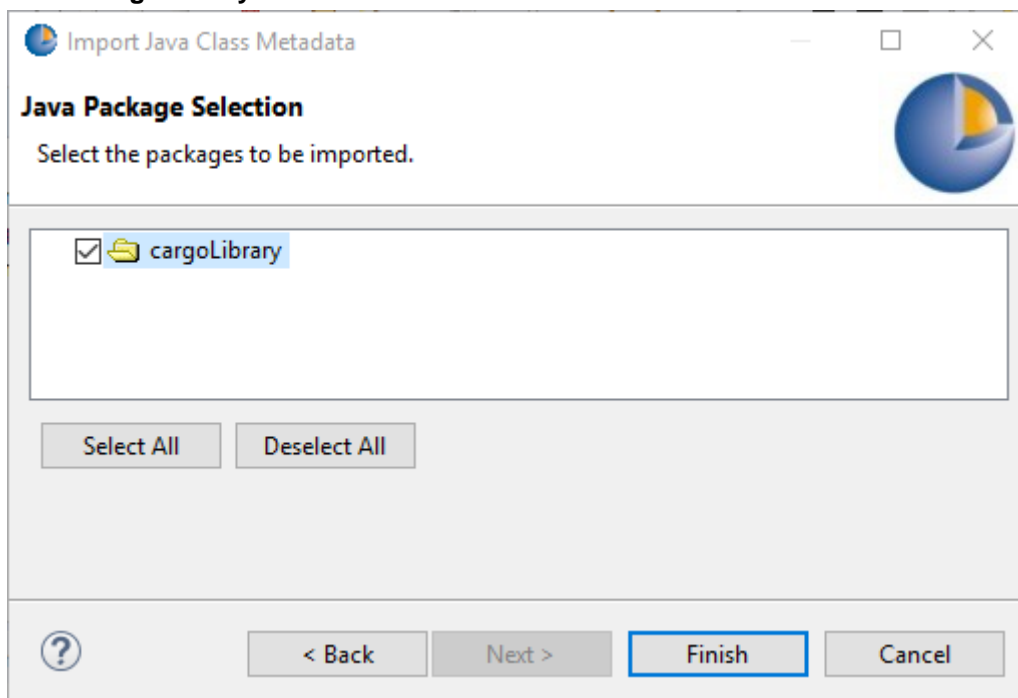


5. In the Import **Java Class Metadata** window, select Cargo.jar:



6. Click **Next**.

7. Select **cargoLibrary**:



8. Click **Finish**.

9. Save the Vocabulary file.

Verify that the Java classes are imported and mapped

Once the Java classes and variables have been imported and mapped into the Vocabulary, you can see them in the Vocabulary. Click on the Cargo entity. You can see the Java properties—Java Package and Java Class Name. The Java Package name matches the package we defined earlier. The Java Class Name is the same as the entity name.

The screenshot shows the Vocabulary tool interface. On the left, a tree view displays the hierarchy: Cargo (expanded) contains Aircraft, Cargo (selected), container, manifestNumber, needsRefrigeration, volume, weight, flightPlan (FlightPlan), and FlightPlan. On the right, the 'Basic Properties' and 'Java Object Properties' tabs are visible. The 'Basic Properties' tab shows the Entity Name as 'Cargo' and Inherits From as empty. The 'Java Object Properties' tab shows the Java Package as 'cargoLibrary' and the Java Class Name as 'Cargo'.

Basic Properties	
Property Name	Property Value
Entity Name	Cargo
Inherits From	

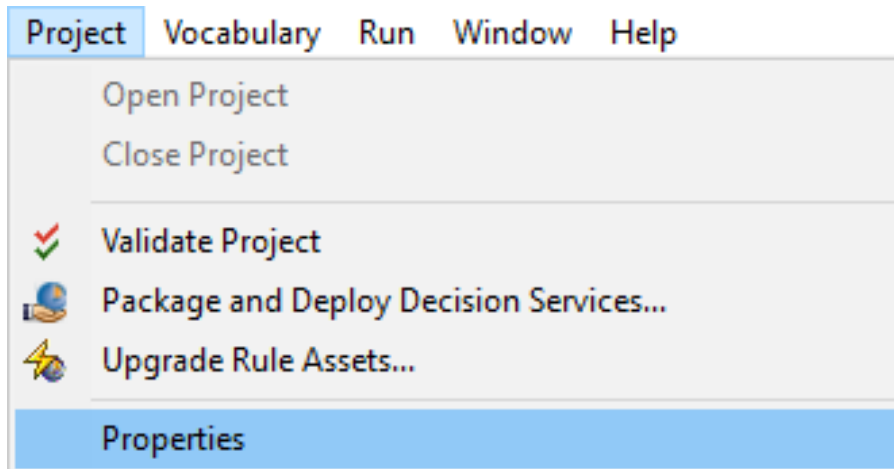
Java Object Properties	
Property Name	Property Value
Java Package	cargoLibrary
Java Class Name	Cargo

Click on the manifestNumber attribute. Note that each attribute has the following Java properties—Java Object Get Method, Java Object Set Method, and Java Object Field Name. The get and set methods were populated by importing the Java class metadata.

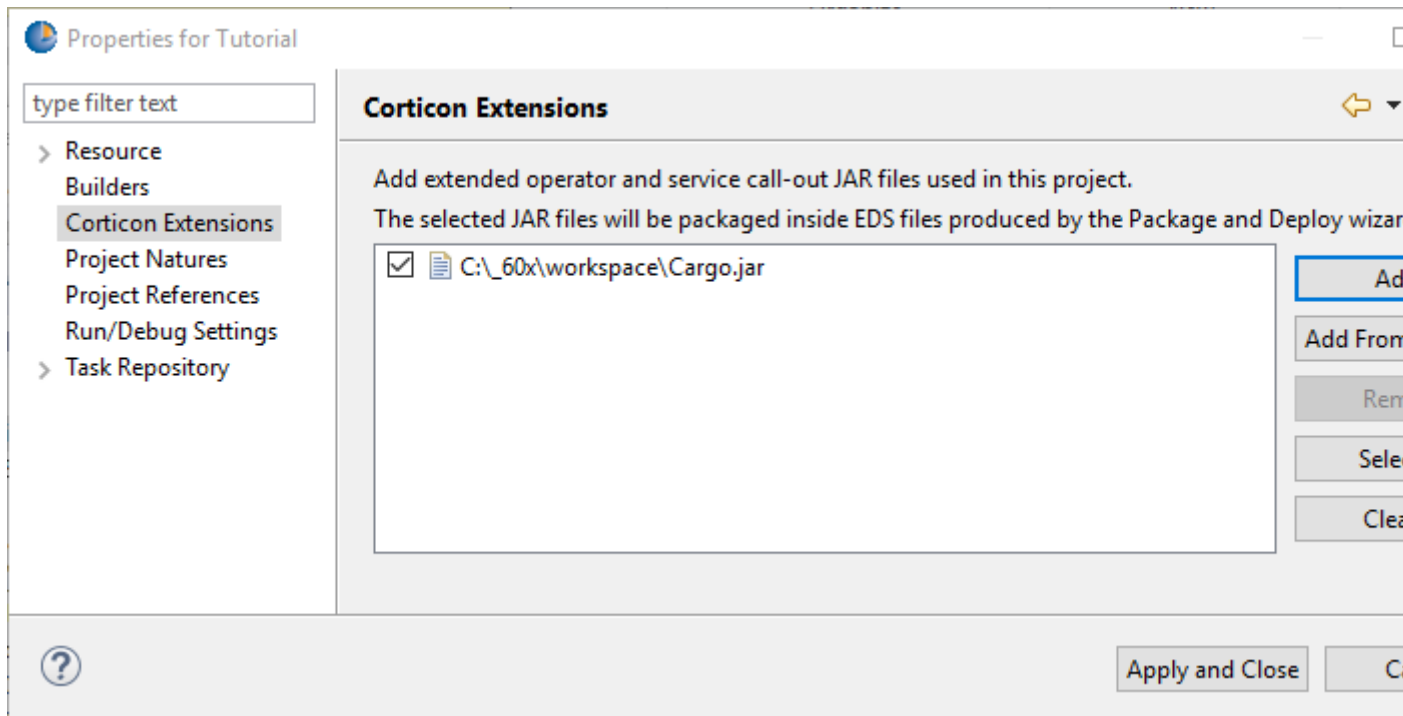
Add the JOM JAR to the package

Before we package the project, we need to add the Java Object class library to the project.

1. Right click on **Tutorial**, and then choose **Project > Properties**.



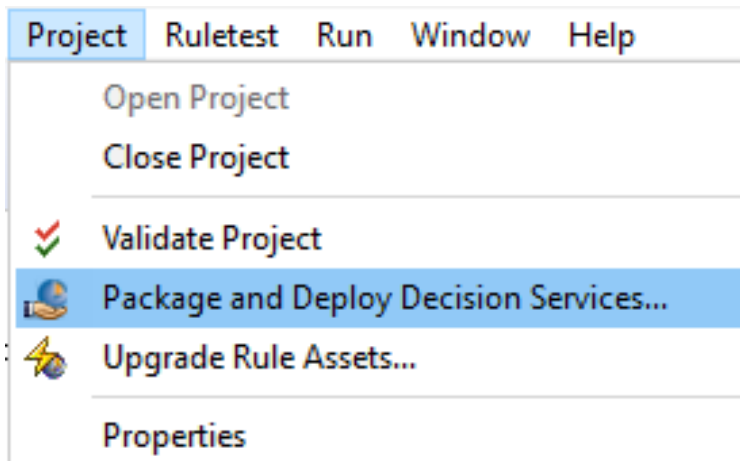
2. In the Properties dialog box, click on **Corticon Extensions**, and then click **Add** to locate the Cargo.jar.



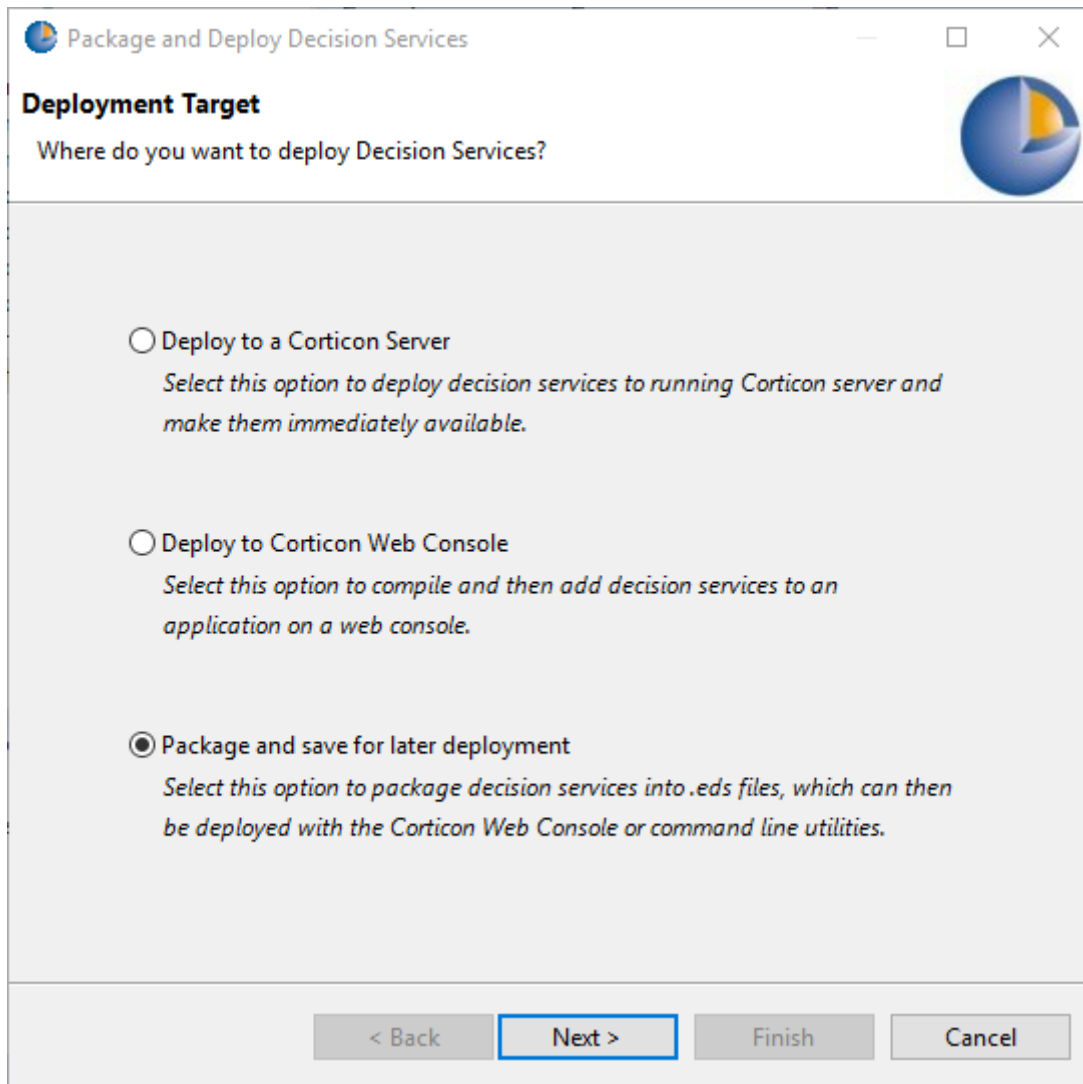
Package the Project

Now we can compile the project into a Decision Service, ready for deployment.

1. Click on **Tutorial**, and then select **Project > Package and Deploy Decision Services**.



2. In the Deployment Target panel, choose **Package and save for later deployment**.



3. Click **Next**.
4. In the Package Ruleflows panel, select the **tutorial_example** Ruleflow.

Package and Deploy Decision Services

Package Ruleflows

Select the ruleflows to be packaged for later deployment

Select All Deselect All

	Decision Service Name	Version	Database Mode	Ruleflow
<input checked="" type="checkbox"/>	tutorial_example	0.16	None	/Tutorial/Tutorial-Done/tutorial_example.erf

To directory:

< Back Next > Finish Cancel

5. Edit the Decision Service Name, and change it to **Cargo**
6. Click **Finish**.

Create the In-Process client app project

The client is a separate project.

1. Switch to Corticon Studio Java perspective.
2. Choose **File > New > Java Project**.
3. Name the project **App**, and then click **Finish**.

Add Libraries to the project and the Java Build Path

The JARs in the Corticon Server installation will enable the in-process server's rule processing engine. While we could reference the required JARs, adding them into the project will make it clear that the in-process server is standing on its own. Actually, once the Server JARs have been added to the client project, the Server could be uninstalled:

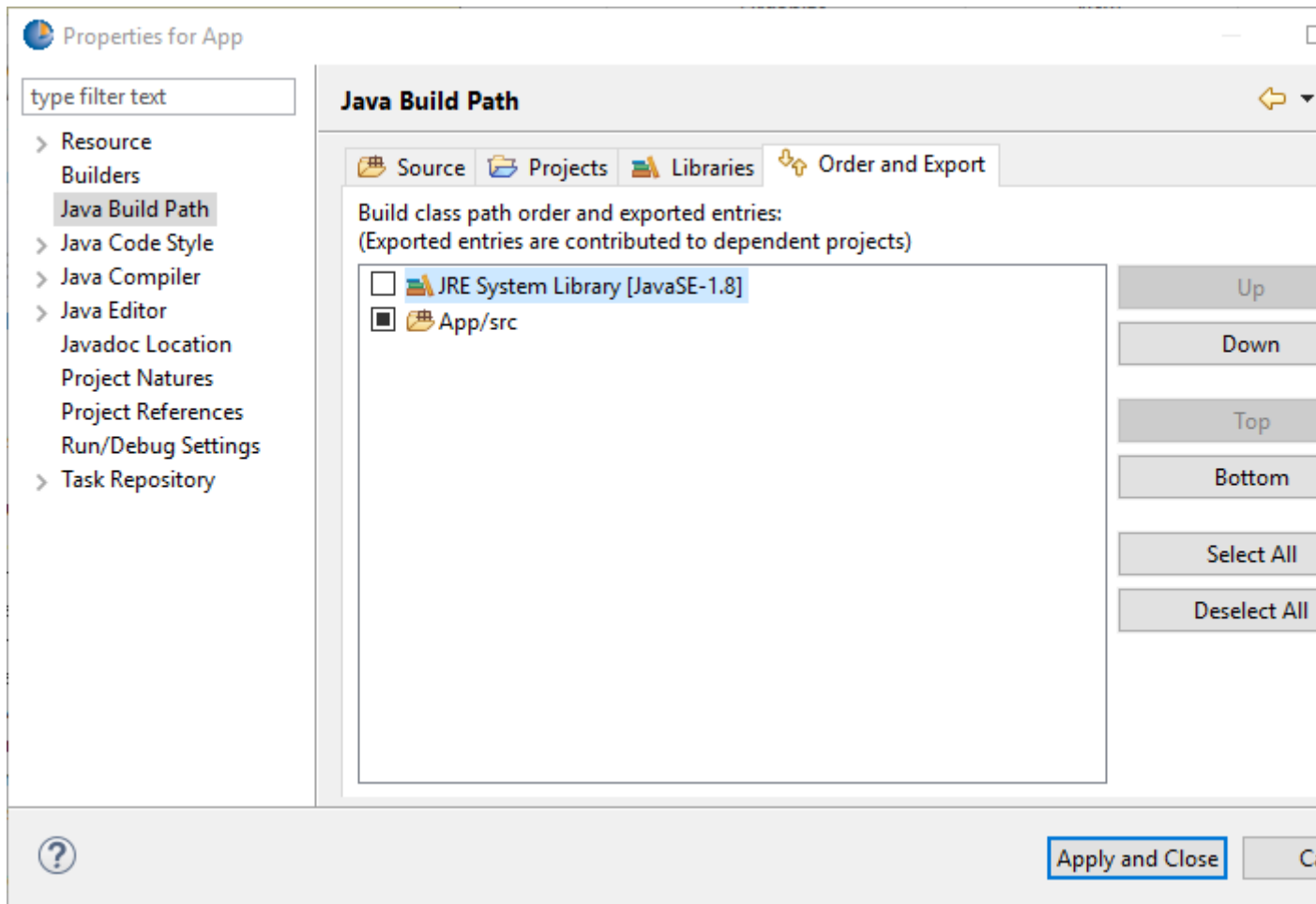
1. Copy the following JAR files from Corticon Server's `[CORTICON_HOME]/Server/lib`:

- `ant_launcher.jar`
- `CcConfig.jar`
- `CcExtensions.jar`
- `CcI18nBundles.jar`
- `CcLicense.jar`
- `CcServer.jar`
- `CcThirdPartyJars.jar`

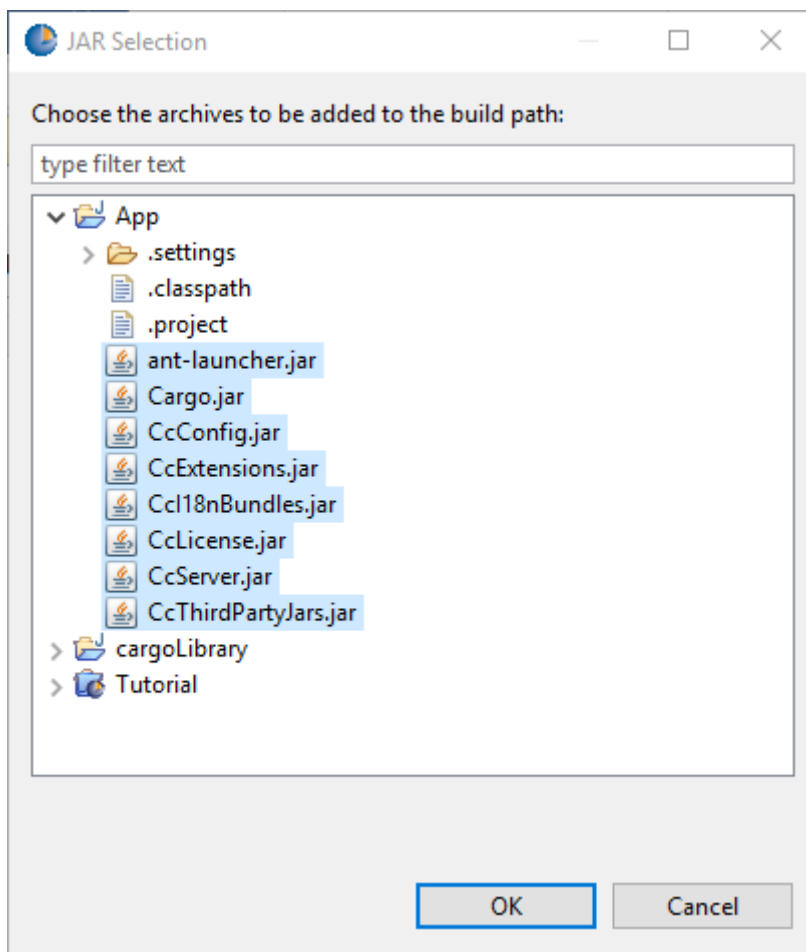
2. Paste them into the root of App Project.

3. Copy **Cargo.jar**, and then paste into the root of the App Project.

4. Right-click on **App**, choose **Properties**, and then click **Java Build Path**.



5. Select the **Libraries** tab, and then click **Add JARs**.
6. Add all the JARs in the App to the Java classpath:



7. Click **OK**, and then click **Apply** and **Close**.

Write Java Client code to deploy and access the Decision Service

To deploy and access a Decision Service from a Java client, you write Java code that uses Corticon APIs.

1. In the **App** project, choose **New > Package**. Name the package program.
2. Click on the program package, and then choose **New > Class**. Name the class **InvokeDS**.
3. The editor opens for **InvokeDS.java**.
4. Let's begin writing the Java code:

In your client program, begin by importing the required packages and classes as shown.

```
package program;

import com.corticon.eclipse.server.core.CcServerFactory;
import com.corticon.eclipse.server.core.ICcServer;
import com.corticon.service.ccserver.*;
import cargoLibrary.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;
import java.util.Properties;
```

5. Next, create a String variable to hold the Decision Service name. Note that the Decision Service name you provide overrides the name you may have specified earlier, while compiling the Ruleflow.

```
public class InvokeDS {

    private static String decisionServiceName = "Cargo";
```

```
public void callDS(){
}
}
```

6. Now, write code to create an object of the entity class. In this step, you create an object of the entity class and populate it with data that you want to test against the Decision Service. In our example, we will set an input volume of 10 and weight of 1000. You can try various other input combinations, based on the attributes defined in the rules.

```
try {
    Cargo cargo = new Cargo();
    cargo.setWeight(1000);
    cargo.setVolume(10);
}
```

7. Write code to create an arraylist and add the object to the arraylist. The method to send a request message requires a collection as one of the parameters. So, you must create a collection, such as an arraylist, and add the entity object to it.
8. In this example, the weight and volume values for Cargo, defined in the Cargo request object, will be processed and the Cargo's container variable will be determined by the Decision Service.

```
List<Cargo> cargoList = new ArrayList<>();
cargoList.add(cargo);
```

9. Write code to initialize Corticon Server. To do this, you use the ICcServer interface. ICcServer is implemented by the CcServerFactory class. To instantiate Corticon Server you use a method in CcServerFactory named getCcServer() to return an object of the type ICcServer as shown in the example. In the example, the object is assigned to the variable ccServer.

```
ICcServer server = CcServerFactory.getCcServer();
```

10. Once you have obtained an object of the type ICcServer, you use it to call the addDecisionService() method to deploy the Ruleflow. In our example, we will deploy the Cargo.eds file that we packaged earlier in Corticon Studio. The addDecisionService() method is an overloaded method. However, at minimum, you must pass three parameters—the Decision Service name, the path to the compiled EDS file, and a Boolean parameter that specifies whether dynamic reload should be set to true or false. Before you deploy the decision service, check if the Decision Service is already loaded using an 'if' statement.

```
Properties p = new Properties();
p.put(ICcServer.PROPERTY_AUTO_RELOAD, false);
if (!server.isDecisionServiceDeployed("Cargo")){
    server.addDecisionService(decisionServiceName, "Cargo_v0_16.eds", p);
}
```

11. Write code to execute the Decision Service. The execute() method accepts two parameters—the Decision Service name, and the collection that contains the request object. This sends the request object message to Corticon Server. This message is then processed by the Decision Service that is deployed on the server.

The execute method then updates the Cargo request object in memory and returns rule messages. You can access the rule messages by creating an object of the type ICcRuleMessages.

```
ICcRuleMessages msgs = server.execute("Cargo", cargoList);
```

12. Next, create a list that will receive the rule messages from the ICcRuleMessages object.

```
List<ICcRuleMessage> msgList = msgs.getMessage();
```

13. Write code to iterate through each rule message.

```
Iterator<ICcRuleMessage> itr = msgList.iterator();
while(itr.hasNext()) {
    ICcRuleMessage m = itr.next();
}
```

14. Now write code to get the rule message's associated object, and convert the generic object to a Cargo object.

```
Object obj = m.getEntityReference();
Cargo x = (Cargo) obj;
```

15. Write print statements to display the output in the console.

```
System.out.println("-----");
System.out.println("Cargo weight is "+x.getWeight());
System.out.println("Cargo volume is "+x.getVolume());
System.out.println("Cargo container value is "+x.getContainer());
System.out.println(m.getSeverity());
System.out.println(m.getText());
```

16. Add a catch block for the try block we have used in the code. And finally, write the Java main method.

```
} catch(Exception e){
    System.out.println(e);
}

public static void main(String[] args) {
    InvokeDS d = new InvokeDS();
    d.callDS();
}

}
```

You have now finished writing the Java client code. Save the Java code under an appropriate package and project name. Here is the complete code used to deploy and access the Decision Service.

```
package program;
import com.corticon.eclipse.server.core.CcServerFactory;
import com.corticon.eclipse.server.core.ICcServer;
import com.corticon.service.ccserver.*;
import cargoLibrary.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;
import java.util.Properties;

public class InvokeDS {

    private static String decisionServiceName = "Cargo";

    public void callDS(){
        try {
            Cargo cargo = new Cargo();
            cargo.setWeight(1000);
            cargo.setVolume(10);
            List<Cargo> cargoList = new ArrayList<>();
            cargoList.add(cargo);

            ICcServer server = CcServerFactory.getCcServer();
            Properties p = new Properties();
            p.put(ICcServer.PROPERTY_AUTO_RELOAD, false);
            if (!server.isDecisionServiceDeployed("Cargo")){
                server.addDecisionService(decisionServiceName,"Cargo_v0_16.eds", p); }
            ICcRuleMessages msgs = server.execute("Cargo", cargoList);
            List<ICcRuleMessage> msgList = msgs.getMessages();
            Iterator<ICcRuleMessage>
            itr = msgList.iterator();

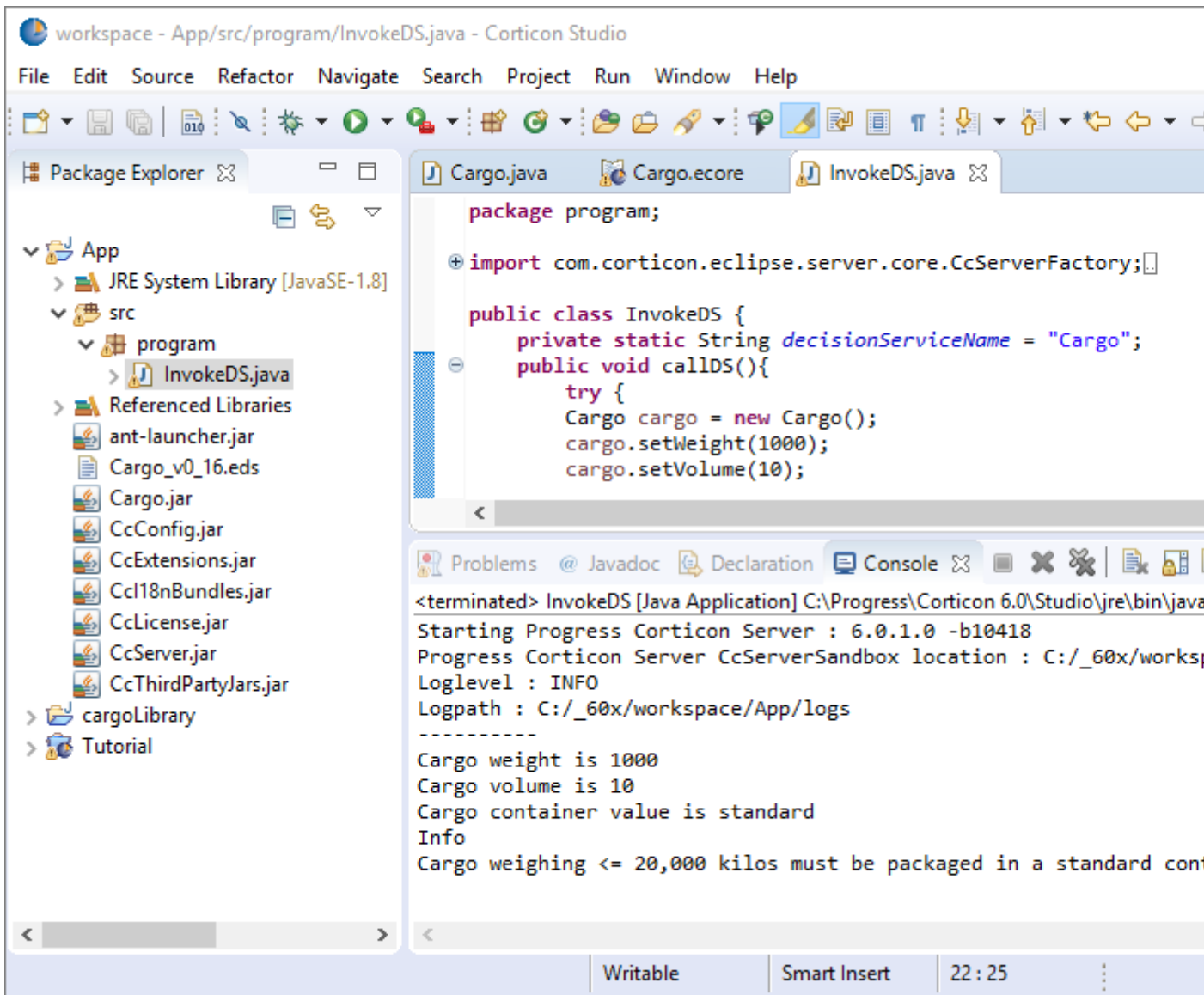
            while(itr.hasNext()) {
                ICcRuleMessage m = itr.next();
                Object obj = m.getEntityReference();
                Cargo x = (Cargo) obj;
                System.out.println("-----");
                System.out.println("Cargo weight is "+x.getWeight());
                System.out.println("Cargo volume is "+x.getVolume());
                System.out.println("Cargo container value is " +x.getContainer());
                System.out.println(m.getSeverity());
                System.out.println(m.getText());
            } catch(Exception e){
                System.out.println(e);
            }
        }

        public static void main(String[] args) {
            InvokeDS d = new InvokeDS();
            d.callDS();
        }
    }
}
```


Test the Decision Service

Now that you have written the Java client code, you can test it and confirm whether the code accesses the Decision Service.

1. Copy the Decision Service file, **Cargo_v0_16.eds**, from the workspace.
2. Paste it into **App**.
3. Run **InvokeDS.java**.

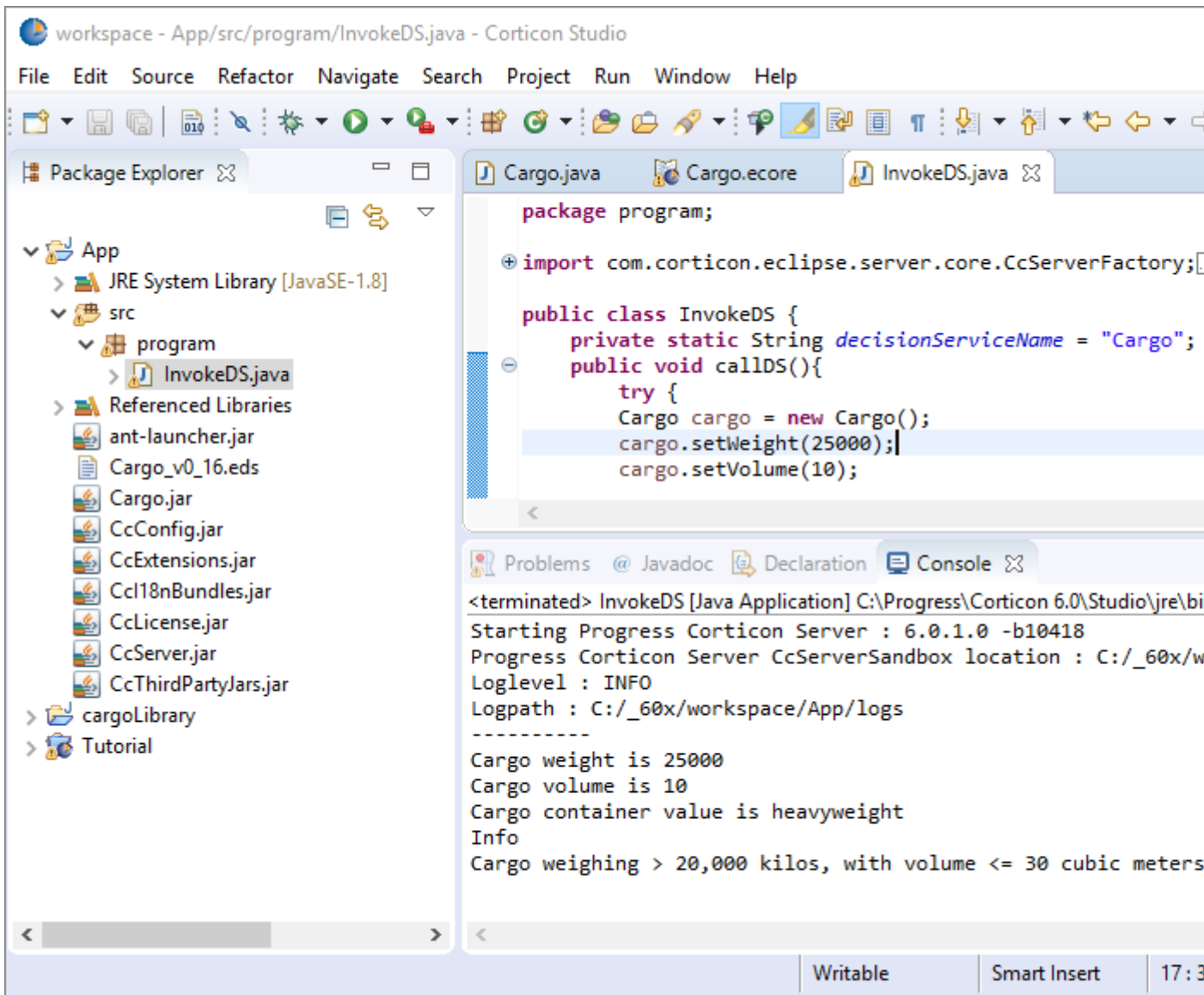


The Decision Service updated the container variable based on the values of weight and volume provided in the request object message. Based on the rules defined for this project in Corticon Studio, for an input weight of 1000 and volume of 10, the rules respond that a **standard** container should be used. The Decision Service is responding to the Java client code and the business rules.

- Let's change the value of weight defined in the Java client code to 25000.

```
try {
    Cargo cargo = new Cargo();
    cargo.setWeight(25000);    cargo.setVolume(10);
}
```

- Save and run the Java client code.



The rules triggered to prefer a heavyweight container.

The Decision Service is responding to the client code within the Java app.

Congratulations! You have completed this tutorial.

You have created Java Object get and set methods, mapped them to the Vocabulary, packaged the project as a compiled Decision Service file, and created Java client code to deploy and access the Decision Service. Finally, you tested the Java client code to confirm that it accesses the Decision Service.

You have created and tested a Corticon Decision Service deployed in-process under Java.

