**Progress**®

# Corticon Server:
## Deploying Web Services with Java

**Progress**® Corticon®

# Copyright

**Updated: 2018/09/27**

# Table of Contents

# Chapter 7: Summary of Java samples........................................................53

# Appendix A: Corticon 5.7.2 Online Tutorials and Documentation..........55

# 1

# Conceptual overview of the Java server

This guide is a walkthrough of fundamental concepts and functions of Corticon Server for Java. The examples focus on the default Corticon Server, the Progress Application Server (PAS) from Progress Software. The Progress Application Server is a platform that provides Web server support for Progress applications. Progress applications are packaged as Web application archives (WAR files) and deployed to the Java Servlet Container of a running instance of PAS.

The foundation of PAS is Apache Tomcat (see http://tomcat.apache.org/), a Web server that includes a Java servlet container for hosting Web applications. The Apache Tomcat that you can download from the Apache Software Foundation is tailored primarily as a development server for testing, validating and debugging Web applications. PAS is tailored primarily as a production server for deploying Progress web applications.

For details, see the following topics:

- What is a web service?

- What is a Decision Service?

- What is the Corticon Server for Java?

- What is a web services consumer?

## What is a web service?

**From the business perspective:** A Web Service is a software asset that automates a task and can be shared, combined, used, and reused by different people or systems within or among organizations.

**From the information systems perspective:** A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [From http://www.w3c.org.]

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

# What is a Decision Service?

A Decision Service automates a discrete decision-making task. It is implemented as a set of business rules and exposed as a Web Service (or Java Service). By definition, the rules within a Decision Service are complete and unambiguous; for a given set of inputs, the Decision Service addresses every logical possibility uniquely, ensuring decision integrity.

A Ruleflow is built in Corticon Studio. Once deployed to the Corticon Server, it becomes a Decision Service.

# What is the Corticon Server for Java?

Corticon Servers implement web services for business rules defined in Corticon Studios.

The Corticon Server is a high-performance, scalable and reliable system resource that manages pools of Decision Services and executes their rules against incoming requests. The Corticon Server can be easily configured as a Web Services server, which exposes the Decision Services as true Web Services.

Corticon Server is provided in two installation sets: Corticon Server for Java, and Corticon Server for .NET.

- The **Corticon Server for deploying web services with Java** -- the product documented here -- is supported on various application servers, databases, and client web browsers. The Java server is a component part of the Corticon Server installer. After installation on a supported Windows platform, that server installation's deployment artifacts can be redeployed on various UNIX and Linux web service platforms. The server components are also packaged for deployment on various EAR and WAR servers. See the *Integration and Deployment Guide* and the *Data Integration Guide* for more information. Also refer to the Progress Software web page Progress Corticon 5.7 - Supported Platforms Matrix.

- The **Corticon Server for deploying web services with .NET** facilitates deployment on Windows .NET framework and Microsoft Internet Information Services (IIS) that are packaged in the supported operating systems. The .NET server is a component part of the Corticon Server installer. See *Deploying Web Service with .NET* for more information.

# What is a web services consumer?

A Web Services Consumer is a software application that makes a request to, and receives a response from, a Web Service. Most modern application development environments provide native capabilities to consume Web Services, as do most modern Business Process Management Systems.

# 2

# Getting started

This guide steps through the procedures for running the Corticon Java Server as a Web Services server, deploying Ruleflows to the Server, exposing the Ruleflows as Decision Services, and then testing them with document-style SOAP requests. There are other installation, deployment and integration options available beyond the SOAP/Web Services method described here, including Java-centric options using Java objects and APIs. More detailed information on all available methods is contained in the *Integration & Deployment Guide*.

## Installing Corticon Server for Java

To work along with the material in this guide, you must download and install Corticon Server for Java on a supported Windows server machine. See the *Corticon Installation Guide* for details on installing this Corticon Server component.

**Note:**  When production systems are created, it is typical to move away from a standard Windows platform. Refer to the Progress Software web page Progress Corticon 5.7 - Supported Platforms Matrix to review the currently supported UNIX/Linux platforms and brands of Application Servers. Also see the Corticon KnowledgeBase entry  Corticon Server 5.X sample EAR/WAR installation for different Application Servers  for detailed instructions on configuring Apache Tomcat, JBoss, WebSphere, WebLogic on all supported platforms.

**Note:  HTTPS** - If you want to set up your Java server for SSL-secured communications, see the topic *"Setting up encryption between servers and clients" in the Integration and Deployment Guide.*

# 3

# Starting Corticon Server for Java

When you installed Corticon Server, it deployed into Progress Application Server during installation by pre-loading Corticon Server's `axis.war` file into its `webapps` directory:

```
[CORTICON_WORK_DIR]\Server\pas\server\webapps
```

Then, when you start the Corticon Server for the first time, it unpacks `axis.war`, and creates a new `axis` directory inside `\webapps` This new folder contains the complete Corticon Server web application.

---
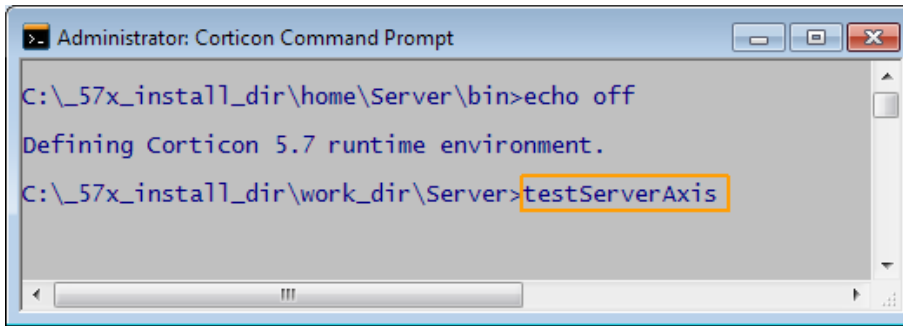
**Note:** If you intend to install Corticon Server on a web server other than the bundled Progress Application Server, refer to the *Integration & Deployment Guide* for details on using the standard `axis.war` package in your preferred web server.

---

To verify that Corticon Server is installed correctly on Progress Application Server:

1. Start Corticon Server from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.7** > **Start Corticon Server**

2. Open a Corticon Command Prompt from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.7** > **Corticon Command Prompt**

3. At the command prompt, enter `testServerAxis`, as shown:

4. At the `Enter transaction number:` prompt, enter `120`, as shown:



5. The test script retrieves the names of the running Decision Services, then displays them in a list, as shown:



The list of Decision Services indicates that the Corticon Server for Java is running correctly.

---

**Note:**  This tool is discussed in detail in the topic Testing the installed Corticon Server as a J2EE SOAP servlet on page 21

---

**Note:**  The **Start** menu command to **Start Corticon Server** launches the script `[CORTICON_HOME]\Server\bin\startServer.bat` to set the Corticon environment and the Java options before starting the Progress Progress Application Server (PAS). You should never directly launch the PAS startup script or the TCMAN server action as that will skip these crucial settings.

---

# 4

# Corticon Java Server files and API tools

Corticon Server for deploying web services with Java facilitates deployment on supported Windows operating systems. This guide describes various deployment technologies and strategies.

**Note:** When production systems are created, the skills and technologies you have learned on a Windows-based installation using the default application server will transfer readily to supported UNIX/Linux platforms and brands of Application Servers. Download packages at Progress Electronic Download and instructions in the Progress Corticon KnowledgeBase provide detailed configuration instructions.

This guide will first deploy a Ruleflow to the Java server as a Decision Service, then consume that Decision Service with various manual, SOAP/XML, and JSON/RESTful techniques.

But before exploring these features, you should become acquainted with some of the Corticon Server for .NET files and API tools.

For details, see the following topics:

- Basic server classes

- Setting up Corticon Server use cases

- The Corticon home and work directories

- The Corticon Server Sandbox

- Testing the installed Corticon Server

# Basic server classes

At its most basic level, Corticon Server is simply a set of Java classes, packaged in Java archive, or `.jar`, files. The minimum set of jars needed to deploy and call Corticon Server is listed below:

- `CcServer.jar` – The main Corticon Server JAR, containing the core engine logic.

- `CcConfig.jar` – Contains a set of text `.property` files that list and set all the configuration properties needed by Corticon Server. These properties pages are not intended for user access. Instead, the file `brms.properties`, installed by every product at the root of `[CORTICON_WORK_DIR]`, enables you to add your override settings to be applied after the default settings have been loaded.

- `CcLicense.jar` – An encrypted file containing the licensing information required to activate Corticon Server.

- `CcThirdPartyJars.jar` – Contains third-party software such as XML parsers and JDOM. Corticon warrants Corticon Server operation ONLY with the specific set of classes inside this jar.

- `CcI18nBundles.jar` – Contains the English and other language templates used by Corticon Server.

- `ant-launcher.jar` – Supports Corticon Server's deployment-time Decision Service compilation capability

- `CcExtensions.jar` – ***Optional.*** – Necessary only if you have added new rule language operators as "Extended Operators." (See the *Rule Language Guide* for details.)

# Setting up Corticon Server use cases

In most production deployments, Corticon Server JARs are bundled and given a J2EE interface class or classes. The interface class is often called a "helper" or "wrapper" class because its purpose is to receive the client application's invocation, translate it (if necessary) into a call which uses Corticon Server's native API, and then forwards the call to Corticon Server's classes. The type of interface class depends on the J2EE container where you intend to deploy the Corticon Server.

Corticon Studio makes in-process calls to the same Corticon Server classes (although packaged differently) when Ruletests are executed. This ensures that Ruleflows behave exactly the same way when executed in Studio Ruletests as they do when executed by Corticon Server, no matter how Corticon Server is installed.

**Note:** For detailed information on using packages that facilitate setup of Corticon Server for Java and Corticon Web Console on supported UNIX/Linux platforms and brands of Application Servers, refer to the Progress Software web page Progress Corticon 5.7 - Supported Platforms Matrix for the currently supported platforms and app servers. Then see the Corticon KnowledgeBase entry Corticon Server 5.X sample EAR/WAR installation for different Application Servers for detailed instructions on configuring Apache Tomcat, JBoss, WebSphere, WebLogic on all supported platforms. The Corticon Server ZIP download includes key scripts in shell script format for use in UNIX/Linux applications, such as `corticonManagement.sh`, `testServerAxis.sh`, and `testServerREST.sh`.

# Installing Corticon Server as a J2EE SOAP servlet

If *Installation Option 1 (Web Services) -- as described in Chapter 1 of the Integration and Deployment guide -- * was chosen, then a SOAP Servlet interface will be used for production deployments. Install Corticon Server into the Servlet container of a J2EE web or application server such as WebLogic, Oracle, or WebSphere. Commercial application servers offer the greatest degree of manageability, security and reliability, but open-source J2EE web servers such as Apache Tomcat -- especially as implemented by Progress in its Progress Application Server -- are also excellent, production-quality options. One advantage of the wrapper or helper class approach to installation is that variations in the web server environment (such as SOAP version) may be addressed in the wrapper class itself, rather than in the set of Corticon Server classes. The base set of Corticon Server jars remains the same irrespective of deployment environment – only the wrapper class changes.

The industry-standard method of deploying a Servlet into a J2EE web server's Servlet container is via a "web archive", or `.war` file. This file contains everything required to deploy a fully functional Servlet, including all classes, configuration files, and interfaces. Corticon provides a complete sample `.war` file, along with all source code, with the standard default Corticon Server installation. In addition to the base set of Corticon JARs, the provided `.war` file also contains Apache Axis SOAP messaging framework, which is supported by most commercial J2EE web and application servers, including the Progress Application Server. Your web server documentation will include instructions for installing or loading a `.war` file.

This `.war` file, `CcServer.war`, is available from the Progress download site in the `PROGRESS_CORTICON_5.7_SERVER.zip` package.

The unpackaged files are typically installed in the Corticon directory `[CORTICON_HOME]\Server\Containers\WAR`.

---

**Note:**  Refer to the Progress Software web page Progress Corticon 5.7 - Supported Platforms Matrix to review the currently supported UNIX/Linux platforms and brands of Application Servers. Also see the Corticon KnowledgeBase entry  Corticon Server 5.X sample EAR/WAR installation for different Application Servers  for detailed instructions on configuring Apache Tomcat, JBoss, WebSphere, WebLogic on all supported platforms.

---

**Important:**  The `.war` file provided is intended to be a sample wrapper for instructional purposes. Source code is provided in the `[CORTICON_HOME]Server\src` directory so you can adapt the wrapper to your environment and platform. **The sample `.war` file is not certified for use on any specific web server and is not warranted by Corticon .**

---

This `.war` file contains the default evaluation license, named `CcLicense.jar`. If you are a Corticon customer, you will be provided with a permanent version of this file. You must insert it into the `.war` (replacing the original) in order to remove the evaluation license limitations.

For a quick start, Corticon Server ships with Progress Application Server (built on Apache Tomcat), and the Apache Axis SOAP messaging infrastructure. The Corticon Server Installer sets up and configures Progress Application Server.

# Installing Corticon Server as a J2EE enterprise Java bean (EJB)

If *Installation Option 2 (Java Services with XML Payloads) or Installation Option 3 (Java Services with Java Object Payloads) -- as described in Chapter 1 of the Integration and Deployment guide --* was chosen above, then an EJB interface will be used for production deployments. Install Corticon Server into the EJB container of a J2EE application server such as WebLogic, Oracle, or WebSphere. Commercial application servers offer the greatest degree of manageability, security and reliability, but open-source J2EE application servers such as JBOSS are also available. One advantage of the wrapper or helper class approach to installation is that variations in the application server environment may be addressed in the wrapper class itself, rather than in the set of Corticon Server classes. The base set of Corticon Server JARs remains the same regardless of the deployment environment – only the wrapper class changes.

The industry-standard method of deploying an EJB into a J2EE application server's EJB container is via an "enterprise archive", or `.ear`, file. This file contains everything required to deploy a fully functional Session EJB (stateless), including all classes, configuration files and interfaces. Corticon includes a complete sample `.ear` file along with all source code with the standard default Corticon Server installation. Your application server documentation will include instructions for installing an `.ear` file.

---

**Note:** This `.ear` file, `CcServer.ear`, is available from the Progress download site in the `PROGRESS_CORTICON_5.7_SERVER.zip` package. The unpackaged files are typically installed in the Corticon directory `[CORTICON_HOME]\Server\Containers\EAR`. Refer to the Progress Software web page Progress Corticon 5.7 - Supported Platforms Matrix to review the currently supported UNIX/Linux platforms and brands of Application Servers. Also see the Corticon KnowledgeBase entry  Corticon Server 5.X sample EAR/WAR installation for different Application Servers  for detailed instructions on configuring Apache Tomcat, JBoss, WebSphere, WebLogic on all supported platforms.

---

This `.ear` file contains the default evaluation license, named `CcLicense.jar`. Progress Corticon customers are provided with a permanent version of this file. You must insert it into the `.ear` (replacing the original) in order to remove the evaluation license limitations.

The sample `.ear` file also contains `axis.war`, enabling you to deploy both a Servlet and an EJB version of Corticon Server simultaneously. This allows you to expose both SOAP and Java interfaces to the same Decision Service, making your Decision Services even easier to use throughout your enterprise infrastructure.

# Installing Corticon Server as Java classes in-process or in a custom Java container

If you choose to manage Corticon Server in-process via your client application or via a custom container, you are taking responsibility for many of the tasks that are normally performed by a J2EE web or application server. But by doing it in your own code, you can optimize your environment and eliminate unneeded overhead. This can result in much smaller footprint installations and faster performance.

Because Corticon Server is a set of Java classes, it can easily be deployed in-process in a JVM. When deployed in-process, the following tasks are the responsibility of the client application:

- Management of Java classpaths, ensuring the base set of Corticon Server classes is properly referenced.

- JVM lifecycle management, including startup/shutdown

- Concurrency & Thread management

- Security (if needed)

- Transaction management, including invocation of Corticon Server using the standard Java API set.

Corticon Server can also be installed into a custom container within any application. It has a small footprint and thus can be installed into client applications including browser-based applications, laptops and mobile devices.

For step-by-step instructions on using the Installer to gain access to Corticon Server's core jar files, see *"Installinig Corticon Servers and Web Console" in the Corticon Installation Guide.*

Installation in-process or in a custom container involves these basic steps:

1. Place the following Corticon Server JAR files in a location accessible by the surrounding Java container:

   - `CcServer.jar`
   - `CcConfig.jar`
   - `CcLicense.jar`
   - `CcThirdPartyJars.jar`
   - `CcI18nBundles.jar`
   - `ant_launcher.jar`
   - `CcExtensions.jar` [optional – only necessary if you have added custom rule language operators to the Operator Vocabulary as "Extended Operators." (See the *Corticon Studio: Extensions Guide* for more information.)]

2. Configure the Java classpath to include the JAR files listed above.

3. Write code that:

   - Initializes Corticon Server
   - Sets environment variables such as `CORTICON_HOME` and `CORTICON_WORK_DIR` (see The Corticon home and work directories on page 19)
   - Deploys the Decision Services into Corticon Server
   - Requests a decision by marshaling the data payload and then invoking the relevant Corticon Decision Service
   - Processes the response from the Decision Service.

Sample code is provided that demonstrates an in-process deployment of Corticon Server. This code, named `CcServerApiTest.bat`, is located in the `[CORTICON_HOME]\Server\src` directory.

For an example of a Java in-process installation along with a sample project life cycle, see "Tutorial: Deploying a Progress Corticon Decision Service in Process for Java" at the Corticon Learning Center, available at https://www.progress.com/tutorials/corticon.

# The Corticon home and work directories

As a Corticon installation completes, it tailors two properties that define its global environment. These variables are used throughout the product to determine the relative location of other files.

## Corticon environment

The installer establishes a common environment configuration file, `\bin\corticon_env.bat`, at the program installation location. That file defines the Progress Corticon runtime environment so that most scripts simply call it to set common global environment settings, such as `CORTICON_HOME` and `CORTICON_WORK_DIR` (and in some cases simply `CORTICON_WORK`.)

### CORTICON_HOME

The installation directory -- either the default location, `C:\Program Files\Progress\Corticon 5.7`, or the preferred location you specified -- is assigned to `[CORTICON_HOME]`.

### CORTICON_WORK_DIR

The work directory -- either the default location, `C:\Users\{`*username*`}\Progress\CorticonWork 5.7`, or the preferred location you specified -- is assigned to `[CORTICON_WORK_DIR]`.

### It is a good practice to use global environment settings

Many file paths and locations are determined by the `CORTICON_HOME` and `CORTICON_WORK_DIR` variables. Be sure to call `corticon_env.bat`, and then use these variables in your scripts and wrapper classes so that they are portable to deployments that might have different install paths.

---

**Note:** While you could change these locations with the assurance that well-behaved scripts will follow your renamed path or location, you might also encounter unexpected behaviors from any that do not. Also, issues might arise when running update, upgrade, and uninstall utilities.

---

# The Corticon Server Sandbox

When Corticon Server starts up, it checks for the existence of a "sandbox" directory. This Sandbox is a directory structure used by Corticon Server to manage its state and deployment code.

The location of the Sandbox is controlled by `com.corticon.ccserver.sandboxDir` settings in your `brms.properties` file. This configuration setting is defined by the `CORTICON_WORK_DIR` variable, in this case:

```
com.corticon.ccserver.sandboxDir=%CORTICON_WORK_DIR%/%CORTICON_SETTING%/CcServerSandbox
```

In a default Windows installation, the result for this is `C:\Users\{username}\Progress\CorticonWork_5.7\SER\CcServerSandbox`. In other words, in the `SER` subdirectory of the `CORTICON_WORK_DIR`. This directory is created (as well as peer directories, `logs` and `output`) during the first launch of Corticon Server.

---

**Note:** If the location specified by `com.corticon.ccserver.sandboxDir` cannot be found or is not available, the Sandbox location defaults to the current working directory as it is typically the location that initiated the call.

---

# Testing the installed Corticon Server

With Corticon Server installed in the environment and container of your choice, it is useful to test the installation to ensure Corticon Server is running and listening. At this point, no Decision Services have been deployed, so Corticon Server is not ready to process transactions. However, the Corticon Server API set contains administrative methods that interrogate it and return status information. Several tools are provided to help you perform this test.

> **Note:** The Corticon **Start** menu provides a **Corticon Command Prompt** command that calls
> `corticon_env.bat`, adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts
> by name from several locations -- `\bin`, `\Server\bin`, `\Server\pas\bin`, `\Studio\bin`, and,
> `\Studio\eclipse` -- and then relocates the prompt to the root of the Corticon work directory.

# Testing the installed Corticon Server as a J2EE SOAP servlet

To test that Corticon Server deployed as a SOAP Servlet is running correctly, all you need is a SOAP client or
the sample batch file provided and described below.

Testing the Servlet installation here assumes you have already installed and started Corticon Server as a Web
Service in the bundled Progress Application Server or using the `.war` file in another web server.

Because a SOAP Servlet is listening for SOAP calls, we need a way to invoke an API method via a SOAP
message then send that message to Corticon Server using a SOAP client. In the sample code supplied in the
default installation, Corticon provides an easy way to make API calls to it via a SOAP message.

The included batch file, `testServerAxis.bat`, will help ensure that Corticon Server is installed properly and
listening for calls. Located in the `[CORTICON_HOME]\Server\bin` directory, this script provides a menu of
available Corticon Server methods to call into the SOAP Servlet. Running `testServerAxis.bat` (or `.sh` in
a UNIX environment) does the following:

- Sets classpaths needed by the `CcServerTest` class, which is acting as our menu-driven SOAP client.
  The source code (`.java`) is included in the `[CORTICON_WORK_DIR]\Samples\Clients\SOAP` directory.

- Defines variables for web server location and ports. Port changes may be necessary depending on the type
  of application or web server you are using to host the Servlet.
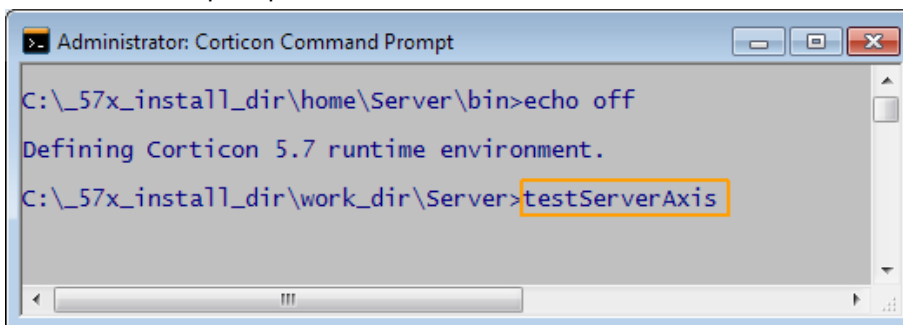
> **Note:** The bundled Progress Application Server uses `localhost` as the application server's location, and
> defaults to port settings of `8850` for HTTP and `8851` for HTTPS.

- Starts a JVM for our SOAP client class, `CcServerTest`, to run inside.

- Calls the `CcServerTest` class (our simple SOAP client) with arguments for web server location and port.
  Notice that the rest of the URI has been hard-coded in this batch file.

**To use the server test script:**

1. Open a Corticon Command Prompt from the Windows **Start** menu by choosing **All Programs** > **Progress**
   > **Corticon 5.7** > **Corticon Command Prompt**

2. At the command prompt, enter `testServerAxis`, as shown:

> **Note:** When you launch Corticon Command Prompt, it calls `corticon_env.bat`, adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts by name from several locations -- `\bin`, `\Server\bin`, `\Server\pas\bin`, `\Studio\bin`, and, `\Studio\eclipse` -- and then relocates the prompt to the root of the Corticon work directory.

**3.** When the `Enter transaction number:` prompt displays, enter `help`, as shown:

```
Administrator: Corticon Command Prompt - testserveraxis

C:\_57x_install_dir\work_dir\Server>testserveraxis

C:\_57x_install_dir\work_dir\Server>echo off

Defining Corticon 5.7 runtime environment.

C:\_57x_install_dir\work_dir\Server>CALL "C:\_57x_install_dir\home\Server\Server\JRE\bin\java.exe" -cp
 "C:\_57x_install_dir\home\Server\Server\lib\CcConfig.jar;C:\_57x_install_dir\home\Server\Server\lib\C
cServer.jar;C:\_57x_install_dir\home\Server\Server\lib\CcI18nBundles.jar;C:\_57x_install_dir\home\Serv
er\Server\lib\CcThirdPartyJars.jar;C:\_57x_install_dir\home\Server\Server\lib\activation-1.2.jar;C:\_5
7x_install_dir\home\Server\Server\lib\axis.jar;C:\_57x_install_dir\home\Server\Server\lib\commons-disc
overy.jar;C:\_57x_install_dir\home\Server\Server\lib\commons-logging-1.1.3.jar;C:\_57x_install_dir\hom
e\Server\Server\lib\jaxrpc.jar;C:\_57x_install_dir\home\Server\Server\lib\mail-1.2.jar;C:\_57x_install
_dir\home\Server\Server\lib\saaj.jar;C:\_57x_install_dir\home\Server\Server\lib\soap-2.3.1.jar" -DCORT
ICON_HOME="C:\_57x_install_dir\home\Server" -DCORTICON_WORK_DIR="C:\_57x_install_dir\work_dir\Server"
com.corticon.eclipse.server.core.CcServerApiTest remote http://localhost:8850 axis


----------------------------------------------------------------
Axis Servlet Location : http://localhost:8850
Execute Servlet (Message Style) : http://localhost:8850/axis/services/Corticon
Execute Servlet (RPC Style)     : http://localhost:8850/axis/services/CorticonExecute
Admin Servlet                   : http://localhost:8850/axis/services/CorticonAdmin
Heartbeat                       : http://localhost:8850/axis/services/CorticonHeartbeat
Output Directory                : C:/_57x_install_dir/work_dir/Server/output

----------------------------------------------------------------
Enter transaction number:
help
```
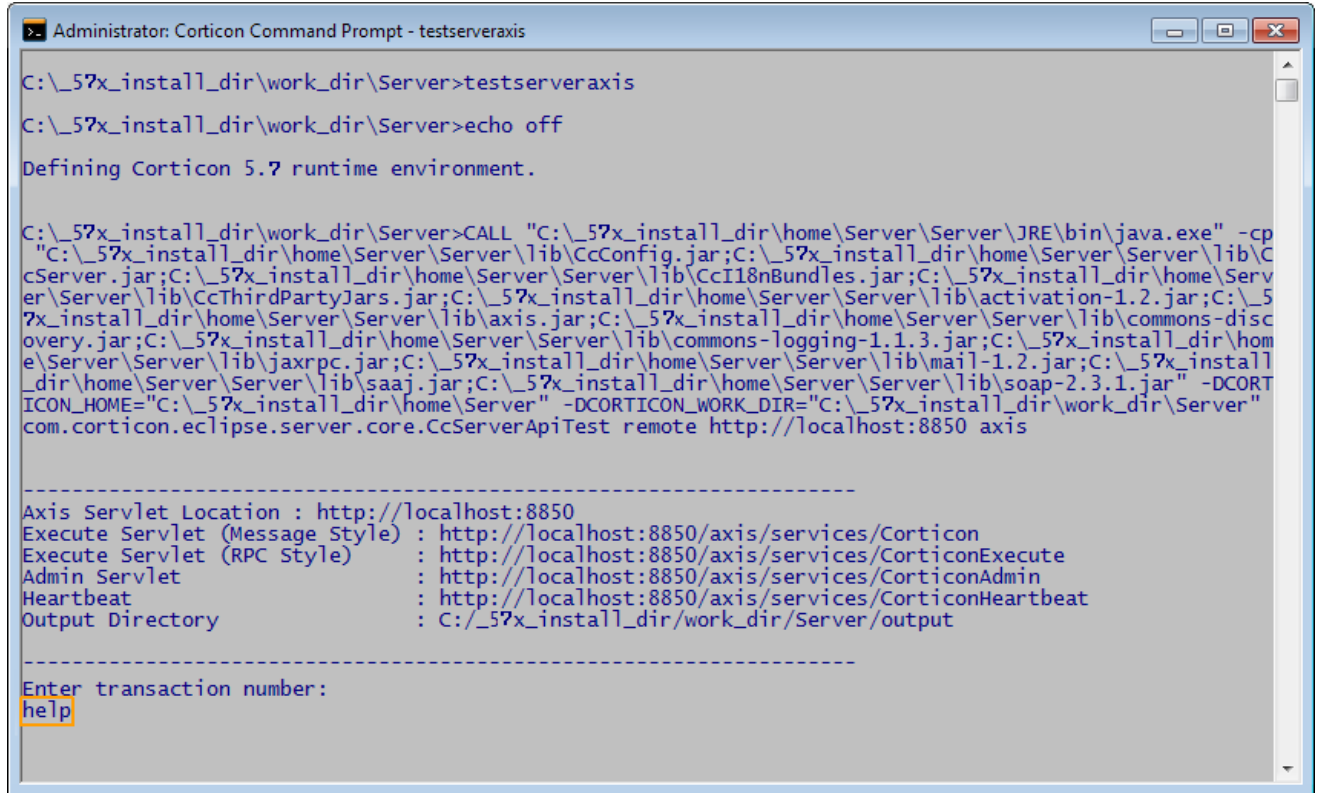
**4.** The 100 series commands are listed.

```
Administrator: Corticon Command Prompt - testserveraxis

-------------------------------------------------------------
Enter transaction number:
help


-------------------------------------------------------
--- Current Apache Axis Location: http://localhost:8850
-------------------------------------------------------

Transactions:
 -1 - Exit Server Api Test
------------------------------------------------------------------------
   0 - Change Connection Parameters
------------------------------------------------------------------------
101 - Add a Decision Service (3 parameters)
102 - Add a Decision Service (6 parameters)
103 - Add a Decision Service (9 parameters)
------------------------------------------------------------------------
110 - Load CcServer with .cdd file
111 - Load CcServer files from directory
------------------------------------------------------------------------
112 - Reload Decision Service
113 - Reload Decision Service (by specific Decision Service Major Version)
114 - Reload Decision Service (by specific Decision Service Major and Minor Version)
------------------------------------------------------------------------
115 - Remove Decision Service
116 - Remove Decision Service (by specific Decision Service Major Version)
117 - Remove Decision Service (by specific Decision Service Major and Minor Version)
------------------------------------------------------------------------
118 - Clear All Non-Cdd Decision Services
------------------------------------------------------------------------
120 - Get Decision Service Names
121 - Get CcServer current info
------------------------------------------------------------------------
130 - Execute SOAP Document Style (CorticonRequest Document)
131 - Execute SOAP RPC Style (CorticonRequest String)
------------------------------------------------------------------------
150 - Precompile a Ruleflow into a .eds file
151 - Precompile a Ruleflow into a Database Access optimized .eds file
------------------------------------------------------------------------
100 - Switch menu to Common Functions
200 - Switch menu to Decision Service Functions
300 - Switch menu to Monitoring Functions
400 - Switch menu to CcServer Functions
------------------------------------------------------------------------

Enter transaction number:
```

In the lower portion of the Windows console, shown in the figure above, we see the available API methods of the **Common Functions** (the 100 series) listed by number. You can list the commands in the other series by entering their series number:

- Enter `200` to list the Decision Service Functions command set

- Enter `300` to list the Monitoring Functions command set

- Enter `400` to list the CcServer Functions command set

- Enter `100` to again list the Common Functions command set

Since we have not deployed any Ruleflows yet, we will use an administrative method to test if Corticon Server is correctly installed as a SOAP Servlet inside our web server.

---

**Note:** Even though the script is running, a server connection has not yet been attempted. If you enter any command when the application server is not running, you will get a `Connection refused` exception.
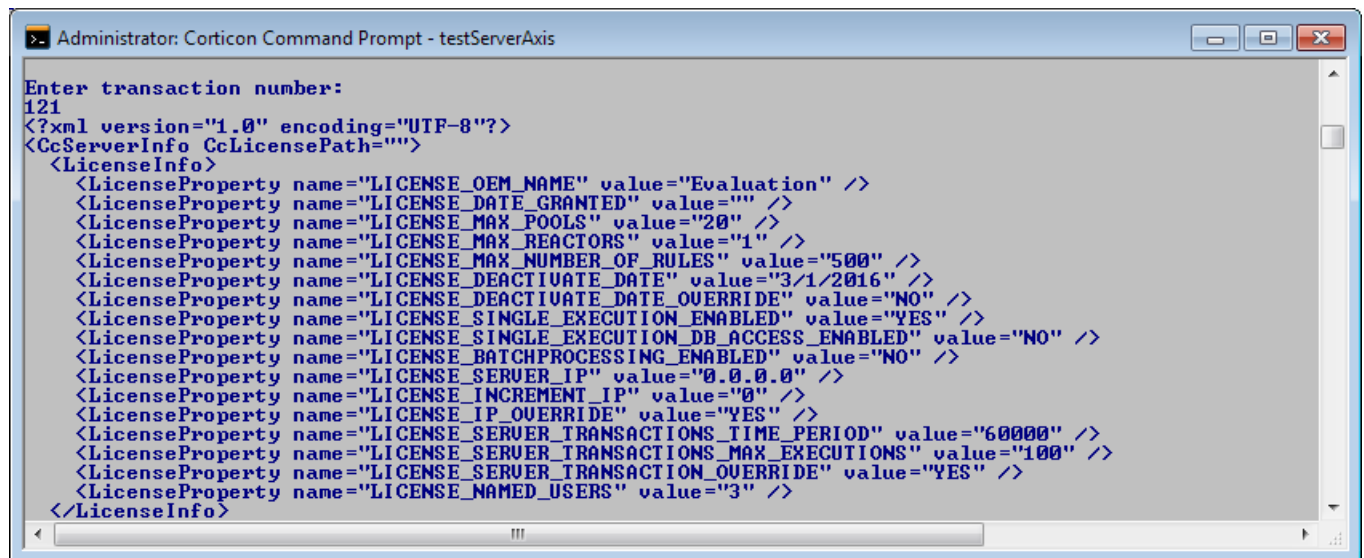
---

A good administrative method to call is transaction #121, **Get CcServer current info**. This choice corresponds directly to the Java API method `getCcServerInfo()`, described in complete detail in the *JavaDocs* provided in the standard Corticon Server installation.
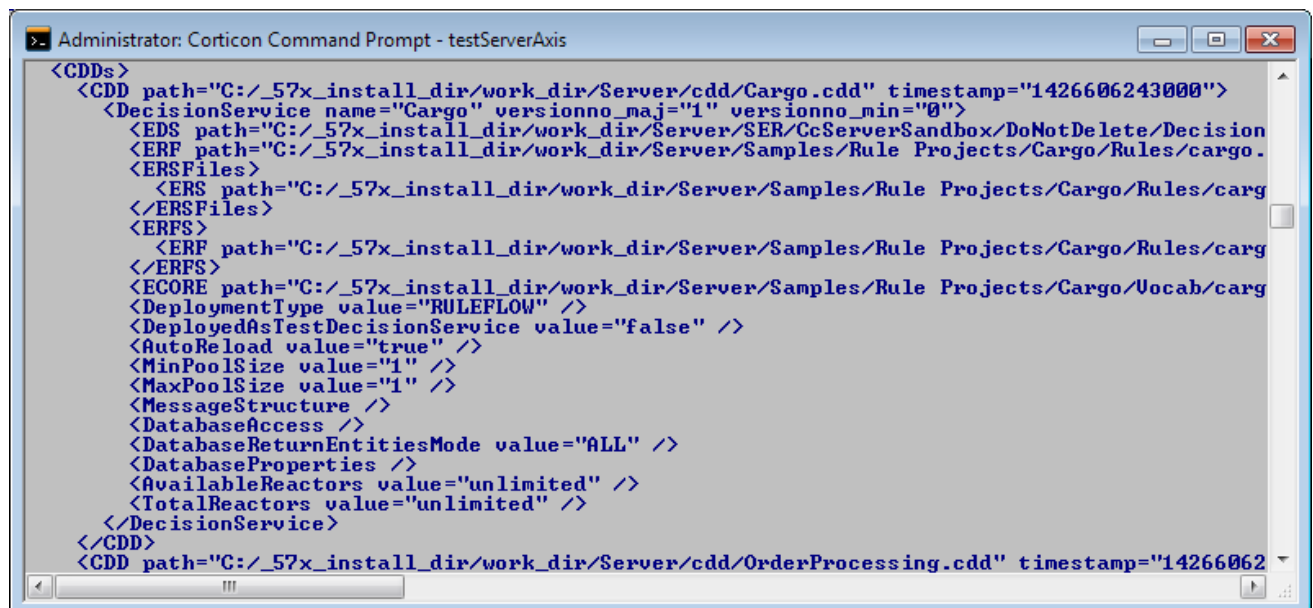
To try this, confirm that Corticon Server is running, and then enter `121` in the `testServerAxis` window. The `CcServerAxisTest` class makes a call to the server's SOAP Servlet. It asks for a list of configuration parameters and returns them to the Windows console. The results of the call are shown in the following figures:

**Figure 1: `testServerAxis` Response to command 121: License information**



**Figure 2: `testServerAxis` Response to command 121: Info on one of the deployed Decision Services**



**Figure 3: `testServerAxis` Response to command 121: Additional Server information**

The response verifies that our Corticon Server is running correctly as a SOAP Servlet and is listening for -- and responding to -- calls. At this stage in the deployment, this is all we want to verify.

# Testing the installed Corticon Server as in-process Java classes

The batch file `testServer.bat` is located in the `[CORTICON_HOME]\Server\bin` is designed to perform the basic in-process initialization of a Corticon Server, and then present a menu of API methods you can invoke from within a Windows console.

Viewing `testServer.bat` with a text editor, you can see how it sets classpaths, starts the JVM and then invokes the `CcServerApiTest` class. The source code (`.java`) for this class is provided in the `[CORTICON_WORK_DIR]\Samples\Clients\SOAP` directory. It is a good reference to use when you want to see exactly how the Corticon Server API set is used in your own code.

In addition to the several base Corticon Server JARs listed in Installing Corticon Server as Java classes in-process or in a custom java container section, the batch file also loads some hard-coded Java business objects for use with the Java Object Messaging commands 132-141. These hard-coded classes are included in `CcServer.jar` so as to ensure their inclusion on the JVM's classpath whenever `CcServer.jar` is loaded. The hard-coded Java objects are intended for use when invoking the `OrderProcessing.erf` Decision Service included in the default Corticon Server installation.

**Figure 4: testServer.bat**



**To use the in-process server's test script:**

1.  Open a Corticon Command Prompt from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.7** > **Corticon Command Prompt**

2.  At the command prompt, enter `testServer`.

> **Note:** When you launch Corticon Command Prompt, it calls `corticon_env.bat`, adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts by name from several locations, and then relocates the prompt to the root of the Corticon work directory.

3.  When the `Enter transaction number:` prompt displays, enter `help`, as shown:

```
Enter transaction number:
help
```

4.  The 100 series commands are listed.

```
Administrator: Corticon Command Prompt - testServer

Enter transaction number:
help

Transactions:
 -1 - Exit Server Api Test
-----------------------------------------------------------------------------------
101 - Add a Decision Service (3 parameters)
102 - Add a Decision Service (6 parameters)
103 - Add a Decision Service (9 parameters)
-----------------------------------------------------------------------------------
110 - Load CcServer with .cdd file
111 - Load CcServer files from directory
-----------------------------------------------------------------------------------
112 - Reload Decision Service
113 - Reload Decision Service (by specific Decision Service Major Version)
114 - Reload Decision Service (by specific Decision Service Major and Minor Version)
-----------------------------------------------------------------------------------
115 - Remove Decision Service
116 - Remove Decision Service (by specific Decision Service Major Version)
117 - Remove Decision Service (by specific Decision Service Major and Minor Version)
-----------------------------------------------------------------------------------
118 - Clear All Non-Cdd Decision Services
-----------------------------------------------------------------------------------
120 - Get Decision Service Names
121 - Get CcServer current info
-----------------------------------------------------------------------------------
130 - Execute using a JDOM Document (CorticonRequest Document)
131 - Execute using a XML String (CorticonRequest String)
-----------------------------------------------------------------------------------
132 - Execute using a hard-coded set of Business Objects (Collection)
133 - Execute using a hard-coded set of Business Objects (Collection) (by specific Decision Service Major Version)
134 - Execute using a hard-coded set of Business Objects (Collection) (by specific Decision Service Major and Minor Version)
135 - Execute using a hard-coded set of Business Objects (Collection) (by specific execution Date)
136 - Execute using a hard-coded set of Business Objects (Collection) (by specific execution Date and Decision Service Major Versi
-----------------------------------------------------------------------------------
137 - Execute using a hard-coded set of Business Objects (HashMap)
138 - Execute using a hard-coded set of Business Objects (HashMap) (by specific Decision Service Major Version)
139 - Execute using a hard-coded set of Business Objects (HashMap) (by specific Decision Service Major and Minor Version)
140 - Execute using a hard-coded set of Business Objects (HashMap) (by specific execution Date)
141 - Execute using a hard-coded set of Business Objects (HashMap) (by specific execution Date and Decision Service Major Version)
-----------------------------------------------------------------------------------
150 - Precompile a Ruleflow into a .eds file
151 - Precompile a Ruleflow into a Database Access optimized .eds file
-----------------------------------------------------------------------------------
100 - Switch menu to Common Functions
200 - Switch menu to Decision Service Functions
300 - Switch menu to Monitoring Functions
400 - Switch menu to CcServer Functions
-----------------------------------------------------------------------------------

Enter transaction number:
```

These are the available API methods of the **Common Functions** (the 100 series) listed by number. You can list the commands in the other series by entering their series number:

- Enter `200` to list the Decision Service Functions command set

- Enter `300` to list the Monitoring Functions command set

- Enter `400` to list the CcServer Functions command set

- Enter `100` to again list the Common Functions command set

5. To get information about the in-process Server, enter `121` in the `testServer` window. The `CcServerTest` class makes a call to the server's SOAP Servlet. It asks for a list of configuration parameters and returns them to the Windows console.

The response verifies that our in-process Corticon Server is running correctly as in-process Java classes and is listening for -- and responding to -- calls. At this stage in the deployment, this is all we want to verify.
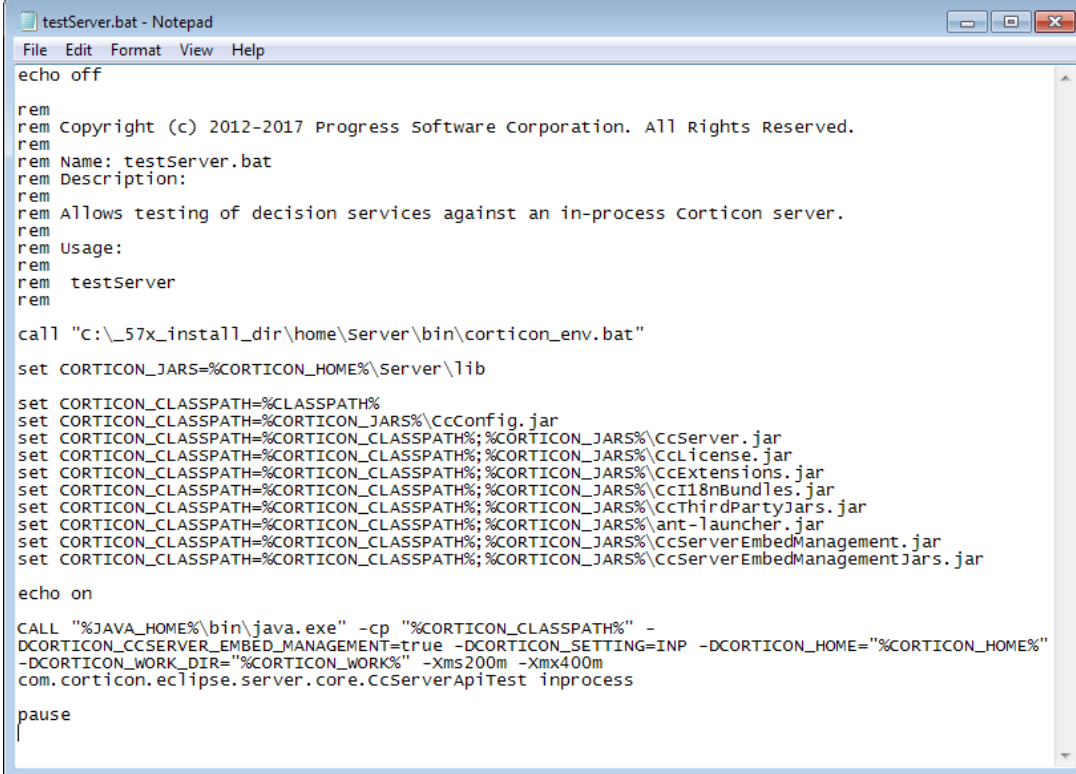
# 5

# Deploying a Ruleflow to the Corticon Server

Just because the Corticon Server is running does not mean it is ready to process transactions. It must still be "loaded" with one or more Ruleflows. Once a Ruleflow has been loaded, or deployed, to the Corticon Server we call it a Decision Service because it is a service ready and able to make decisions for any external application or process ("client") that requests the service properly.

Loading the Corticon Server with Ruleflows can be accomplished in several ways:

- **Package and Deploy Decision Services wizard** - The easiest method. It is discussed in detail in the topics in *"Using Corticon Studio to compile and deploy Decision Services" in the Integration and Deployment Guide*.

- **Deployment Descriptor files** - An easy method, and the one we will use in this guide.

- **Deployment using the Server Web Console** - Another easy way to load Decision Services. It is discussed in detail in topics in *"Decision Services and Applications" in the Web Console Guide*.

- **Compile and deploy APIs and command line utilities,** - These methods require more expertise in development tools, and are not discussed in this guide. For more information, see the topics in *"Packaging and deploying Decision Services" in the Integration and Deployment Guide*.

All these methods are described more thoroughly in the *Server Integration & Deployment Guide*.

For details, see the following topics:

- Creating a Ruleflow

- Creating and installing a Deployment Descriptor file

# Creating a Ruleflow

You created a Ruleflow suitable for deployment in <span style="color:blue">Tutorial: Advanced Rule Modeling in Corticon Studio</span> that is ready for deployment to the Corticon Server.

You could also use a simple one (a single Rulesheet) that was installed in the `Cargo` tutorial files, `tutorial_example.erf` located in the server's `[CORTICON_WORK_DIR]\Samples\Rule Projects\Tutorial\Tutorial-Done`.

To create a new Ruleflow, see the topic *"Creating a New Ruleflow" in the Quick Reference Guide.*

# Creating and installing a Deployment Descriptor file

A Deployment Descriptor file tells the Corticon Server which Ruleflows to load and how to handle transaction requests for those Ruleflows. A Deployment Descriptor file has the suffix `.cdd`, and we will often simply refer to it as a `.cdd` file.

---

**Important:** The `.cdd` file "points" at the Ruleflow via a path name - a name can contain spaces but it is a good practice to avoid spaces and special characters except for underscore (_) character.

---
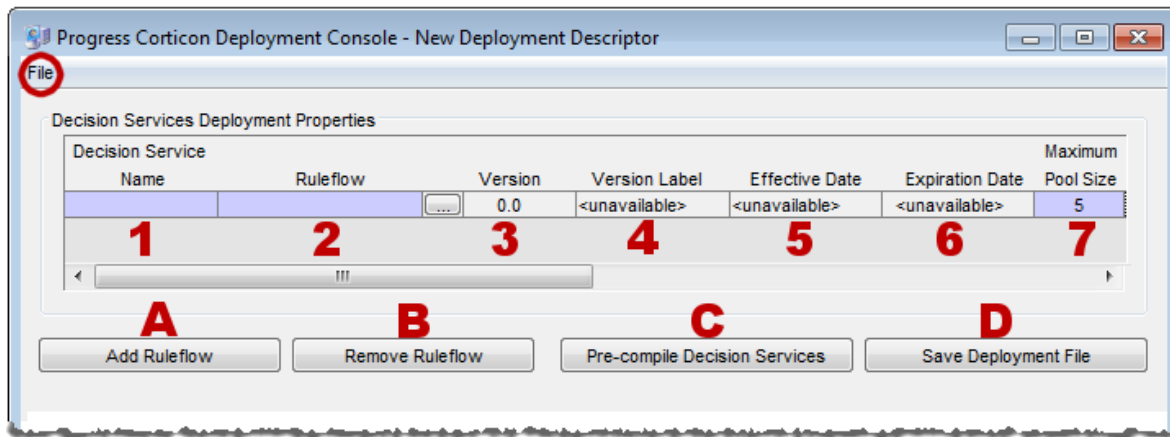
Deployment Descriptors are easily created using the Deployment Console, which is installed only by the Corticon Server installers.

## Using the Java Server's Deployment Console Decision Services
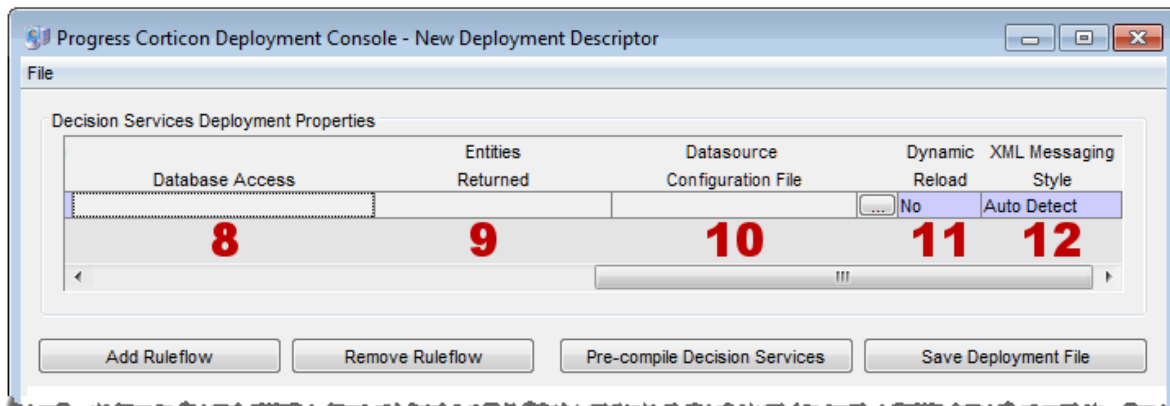
To start the Corticon Deployment Console, choose the Windows **Start** menu command **All Programs** > **Progress** > **Corticon 5.7** > **Corticon Deployment Console** to launch the script file `\Server\deployConsole.bat`.

The Deployment Console is divided into two sections. Because the Deployment Console is a rather wide window, its columns are shown as two screen captures in the following figures. The **red** identifiers are the topics listed below.

**Figure 5: Left Portion of Deployment Console, with Deployment Descriptor File Settings Numbered**



**Figure 6: Right Portion of Deployment Console, with Deployment Descriptor File Settings Numbered**



The name of the open Deployment Descriptor file is displayed in the Deployment Console's title bar.

The **File** menu, circled in the top figure, enables management of Deployment Descriptor files:

- To save the current file, choose (**File** > **Save**).

- To open an existing `.cdd`, choose (**File** > **Open**).

- To save a `.cdd` under a different name, choose (**File** > **Save As**).

The marked steps below correspond to the Deployment Console columns for each line in the Deployment Descriptor.

1. **Decision Service Name** - A unique identifier or label for the Decision Service. It is used when invoking the Decision Service, either via an API call or a SOAP request message. See *"Invoking Corticon Server" in the Integration and Deployment Guide.* for usage details.

2. **Ruleflow** - All Ruleflows listed in this section are part of this Deployment Descriptor file. Deployment properties are specified on each Ruleflow. Each row represents one Ruleflow. Use the [...] button to navigate to a Ruleflow file and select it for inclusion in this Deployment Descriptor file. Note that Ruleflow *absolute* pathnames are shown in this section, but *relative* pathnames are included in the actual `.cdd` file.

The term "deploy", as we use it here, means to "inform" the Corticon Server that you intend to load the Ruleflow and make it available as a Decision Service. It does **not** require actual physical movement of the `.erf` file from a design-time location to a runtime location, although you may do that if you choose – just be sure the file's path is up-to-date in the Deployment Descriptor file. But movement isn't required – you can save your `.erf` file to any location in a file system, and also deploy it from the same place *as long as the running Corticon Server can access the path*.

3. **Version** - the version number assigned to the Ruleflow in the **Ruleflow** > **Properties** window of Corticon Studio. Note that this entry is editable only in Corticon Studio and not in the Deployment Console. A discussion of how Corticon Server processes this information is found in the topics *"Decision Service Versioning and Effective Dating" of the Integration and Deployment Guide.* Also see the *Quick Reference Guide* for a brief description of the Ruleflow Properties window and the *Rule Modeling Guide* for details on using the Ruleflow versioning feature. It is displayed in the Deployment Console simply as a convenience to the Ruleflow deployer.

4. **Version Label** - the version label assigned to the Ruleflow in the **Ruleflow** > **Properties** window of Corticon Studio. Note that this entry is editable only in Corticon Studio and not in the Deployment Console. See the *Quick Reference Guide* for a brief description of the Ruleflow Properties window and the purpose of the Ruleflow versioning feature.

5. **Effective Date** - The effective date assigned to the Ruleflow in the **Ruleflow** > **Properties** window of Corticon Studio. Note that this entry is editable only in Corticon Studio and not in the Deployment Console. A discussion of how Corticon Server processes this information is found in the topics *"Decision Service Versioning and Effective Dating" of the Integration and Deployment Guide.* Also see the *Quick Reference Guide* for a brief description of the Ruleflow Properties window and the purpose of the Ruleflow effective dating feature.

6. **Expiration Date** - The expiration date assigned to the Ruleflow in the **Ruleflow** > **Properties** window of Corticon Studio . Note that this entry is editable only in Corticon Studio and not in the Deployment Console. A discussion of how Corticon Server processes this information is found in the topics *"Decision Service Versioning and Effective Dating" of the Integration and Deployment Guide.* Also see the *Quick Reference Guide* for a brief description of the Ruleflow Properties window and the purpose of the Ruleflow expiration dating feature.

7. **Maximum Pool Size** - Specifies how many execution threads for this Decision Service will be added to the Execution Queue. This parameter is an issue only when Allocation is turned on. If you are evaluating Corticon, your license requires that you set the parameter to **1**.See *'Multi-threading, concurrency reactors, and server pools' in "Inside Corticon Server" section of the Integration and Deployment Guide* for more information.

---

**Note:  Minimum Pool Size**, previously associated with this property, is deprecated as of version 5.5.

---

8. **Database Access** - Controls whether the deployed Rule Set has direct access to a database, and if so, whether it will be read-only or read-write access.

9. **Entities Returned** - Determines whether the Corticon Server response message should include all data used by the rules including data retrieved from a database (**All Instances**), or only data provided in the request and created by the rules themselves (**Incoming/New Instances**).

10. **Datasource Configuration File** - The path and filename of the database access properties file (that was typically created in Corticon Studio) to be used by Corticon Server during runtime database access. Use the adjacent



button to navigate to a Datasource configuration file.

11. **Dynamic Reload** - When **Yes**, the `ServerMaintenanceThread` will detect if the Ruleflow or `.eds` file has been updated; if so, the Decision Service will be updated into memory and -- for any subsequent calls to that Decision Service -- that execution Thread will execute against the newly updated Rules. When **No**,

the `CcServerMaintenanceThread` will ignore any changes to the Ruleflow or `.eds` file. The changes will not be read into memory, and all execution Threads will execute against the existing Rules that are in memory for that Decision Service.

12. **XML Messaging Style** - Determines whether request messages for this Decision Service should contain a flat (**Flat**) or hierarchical (**Hier**) payload structure. The topic *"Service Contracts: Describing the call" topic in the 'Integrating Corticon Decision Services' chapter of the Integration and Deployment Guide.* provides samples of each. If set to **Auto Detect**, then Corticon Server will accept either style and respond in the same way.

The indicated buttons at the bottom of the Decision Service Deployment Properties section provide the following functions:

- **(A) Add Ruleflow** - Creates a new line in the Decision Service Deployment Properties list. There is no limit to the number of Ruleflows that can be included in a single Deployment Descriptor file.

- **(B) Remove Ruleflow** - Removes the selected row in the Decision Service Deployment Properties list.

- **(C) Pre-compile Decision Services** - Compiles the Decision Service before deployment, and then puts the `.eds` file (which contains the compiled executable code) at the location you specify. (By default, Corticon Server does not compile Ruleflows *until* they are deployed to Corticon Server. Here, you choose to pre-compile Ruleflows in advance of deployment.) The `.cdd` file will contain reference to the `.eds` instead of the usual `.erf` file. Be aware that setting the EDC properties will optimize the Decision Service for EDC.

- **(D) Save Deployment File** - Saves the `.cdd` file. (Same as the menu **File** > **Save** command.)

# Installing the Deployment Descriptor file

Once Corticon Server has been installed and deployed to Progress Application Server, the included startup scripts ensure the following sequence occurs upon launching Progress Application Server:

**Note:** If you are using an evaluation license, the Maximum Pool Size in the Ruleflow you are deploying must be exactly `1`. Any other value will issue an alert that you have exceeded the allowed number of reactors.

1. The Progress Application Server starts up.

2. Corticon Server starts up as a web service in Progress Application Server's Servlet container.

3. Corticon Server looks for Deployment Descriptor files in a specific directory.

4. Corticon Server loads into memory the Ruleflow(s) referenced by the Deployment Descriptor files, and creates Reactors for each according to their minimum pool size settings. At this stage, we say that the Ruleflows have become Decision Services because they are now usable by external applications and clients.

In order for the Corticon Server to find Deployment Descriptor files when it looks in step 3, we must ensure that the `.cdd` files are moved to the appropriate location. In the default installation used in this guide, that location is the `[CORTICON_WORK_DIR]\cdd` directory. In the future, when creating `.cdd` files, you may want to save them straight to this directory so they become immediately accessible to the default Corticon Server deployed in this guide.

This location is fully configurable. For more information, see the topics in *"Packaging and deploying Decision Services" in the Integration and Deployment Guide*.

Now, when the startup sequence reaches step 3, Corticon Server will "know" where all Ruleflows are located because `.cdd` files contain their pathnames.

# Hot re-deploying Deployment Descriptor files and Ruleflows

Changes to a Deployment Descriptor file or any of the Ruleflows it references do **not** require restarting the application server. A maintenance thread in the Corticon Server watches for additions, deletions, and changes and updates appropriately. A Ruleflow can be modified in Corticon Studio even while it is also simultaneously deployed as a Decision Service and involved in a transaction - Corticon Server can be configured to update the Decision Service dynamically for the very next transaction.

Dynamic updating of deployed Ruleflows is not normally used in production environments because standard IT change control processes require a more disciplined and controlled deployment process. But in development or testing environments, it can be convenient to allow dynamic updates so that Ruleflow changes can be deployed more quickly.

Having selected `No` for the **Dynamic Reload** setting earlier, our `tutorial_example` Decision Service will not update automatically when the `.erf` file is changed. To enable this automatic refresh, choose `Yes` for the **Dynamic Reload** setting.

# 6

# Consuming a Decision Service on Java server

Let's review what we have accomplished so far:

1. We have installed Corticon Server for Java onto the target machine.

2. We have deployed Corticon Server as a web service onto the bundled Progress Application Server.

3. We have used the **Deployment Console** to generate a Deployment Descriptor file for our sample Ruleflow.

4. We have installed the Deployment Descriptor file in the location where Corticon Server looks when it starts.

Now we are ready to consume this Decision Service by sending a real XML/SOAP "request" message and inspecting the response message that returns.

For details, see the following topics:

- Integrating and testing a Decision Service on Java server

- Path 1: Using Corticon Studio as a SOAP client to consume a Decision Service

- Path 2: Using bundled sample code to consume a Decision Service

- Path 3: Using SOAP client to consume a Decision Service

- Path 4: Using JSON/RESTful client to consume a Decision Service on Java Server

- Troubleshooting Java server

# Integrating and testing a Decision Service on Java server

In order to use a Decision Service in a process or application, it is necessary to understand the Decision Service's service contract, also known as its interface. A service contract describes in precise terms the kind of input a Decision Service is expecting, and the kind of output it returns following processing. In other words, a service contract describes how to *integrate* with a Decision Service.

When an external process or application sends a request message to a Decision Service that complies with its service contract, the Decision Service receives the request, processes the included data, and sends a response message. When a Decision Service is used in this manner, we say that the external application or process has successfully "consumed" the Decision Service.

This guide describes four paths for consuming a Decision Service:

- Path 1

  **Use Corticon as a SOAP client to send and receive SOAP messages to a Decision Service running on a remote Corticon Server** - This is different from testing Ruleflows in Corticon "locally." This path is the easiest method to use and requires the least amount of technical knowledge to successfully complete. If you have already installed Corticon Studio, then you have all necessary components to complete this path. If not but want to follow this path, we recommend completing the *Corticon Installation Guide* and the *Corticon Studio Tutorial: Basic Rule Modeling* before continuing on this path.

- Path 2

  **Manually integrate and test a Decision Service** - In this path, we will use bundled sample code (a command file) to send a request message built in Corticon Studio's Tester, and display the results. This path requires more technical knowledge and confidence to complete, but illustrates some aspects of the software which may be interesting to a more technical audience. If you have already installed Corticon Studio, then you have all necessary components to complete this path. If not but want to follow this path, we recommend completing the *Corticon Installation Guide* and the *Corticon Studio Tutorial: Basic Rule Modeling* before continuing on this path.

- Path 3

  **Use a commercially available SOAP client to integrate with and test a Decision Service** - In other words, this SOAP client will read a web-services-standard service contract (discussed below), generate a request message from it, send it to the Corticon Server, process it, and then return the response message. Progress Corticon does not include such an application, so the reader must obtain one in order to complete this path.

- Path 4

  **Use a JSON/RESTful client to consume a Decision Service** - A JSON RESTful interface is one that follows the REST architectural style and uses JSON as its data representation format. Using a sample Corticon requests in JavaScript Object Notation (JSON), the client will send the JSON-formatted request message to the Corticon Server, process it, and then return the response message.
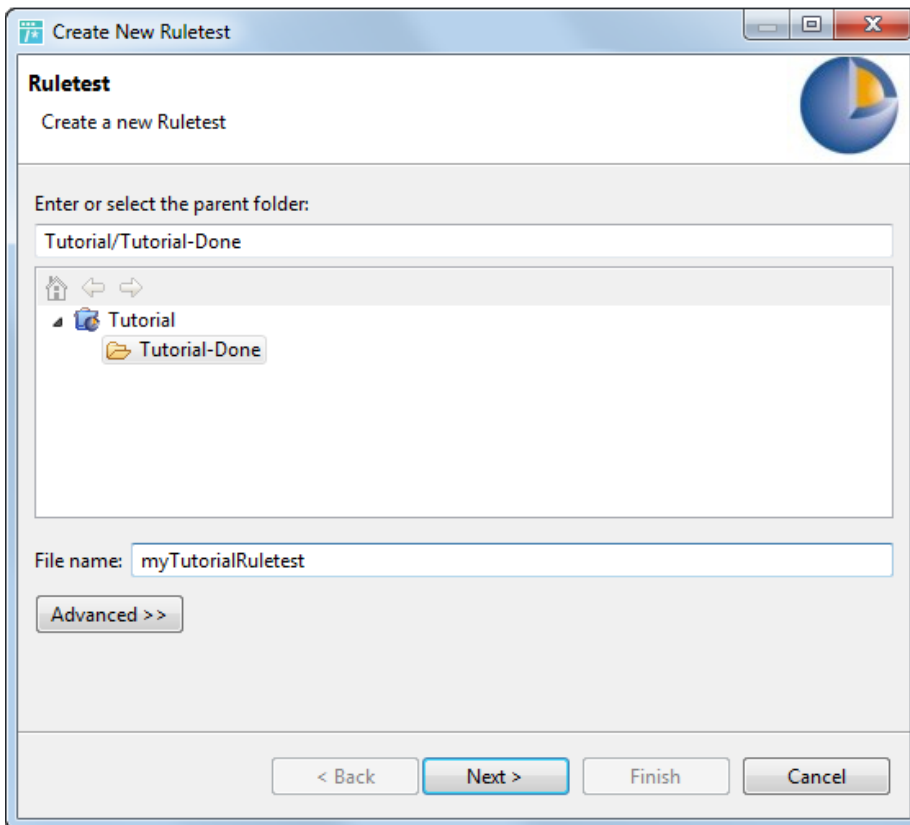
# Path 1: Using Corticon Studio as a SOAP client to consume a Decision Service

In this path, we will use Corticon Studio as a SOAP client to execute Decision Services running on a remote Corticon Server.

## Creating a Java server test in Corticon Studio

**Important:** Even though we are using Corticon Studio to test, we will use its *remote* testing feature, which executes a Decision Service running on a Corticon Server (remotely), not a Ruleflow open in Corticon Studio (locally). To keep this distinction clear, we are **not** going to open `tutorial_example.erf` in Corticon Studio – it is not necessary since we are really testing the Decision Service sample `Cargo` running on a Corticon Java Server.

1. In Corticon Studio, create a new Ruletest by choosing the menu command **File** > **New** > **Ruletest**.



2. Select the **parent folder** for the new Ruletest.

3. Enter a file name for the new Ruletest.

4. Click **Next** to open the **Select Test Subject** panel.

5. Click the **Run against Server** tab to open its panel, as shown:

6. For the **Server URL**, enter a URL; for example, a Java Server installed (and running) on the same machine as the Studio: `http://localhost:8850/axis`. Your entry is validated when you click **Refresh**, and persisted in your Studio. Once you have persisted URLs, click on the right side of the Server URL area to open the dropdown menu to make your selection.

---

**Note:** Only a few Server URLs are persisted this way. If you have a larger list that you want to edit, see "Specifying server URLs for access to test subjects" in the *Quick Reference Guide*.

---

Click **Refresh** to populate the list of deployed Decision Services on that server.

7. Click on an appropriate Decision Service for this Ruletest:

| Name | Major | Minor | Effective Start Date | Effective Stop Date |
|---|---|---|---|---|
| AllocateTrade | 1 | 14 | | |
| Candidates | 1 | 14 | | |
| Cargo | 1 | 0 | | |
| ProcessOrder | 1 | 10 | | |
| | | | | |
| | | | | |

8. You can click **OK** at this point if you do not want to apply the optional overrides.

9. When the selected Decision Service was deployed with a date range defined, it is active from the effective date through the expiration date. You can apply overrides to change the test Decision Service's version or

---

to simulate the Ruletest's call as occurring at a specific point in time. Specify your preferred values -- major version + effective target date -- as illustrated:



**10.** Click **OK**. The dialog closes. The details of the remote server and Decision Service specifications are displayed at the top of the Testsheet:
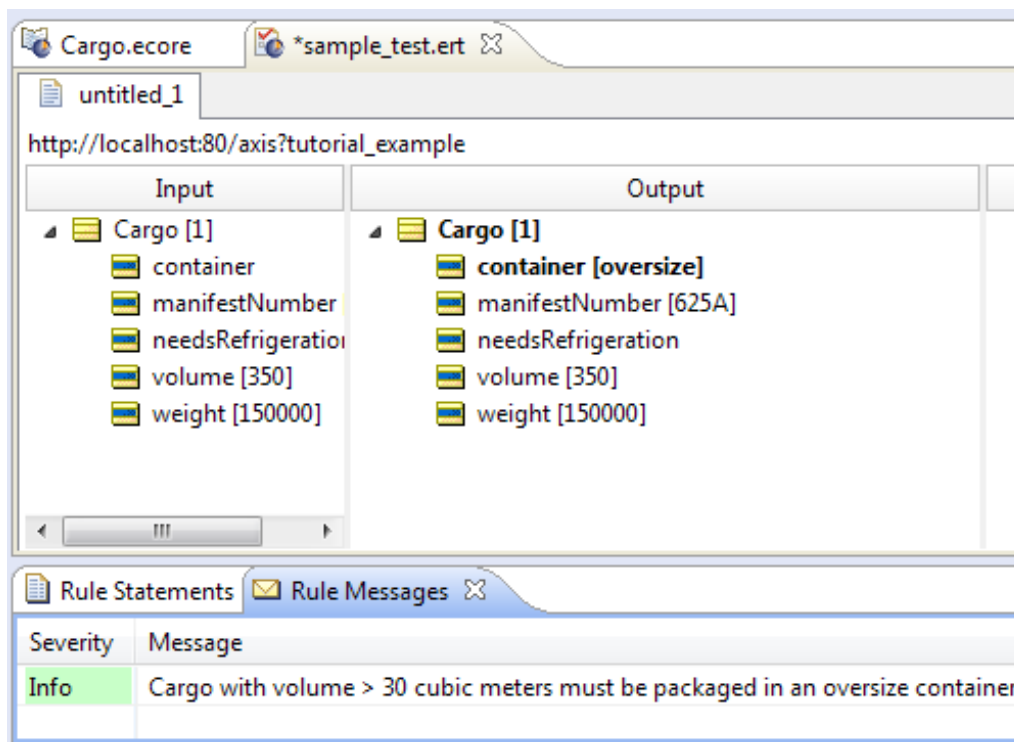


**11.** Run the Ruletest.

The test executes against the specified Decision Service on the selected Java web server using the overrides you entered.

# Executing the remote test

Execute the Test by selecting **Ruletest > Testsheet > Run Test** from the Corticon Studio menubar or ▷ from the toolbar.

We should see an Output pane similar to the following:

**Figure 7: Response from Remote Decision Service**

The Output pane of the Testsheet shown above displays the response message returned by the Corticon Server. This confirms that our Decision Service has processed the data contained in the request and sent back a response containing new data (the `container` attribute and the message).

# Path 2: Using bundled sample code to consume a Decision Service

**Creating a Request Message for a Decision Service**

In this path, we will use a feature of Corticon Studio to generate a request message directly. The steps to accomplish this are:

---
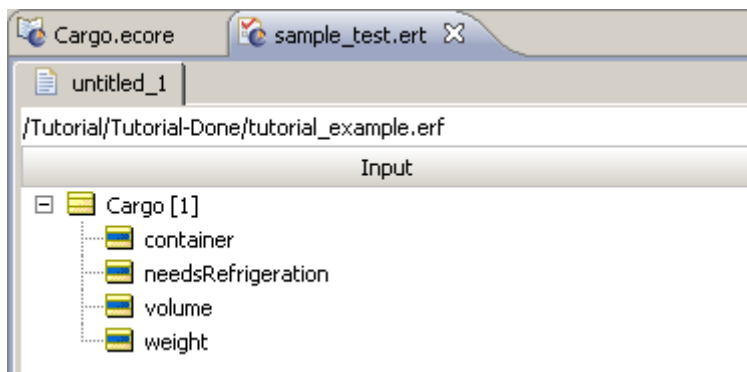
**Note:** This section uses Service Contracts and Work Document Entities. For more on these functions, see *"Service contracts" in the Integration and Deployment Guide.*

---

1. Open Corticon Studio.

2. Open the Ruleflow you have deployed as a Decision Service in a CDD. If you are using the Tutorial example, this is `tutorial_example.erf`.

3. Create a new Ruletest by following the procedure outlined in Option 1 above. For Test Subject, you can choose either your local or remote `tutorial_example.erf`.

4. Create an Input test tree manually as in Option 1 above, or use menubar option **Ruletest>Testsheet>Data>Input>Generate Data Tree**, which produces the structure of a request message in the Input pane. One `Cargo` entity should appear in the Input pane, as shown below:
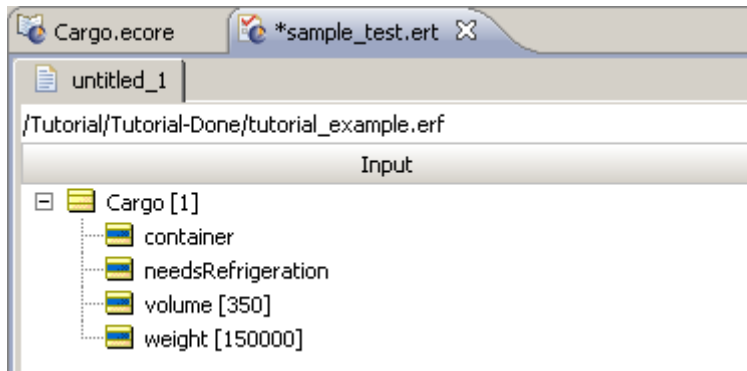
**Figure 8: A New Test**

5. Enter data into the Input Testsheet as you did when testing the Ruleflow in the *Corticon Studio Tutorial: Basic Rule Modeling*. Your Input Testsheet will now look similar to the following:

**Figure 9: Test with Data**



6. Use Corticon Studio's menubar option **Ruletest>Testsheet>Data>Input>Export Request XML** to export this Input pane as an XML document. We will use this exported XML document as the body or "payload" of our request message. Give the export file a name and remember where you save it. We will assign a filename of `sample.xml` for our use case.

7. Open `sample.xml` in any text editor. It should look very similar to the following figure:

**Figure 10: Sample XML File Exported from a Studio Test**
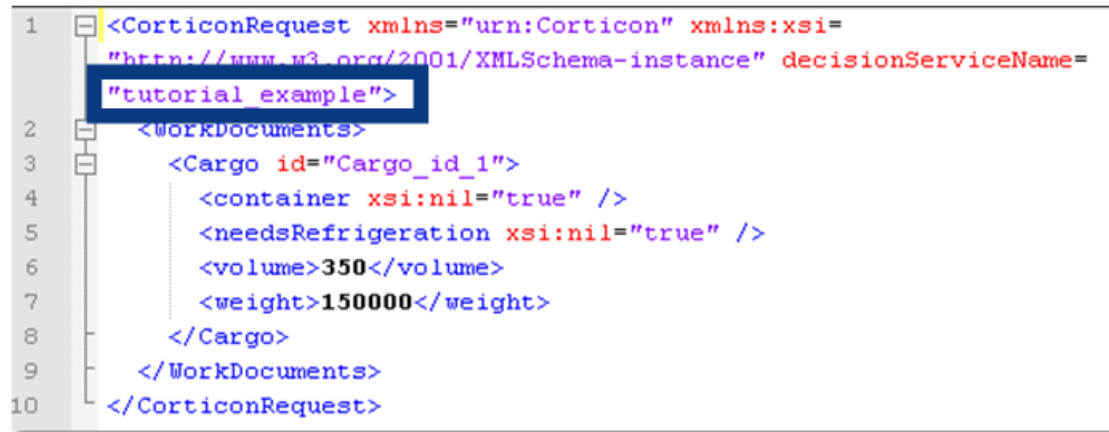


8. Modify `sample.xml` by deleting the `<?XML version="1.0" encoding="UTF-8"?>` tag from the very top (this will be added automatically by the bundled sample code we will use to send this as a request message to the Decision Service). This tag is shown above, enclosed in an **orange box**.

9. Change the `decisionServiceName` attribute value in the `CorticonRequest` element from `InsertDecisionServiceName` to the service name of the Decision Service as it was defined in your deployed `.cdd` file. In our example, this name is `tutorial_example`. This piece is shown in the figures above and below (before and after the changes) enclosed in a **blue box**. Your `sample.xml` file should now look like this:

**Figure 11: Sample XML File with Changes Made**



```
1   <CorticonRequest xmlns="urn:Corticon" xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance" decisionServiceName=
    "tutorial_example">
2     <WorkDocuments>
3       <Cargo id="Cargo_id_1">
4         <container xsi:nil="true" />
5         <needsRefrigeration xsi:nil="true" />
6         <volume>350</volume>
7         <weight>150000</weight>
8       </Cargo>
9     </WorkDocuments>
10  </CorticonRequest>
```

10. **Save** your changes to the XML file and exit your text editor.
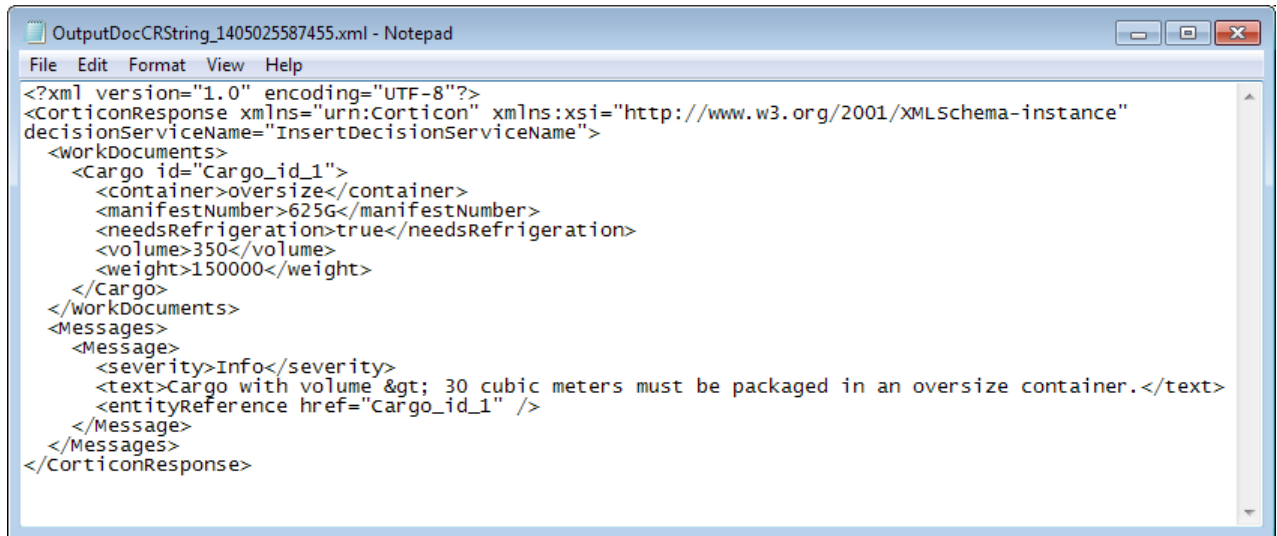
# Sending a request message to Corticon Server

The Corticon Server test scripts, described in Testing the installed Corticon Server on page 20, provides commands that run sample requests.

1. Start Corticon Server.

2. Launch the script `testServerAxis.bat` file, located in `[CORTICON_HOME]\Server\bin`.

3. Enter `130`.

4.  Enter the full path of your test XML file. Our test XML file is `sample.xml`. When you press **Enter** the request is processed and a response is generated.

5.  In the folder `[CORTICON_WORK_DIR]\output`, open the most recent `OutputDoc` file. The response from the Decision Service looks like this:

    **Figure 12: Corticon Response**

    

The response message is exactly what the Corticon Server's output looks like when it is returned to the consuming application. There are several things to note in this response:

*   The `container` attribute has been assigned a value of `Oversize`.

*   The input data `volume` and `weight` have been returned in the response message unchanged. If your rules do not change input data, it will be returned exactly as sent.

*   The Corticon Studio Ruletest assigned unique `id` values to our terms during the XML export. However, if the Server receives a request message *without* `id` values, it will assign them automatically to ensure resulting terms remain associated properly.

*   The `Messages` section of the response has been populated with a posted message. Notice that the contents of the `severity` and `text` tags match those used in the rules.

*   The `entityReference` tag in the `Messages` section uses an `href` to "link" or "point" to the associated element's `id` value. In this case, we see that the posted message links to (or is associated with) `Cargo` with `id` equal to `Cargo_id_1`. In this case, there is only one `Cargo` it *could* link to, but a production request message may contain hundreds or thousands of `Cargo` entities. In those cases, maintaining `href` association between entities and their message(s) is critical.

*   The SOAP "wrapper" or envelope tags were added by the bundled sample code to ensure the request message was sent in accordance with web service standards.

Other details of the Corticon Server response message are described in the *Corticon Server: Integration & Deployment Guide*.

# Path 3: Using SOAP client to consume a Decision Service

### Web Services Service Contracts

Web Services has two main ways of describing a service contract:

**1.** An XML schema document, also known by its file suffix XSD.

**2.** A Web Services Description Language document, or WSDL (often pronounced "wiz-dull").

Many commercial SOAP and web services development tools have the ability to import an XSD or WSDL service contract and generate a compliant request message directly from it. This path assumes you have access to such a tool and want to use it to consume a Decision Service.

The Corticon Deployment Console can produce both XSD and WSDL documents. The *Server Integration & Deployment Guide* contains more information about these documents, including detailed descriptions of their structure and elements. However, if you have chosen this path, we assume you are already familiar enough with service contracts to be able to use them correctly once generated.

## Web services SOAP messaging styles

There are also two types of SOAP messaging styles commonly used in web services:

**1.** RPC-style, which is a simpler, less-capable messaging style generally used to send smaller messages and receive single variable answers. All of the administrative methods in Corticon Server's SOAP API use RPC-style messaging.

**2.** Document-style, which is more complex, but allows for richer content, both in request and response messages. The Corticon Server rule execution (**execute**) interface supports both document-style and RPC-style messaging.

---

**Important:** Any SOAP client or SOAP-capable application used to consume a Decision Service deployed to the Corticon Server typically uses document-style messaging. See the *Integration & Deployment Guide* for complete details on proper structure of a compliant request message.

---

## Creating a Java Server service contract using the Deployment Console

Launch the **Deployment Console**, and then perform the following steps to generate a service contract.

---

> **Note:** The **Deployment Console** options are also described in detail in the *Corticon Server: Integration & Deployment Guide*.

**Figure 13: Deployment Console's Service Contract Specification Window**



1. **Decision Service Level / Vocabulary Level**. These radio buttons determine whether one service contract is generated per listed Ruleflow, or if a single "master" service contract is generated from the entire Vocabulary. A Decision Service-level service contract is usable only for a specific Decision Service, whereas a Vocabulary-level service contract can be used for all Decision Services that were built using that Vocabulary. Choose the option that is most compatible with your SOAP tool.

2. **Vocabulary File**. If generating a Vocabulary-level service contract, enter the Vocabulary file name (`.ecore`) here. If generating a Decision Service-level contract, this field is read-only and shows the Vocabulary associated with the currently highlighted Ruleflow row above.

3. **Type**. This is the service contract type: WSDL, XML Schema, or none. Note, that the **Generate Service Contracts** button will be enabled only when Type is set to either WSDL or XML Schema.

4. **XML Messaging Style**. Describes the message style, flat or hierarchical, in which the WSDL will be structured.

5. **SOAP Server URL**. URL for the SOAP node that is bound to the Corticon Server. Enabled for WSDL service contracts only. The default URL http://localhost:8850/axis/services/Corticon makes a Decision Service available to the default Corticon Server installation performed earlier. Note: this URL can be changed and additional URLs can be added to the drop-down list. See see "Designer properties and settings" in *Server Integration & Deployment Guide* for details.

6. **Generate Service Contracts**. Use this button to generate either the WSDL or XML Schema service contracts into the output directory. If you select Decision Service-level contracts, one service contract per Ruleflow listed at top will be created. If you select Vocabulary-level, only one contract is created per Vocabulary file.

## Creating a SOAP request message for a Decision Service

Once your SOAP development tool has imported the WSDL or XSD service contract, it should be able to generate an instance of a request message that complies with the service contract. It should also provide you with a way of entering sample data to be included in the request message when it is sent to the Decision Service.

**Important:**

Most commercial SOAP development tools accurately read service contracts generated by the Deployment Console, ensuring well-formed request messages are composed.

One occasional problem, however, involves the Decision Service Name, which was entered in field 3 of the Deployment Console's Deployment Descriptor section. Even though all service contracts list `decisionServiceName` as a mandatory element, many SOAP tools do not automatically insert the Decision Service Name attribute into the request message's `decisionServiceName` element. Be sure to check this before sending the request message. If the request message is sent without a `decisionServiceName`, the Server will not know which Decision Service is being requested, and will return an error message.

Corticon Server also offers a mode of WSDL generation that's more compatible with Microsoft's Windows Communications Framework. See the *Server Integration & Deployment Guide* for more details.

Enter all required data into the request message. The `tutorial_example.erf` example produces its best results when you enter positive integer values such as:

- `cargo.weight 200000`

- `cargo.volume 1000`

## Sending a SOAP request message to Corticon Server

Make sure the application server is running and your Deployment Descriptor file is in the correct location as described earlier. Now, use your SOAP tool to send the request message to the Corticon Server.

Your SOAP tool should display the response from the Corticon Server. Are the results what you expected? If not, or if the response contains an error, proceed to the Troubleshooting topic.

# Path 4: Using JSON/RESTful client to consume a Decision Service on Java Server

You can create Corticon requests in JavaScript Object Notation (JSON), a text format that you can use as an alternative to XML. A JSON RESTful interface is one that follows the REST architectural style and uses JSON as its data representation format. Specifically, a standardized `JSONObject` with name-value pairs of `"Objects":<JSONArray>`can be passed in to Corticon Server's `ICcServer.execute(…)` to process the request and return a JSON-formatted reply.

## Running the sample JSON Request

A Corticon Server installation provides a JSON sample and a test script that runs the sample.

---

**Note:** JSON datatypes require double-quotes only on String values. As JSON does not have a date datatype, Corticon treats time values as Strings. When the other datatypes -- Booleans and numbers -- are in quotes, Corticon interprets the datatype from its Vocabulary properties and removes the quotes. Output will conform to the JSON syntax.

---

The sample, located at `[CORTICON_WORK_DIR]\Samples\Rule Projects\OrderProcessing\OrderProcessingPayload.json`, is as follows:

```
{"name": "ProcessOrder",
 "Objects": [{
     "total": null,
     "shipped": null,
     "item": [
         {
             "product": "Ball",
             "price": "10.000000",
             "quantity": "20",
             "subtotal": null,
             "__metadata": {
                 "#id": "Item_id_7",
                 "#type": "Item"
             }
         },
         {
             "product": "Racket",
             "price": "20.000000",
             "quantity": "1",
             "subtotal": null,
             "__metadata": {
                 "#id": "Item_id_8",
                 "#type": "Item"
             }
         },
         {
             "product": "Wrist Band",
             "price": "5.250000",
             "quantity": "2",
             "subtotal": null,
             "__metadata": {
                 "#id": "Item_id_9",
                 "#type": "Item"
             }
         }
     ],
     "shippedOn": null,
     "__metadata": {
         "#id": "Order_id_1",
         "#type": "Order"
     },
     "dueDate": "1/1/2008",
     "note": null
 },
 {
     "total": null,
     "shipped": null,
     "item": [
         {
             "product": "Pencil",
             "price": "1.000000",
             "quantity": "10",
             "subtotal": null,
             "__metadata": {
                 "#id": "Item_id_10",
                 "#type": "Item"
             }
         },
```

```
                    {
                        "product": "Notebook",
                        "price": "3.000000",
                        "quantity": "2",
                        "subtotal": null,
                        "__metadata": {
                            "#id": "Item_id_11",
                            "#type": "Item"
                        }
                    },
                    {
                        "product": "Eraser",
                        "price": "2.000000",
                        "quantity": "1",
                        "subtotal": null,
                        "__metadata": {
                            "#id": "Item_id_12",
                            "#type": "Item"
                        }
                    }
                ],
                "shippedOn": null,
                "__metadata": {
                    "#id": "Order_id_2",
                    "#type": "Order"
                },
                "dueDate": "1/1/2009",
                "note": null
            }
      ]}
```

**Note:** The sample sets the Decision Service name as a parameter is the JSON request. You could add other parameters such as the version and effective timestamp, as shown:

```
{
                "name": "string",
                "majorVersion": "string",
                "minorVersion": "string",
                "effectiveTimestamp": "string",
                "Objects": [
```

**To run the JSON sample:**

1. Start Corticon Server.

2. Open a command prompt window at `[CORTICON_HOME]\Server\bin`.

3. Enter `testServerREST.bat`

4. Enter `142`.

5. When prompted for **Input JSON File Path**, enter (or copy) the path to the sample:

```
C:\Users\{user}\Progress\CorticonWork_5.7\Samples\Rule
Projects\OrderProcessing\OrderProcessingPayload.json
```

The request is processed, and its output is placed at `[CORTICON_WORK_DIR]\output` with a name formatted as `OutputCRString_{epochTime}.json` where `{epochTime}` is the number of seconds that have elapsed since 1/1/1970. The input file is also placed there. The output for the sample is as follows:

```
                {
        "Messages": {
```

```
        "Message": [
            {
                "entityReference": "Order_id_2",
                "text": "The total for the Order is 18.000000.",
                "severity": "Info",
                "__metadata": {"#type": "#RuleMessage"}
            },
            {
                "entityReference": "Order_id_1",
                "text": "The total for the Order is 230.500000.",
                "severity": "Info",
                "__metadata": {"#type": "#RuleMessage"}
            },
            {
                "entityReference": "Order_id_1",
                "text": "This Order was shipped late. Ship date 12/1/2008",
                "severity": "Warning",
                "__metadata": {"#type": "#RuleMessage"}
            },
            {
                "entityReference": "Order_id_2",
                "text": "This Order was shipped on time. Ship date 12/1/2008",
                "severity": "Info",
                "__metadata": {"#type": "#RuleMessage"}
            }
        ],
        "__metadata": {"#type": "#RuleMessages"},
        "version": "1.10"
    },
    "Objects": [
        {
            "total": 230.5,
            "shipped": true,
            "item": [
                {
                    "product": "Ball",
                    "price": "10.000000",
                    "quantity": "20",
                    "__metadata": {
                        "#type": "Item",
                        "#id": "Item_id_7"
                    },
                    "subtotal": 200
                },
                {
                    "product": "Racket",
                    "price": "20.000000",
                    "quantity": "1",
                    "__metadata": {
                        "#type": "Item",
                        "#id": "Item_id_8"
                    },
                    "subtotal": 20
                },
                {
                    "product": "Wrist Band",
                    "price": "5.250000",
                    "quantity": "2",
                    "__metadata": {
                        "#type": "Item",
                        "#id": "Item_id_9"
                    },
                    "subtotal": 10.5
                }
            ],
            "shippedOn": "12/1/2008",
            "__metadata": {
                "#type": "Order",
                "#id": "Order_id_1"
```

```
                    },
                    "dueDate": "1/1/2008",
                    "note": "This Order was shipped late"
                },
                {
                    "total": 18,
                    "shipped": true,
                    "item": [
                        {
                            "product": "Pencil",
                            "price": "1.000000",
                            "quantity": "10",
                            "__metadata": {
                                "#type": "Item",
                                "#id": "Item_id_10"
                            },
                            "subtotal": 10
                        },
                        {
                            "product": "Notebook",
                            "price": "3.000000",
                            "quantity": "2",
                            "__metadata": {
                                "#type": "Item",
                                "#id": "Item_id_11"
                            },
                            "subtotal": 6
                        },
                        {
                            "product": "Eraser",
                            "price": "2.000000",
                            "quantity": "1",
                            "__metadata": {
                                "#type": "Item",
                                "#id": "Item_id_12"
                            },
                            "subtotal": 2
                        }
                    ],
                    "shippedOn": "12/1/2008",
                    "__metadata": {
                        "#type": "Order",
                        "#id": "Order_id_2"
                    },
                    "dueDate": "1/1/2009",
                    "note": "This Order was shipped on time"
                }
            ],
            "name": "ProcessOrder"
        }
```

# Troubleshooting Java server

When the default port settings conflict with the default port setting of Corticon's Progress Application Server (PAS) installation (`8850` for HTTP and `8851` for HTTPS), you should consult with your system administrator to identify alternate ports you might use. Changes to the HTTP port must also be set in the `brms.properties` file located at the server installation's `[CORTICON_WORK_DIR]` root. Override the property value for `com.corticon.deployment.soapbindingurl_1=http://localhost:8850/axis` to set your preferred HTTP port value.

---

**Note:  URLs for the Web Service-enabled instances of the Corticon Server** - Enter the preferred URL value in the `SOAP Server URL` field in the Deployment Console window's Service Contract Specification section. You can add additional sequential integer values for `com.corticon.deployment.soapbindingurl_`, such as:

- `com.corticon.deployment.soapbindingurl_2=http://localhost:9080/axis/services/Corticon` (typically used by IBM WebSphere)

- `com.corticon.deployment.soapbindingurl_3=http://localhost:7001/axis/services/Corticon` (typically used by Oracle/BEA Weblogic)

---

**Note:** When you have problems on the Java Server, refer to *"Using Corticon Server logs" and "Troubleshooting" topics in the Integration and Deployment Guide*.

---

# 7

# Summary of Java samples

Corticon Server for Java contains several sample code files that are useful starting points when building your own integrations. The sample code files are self-documented. They are installed in your Corticon Java Server installation's `[CORTICON_WORK_DIR]\Samples\Clients` directory.

| Subdirectory | Sample Code file | Usage |
|---|---|---|
| SOAP | `CcServerApiTest.java` | Provides the source code used by `testServer.bat` to create a command interface for SOAP on the Server API.<br><br>**Note:** Uses Corticon Server in the same JVM as the Java test code and cannot be executed against Corticon Server inside the Progress Application Server (you can, of course, write your own Java client using these APIs to operate against any deployment). |

| Subdirectory | Sample Code file | Usage |
|---|---|---|
| Deploy | CcDeployApiTest.java | Shows how to control the Deployment Console through Java API calls.  This is the source code used by testDeployConsole.bat to create a Windows console interface for Server API. |
| REST | CcServerRestTest.java | Provides the source code used by testServerRest.bat to create a command interface for JSON/RESTful services on the Server API.<br><br>**Note:** Uses Corticon Server in the same JVM as the Java test code and cannot be executed against Corticon Server inside the Progress Application Server (you can, of course, write your own Java client using these APIs to operate against any deployment). |

# A

# Corticon 5.7.2 Online Tutorials and Documentation

**TUTORIALS:** Learn about Corticon from online lessons at the Corticon Learning Center.

| DOCUMENTATION: About this release | |
|---|---|
| *What's New in Corticon* Online Help PDF | Describes the enhancements and changes to the product since its last point release. |
| *Corticon Installation Guide* Online Help PDF | Step-by-step procedures for installing Corticon Studio and Servers in this release on Windows and Linux platforms. |

| DEVELOPMENT DOCUMENTATION: Define and Model Business Rules | |
|---|---|
| *Rule Modeling Guide* Online Help PDF | Introduces how business rules are modeled in Corticon Studio including the creation of Vocabularies, Rulesheets, Ruleflows, and Ruletests. This is the starting point for new rule modelers. |
| *Quick Reference Guide* Online Help PDF | Reference guide to the Corticon Studio user interface, including descriptions of all menu options, buttons, and actions. |
| *Rule Language Guide* Online Help PDF | Reference information for all operators available in the Corticon Studio Vocabulary. |

| | |
|---|---|
| *Extensions Guide* Online Help PDF | Detailed technical information about the Corticon extension framework for extended operators and service callouts. |
| *Javadoc for Extensions API* Javadoc | Complete Java API reference for Corticon Extensions. |

| | |
|---|---|
| **DEPLOYMENT DOCUMENTATION: Run Decision Services on Servers** | |
| *Integration and Deployment Guide* Online Help PDF | Provides details on deploying and managing Corticon Decision Services. Start here to learn about the options for deploying Corticon, the best practices for managing your deployment, and the APIs for integrating Corticon with clients executing rules. |
| *Data Integration Guide* Online Help PDF | Provides details on how to integrate Corticon with external data sources and how to use Corticon for batch rule processing. Includes details on Corticon EDC and ADC features for accessing data from Decision Services. |
| *Web Console Guide* Online Help PDF | Presents the Corticon Web Console which can be used to manage your Corticon deployment. This GUI interface simplifies the management and monitoring of Corticon servers and Decision Services. |
| *Deploying Web Services with Java* Online Help PDF | Provides details on deploying Corticon as a REST or SOAP web service on Java application servers. |
| *Deploying Web Services with .NET* Online Help PDF | Provides details on deploying Corticon as a REST or SOAP web service on Microsoft IIS. |
| *Javadoc for Corticon Server API* Javadoc | Complete Java API reference for Corticon Server. |