PROGRESS® Corticon®

Corticon Server:
Deploying Web Services with Java

PROGRESS

Notices

Copyright agreement

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

Table of Contents

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. After installation on a supported Windows platform, that server installation's deployment artifacts can be redeployed on various UNIX and Linux web service platforms. See the Progress Software web page Progress Corticon 5.5.2 - Supported Platforms Matrix for more information.
- 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 has its own installer and documentation. 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 tutorial 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

For this tutorial, 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 5.5.2 - Supported Platforms Matrix for 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.

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:

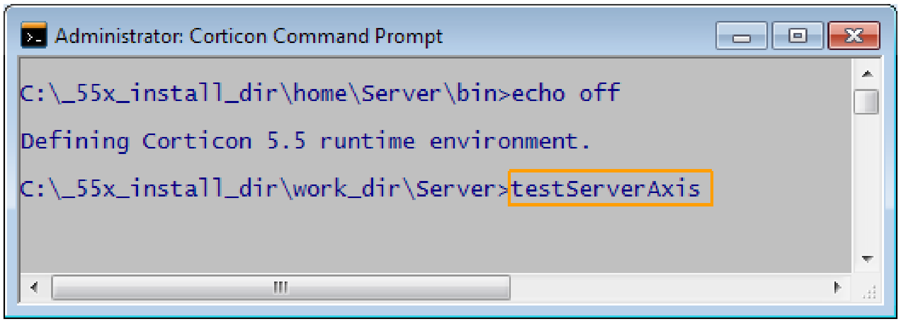
```
[CORTICON_WORK_DIR]\Server\pas\server\webapps
```

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.

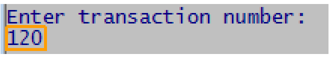
To verify that Corticon Server is installed correctly on Progress Application Server:

1. Start Corticon Server from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.5** > **Start Corticon Server**
2. Open a Corticon Command Prompt from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.5** > **Corticon Command Prompt**
3. At the command prompt, enter `testServerAxis`, as shown:



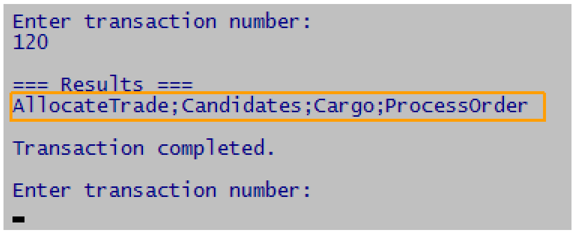
```
C:\_55x_install_dir\home\Server\bin>echo off

Defining Corticon 5.5 runtime environment.

C:\_55x_install_dir\work_dir\Server>testServerAxis
```

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



```
Enter transaction number:
120
```

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



```
Enter transaction number:
120

=== Results ===
AllocateTrade;Candidates;Cargo;ProcessOrder

Transaction completed.

Enter transaction number:
```

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 25

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

Enabling HTTPS

Enabling HTTPS client connections on Corticon Server for Java

Corticon Server supports Secure Socket Layer (SSL)-enabled communications between the Web server and a Web service client. If you attempt to use the default HTTPS port, `8851` (for example, connecting from the Server Console, you get a security message indicating that your connection is not private. If you want to use HTTPS, you must enable the HTTPS (SSL) connections.

Note: The following procedure pertains to the security of communication between the client application and the Server. To enable SSL communication between the Server and the client, you must obtain and install public key certificates for the Server host machine and complete separate configuration procedures for each deployed Client service and for the Server.

To enable SSL on Corticon Server for Java:

1. Obtain a private key and a Web server digital certificate.
2. Install the Web server digital certificate in the Web server.
3. Start the Corticon Server. When startup is complete, stop it. The initial startup creates the `web.xml` file.
4. Edit the file `web.xml` located at `[CORTICON_WORK_DIR]\pas\server\webapps\axis\WEB-INF\` to uncomment the following section:

```
<!-- Uncomment this for HTTPS support
<security-constraint>
  <web-resource-collection>
      <web-resource-name>Corticon Server</web-resource-name>
      <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
```

```
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  -->
```

5. Save the file.

When you restart the Corticon Server, HTTPS is enabled on its default port, `8851`.

Enabling the Corticon Studio to publish to a secure Corticon Server

Corticon Studio supports Secure Socket Layer (SSL)-enabled communications to a Corticon Server. To enable SSL communication between the Server and the Client, you must obtain and install public key certificates for the Corticon Studio. The public certificate then needs to be imported to the Java keystore for the Corticon Studio.

5

Corticon Java Server files and API tools

Corticon Server for deploying web services with Java facilitates deployment on supported Windows operating systems. This guide points out features that enable a walkthrough tutorial experience of 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.

So let's continue with the tutorial. First we'll deploy a Ruleflow to the Java server as a Decision Service, then we'll try out consuming that Decision Service with various manual, SOAP/XML, and JSON/RESTful techniques.

But before exploring these features, you should be aware of some of the .NET server files and API tools.

For details, see the following topics:

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 Java Server Console on supported UNIX/Linux platforms and brands of Application Servers, refer to the Progress Software web page Progress Corticon 5.5.2 - 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_5.5_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 5.5.2 - 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_5.5_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 5.5.2 - Supported Platforms Matrix for 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 24)
   - 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.

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 5.5`, 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 5.5`, 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. For more information, see *Server properties* described in the *Integration and Deployment Guide*.

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_5.5\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.

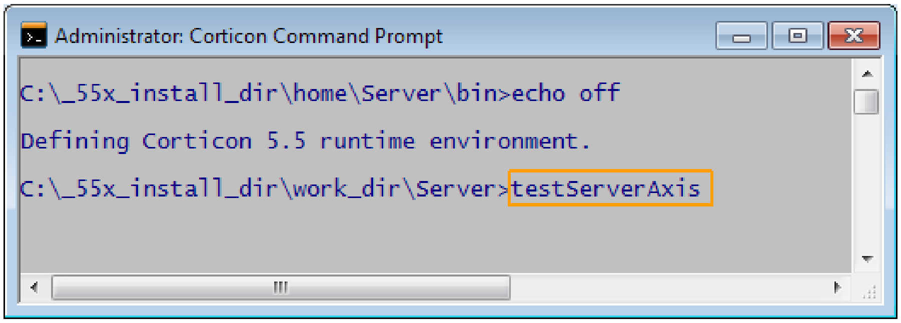
  Note: The bundled Progress Application Server uses `localhost` as the application server's location, and defaults to port settings of `8850` for HTTP and `8851` for HTTPS.

- 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:

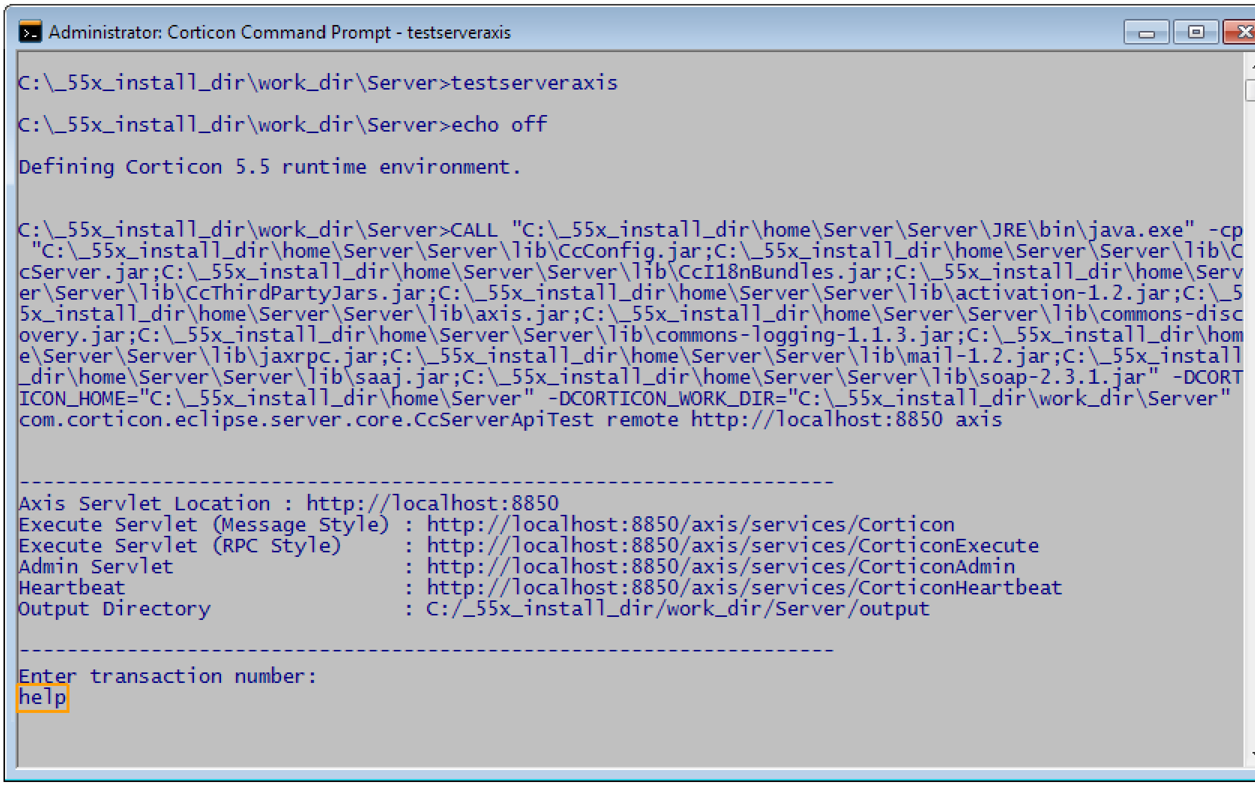
1. Open a Corticon Command Prompt from the Windows **Start** menu by choosing **All Programs** > **Progress** > **Corticon 5.5** > **Corticon Command Prompt**
2. At the command prompt, enter `testServerAxis`, as shown:



```
C:\_55x_install_dir\home\Server\bin>echo off

Defining Corticon 5.5 runtime environment.

C:\_55x_install_dir\work_dir\Server>testServerAxis
```

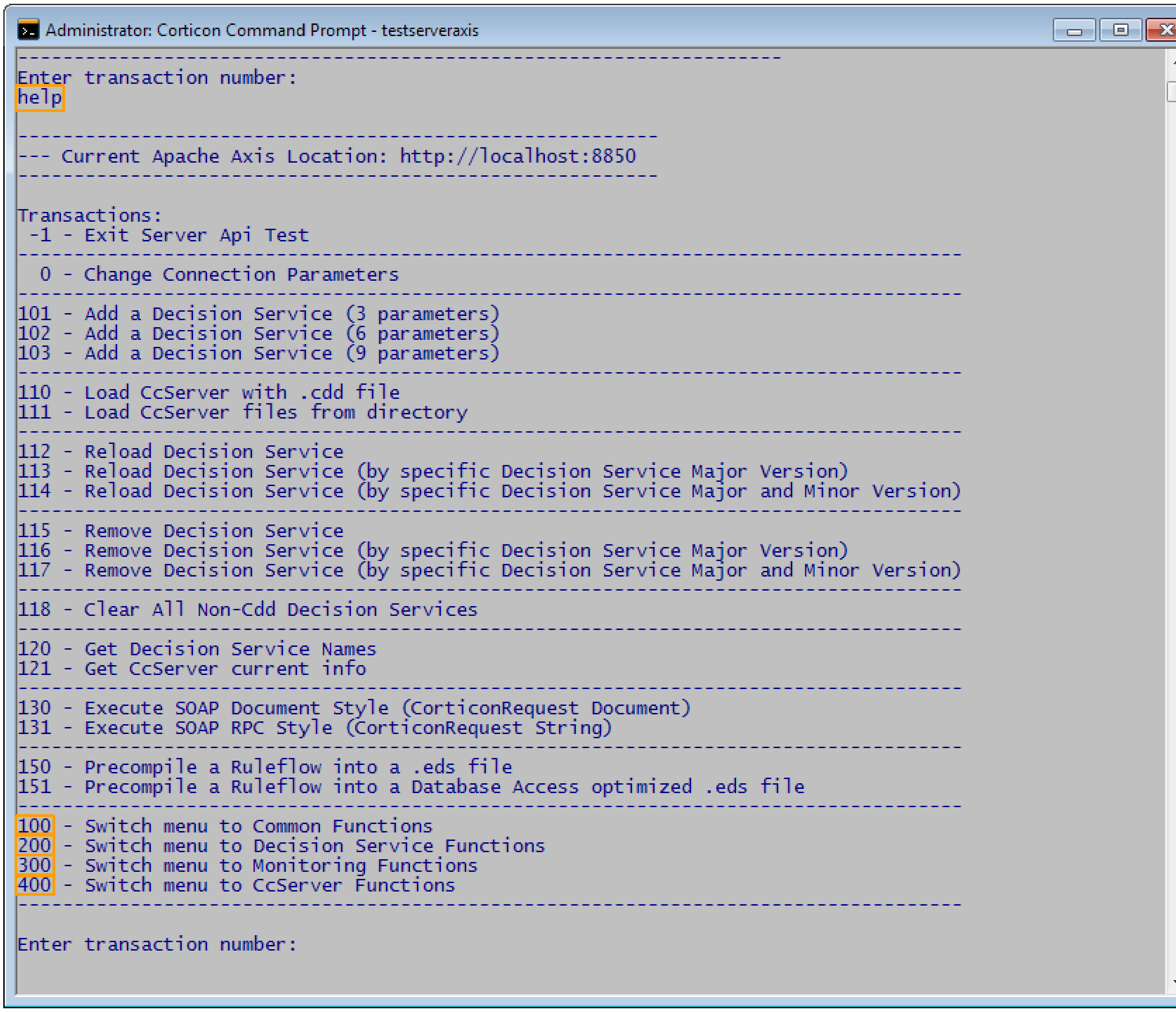
Note: When you launch Corticon Command Prompt, it calls `corticon_env.bat`, adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts by name from several locations -- `\bin`, `\Server\bin`, `\Server\pas\bin`, `\Studio\bin`, and, `\Studio\eclipse` -- and then relocates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

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



```
C:\_55x_install_dir\work_dir\Server>testserveraxis

C:\_55x_install_dir\work_dir\Server>echo off

Defining Corticon 5.5 runtime environment.


C:\_55x_install_dir\work_dir\Server>CALL "C:\_55x_install_dir\home\Server\Server\JRE\bin\java.exe" -cp
 "C:\_55x_install_dir\home\Server\Server\lib\CcConfig.jar;C:\_55x_install_dir\home\Server\Server\lib\C
cServer.jar;C:\_55x_install_dir\home\Server\Server\lib\CcI18nBundles.jar;C:\_55x_install_dir\home\Serv
er\Server\lib\CcThirdPartyJars.jar;C:\_55x_install_dir\home\Server\Server\lib\activation-1.2.jar;C:\_5
5x_install_dir\home\Server\Server\lib\axis.jar;C:\_55x_install_dir\home\Server\Server\lib\commons-disc
overy.jar;C:\_55x_install_dir\home\Server\Server\lib\commons-logging-1.1.3.jar;C:\_55x_install_dir\hom
e\Server\Server\lib\jaxrpc.jar;C:\_55x_install_dir\home\Server\Server\lib\mail-1.2.jar;C:\_55x_install
_dir\home\Server\Server\lib\saaj.jar;C:\_55x_install_dir\home\Server\Server\lib\soap-2.3.1.jar" -DCORT
ICON_HOME="C:\_55x_install_dir\home\Server" -DCORTICON_WORK_DIR="C:\_55x_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:/_55x_install_dir/work_dir/Server/output

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

4. The 100 series commands are listed.



```
--------------------------------------------------------------------
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

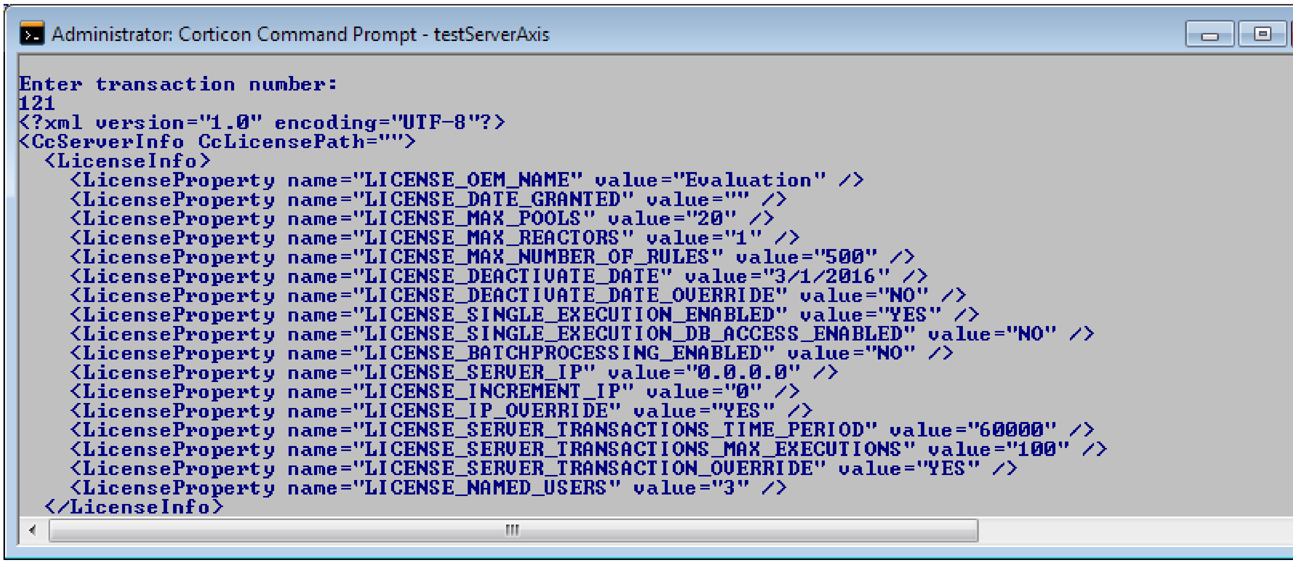
Since we have not deployed any Ruleflows yet, we will use an administrative method to test if Corticon Server is correctly installed as a SOAP Servlet inside our web server.

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:

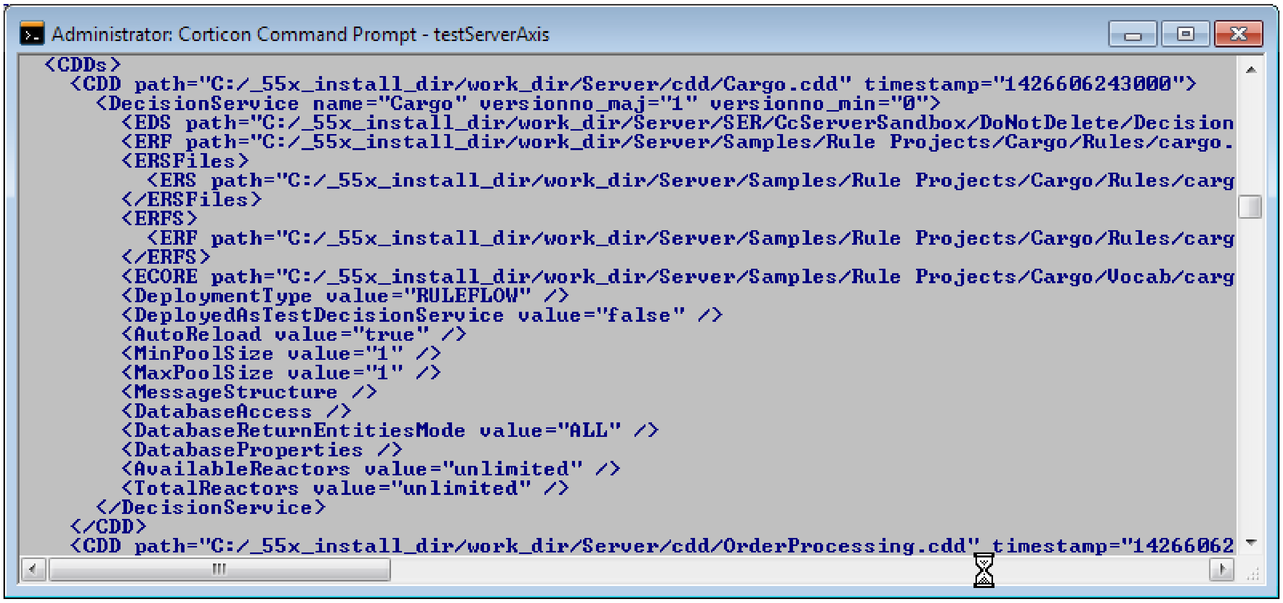
Figure 1: `testServerAxis` Response to command 121: License information



Administrator: Corticon Command Prompt - testServerAxis

```
Enter transaction number:
121
<?xml version="1.0" encoding="UTF-8"?>
<CcServerInfo CcLicensePath="">
  <LicenseInfo>
    <LicenseProperty name="LICENSE_OEM_NAME" value="Evaluation" />
    <LicenseProperty name="LICENSE_DATE_GRANTED" value="" />
    <LicenseProperty name="LICENSE_MAX_POOLS" value="20" />
    <LicenseProperty name="LICENSE_MAX_REACTORS" value="1" />
    <LicenseProperty name="LICENSE_MAX_NUMBER_OF_RULES" value="500" />
    <LicenseProperty name="LICENSE_DEACTIVATE_DATE" value="3/1/2016" />
    <LicenseProperty name="LICENSE_DEACTIVATE_DATE_OVERRIDE" value="NO" />
    <LicenseProperty name="LICENSE_SINGLE_EXECUTION_ENABLED" value="YES" />
    <LicenseProperty name="LICENSE_SINGLE_EXECUTION_DB_ACCESS_ENABLED" value="NO" />
    <LicenseProperty name="LICENSE_BATCHPROCESSING_ENABLED" value="NO" />
    <LicenseProperty name="LICENSE_SERVER_IP" value="0.0.0.0" />
    <LicenseProperty name="LICENSE_INCREMENT_IP" value="0" />
    <LicenseProperty name="LICENSE_IP_OVERRIDE" value="YES" />
    <LicenseProperty name="LICENSE_SERVER_TRANSACTIONS_TIME_PERIOD" value="60000" />
    <LicenseProperty name="LICENSE_SERVER_TRANSACTIONS_MAX_EXECUTIONS" value="100" />
    <LicenseProperty name="LICENSE_SERVER_TRANSACTION_OVERRIDE" value="YES" />
    <LicenseProperty name="LICENSE_NAMED_USERS" value="3" />
  </LicenseInfo>
```

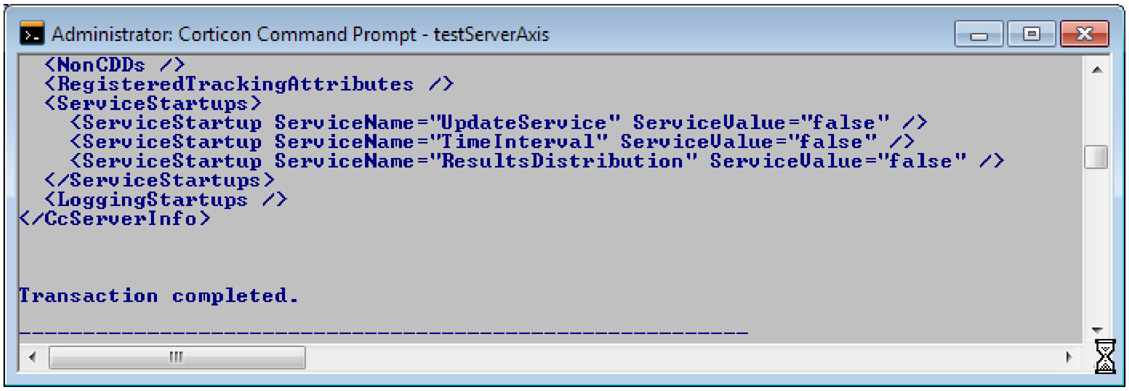
Figure 2: `testServerAxis` Response to command 121: Info on one of the deployed Decision Services



Administrator: Corticon Command Prompt - testServerAxis

```
  <CDDs>
    <CDD path="C:/_55x_install_dir/work_dir/Server/cdd/Cargo.cdd" timestamp="1426606243000">
      <DecisionService name="Cargo" versionno_maj="1" versionno_min="0">
        <EDS path="C:/_55x_install_dir/work_dir/Server/SER/CcServerSandbox/DoNotDelete/Decision
        <ERF path="C:/_55x_install_dir/work_dir/Server/Samples/Rule Projects/Cargo/Rules/cargo.
        <ERSFiles>
          <ERS path="C:/_55x_install_dir/work_dir/Server/Samples/Rule Projects/Cargo/Rules/carg
        </ERSFiles>
        <ERFS>
          <ERF path="C:/_55x_install_dir/work_dir/Server/Samples/Rule Projects/Cargo/Rules/carg
        </ERFS>
        <ECORE path="C:/_55x_install_dir/work_dir/Server/Samples/Rule Projects/Cargo/Vocab/carg
        <DeploymentType value="RULEFLOW" />
        <DeployedAsTestDecisionService value="false" />
        <AutoReload value="true" />
        <MinPoolSize value="1" />
        <MaxPoolSize value="1" />
        <MessageStructure />
        <DatabaseAccess />
        <DatabaseReturnEntitiesMode value="ALL" />
        <DatabaseProperties />
        <AvailableReactors value="unlimited" />
        <TotalReactors value="unlimited" />
      </DecisionService>
    </CDD>
    <CDD path="C:/_55x_install_dir/work_dir/Server/cdd/OrderProcessing.cdd" timestamp="14266062
```

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



```
  <NonCDDs />
  <RegisteredTrackingAttributes />
  <ServiceStartups>
    <ServiceStartup ServiceName="UpdateService" ServiceValue="false" />
    <ServiceStartup ServiceName="TimeInterval" ServiceValue="false" />
    <ServiceStartup ServiceName="ResultsDistribution" ServiceValue="false" />
  </ServiceStartups>
  <LoggingStartups />
</CcServerInfo>


Transaction completed.

---------------------------------------------------------
```

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.

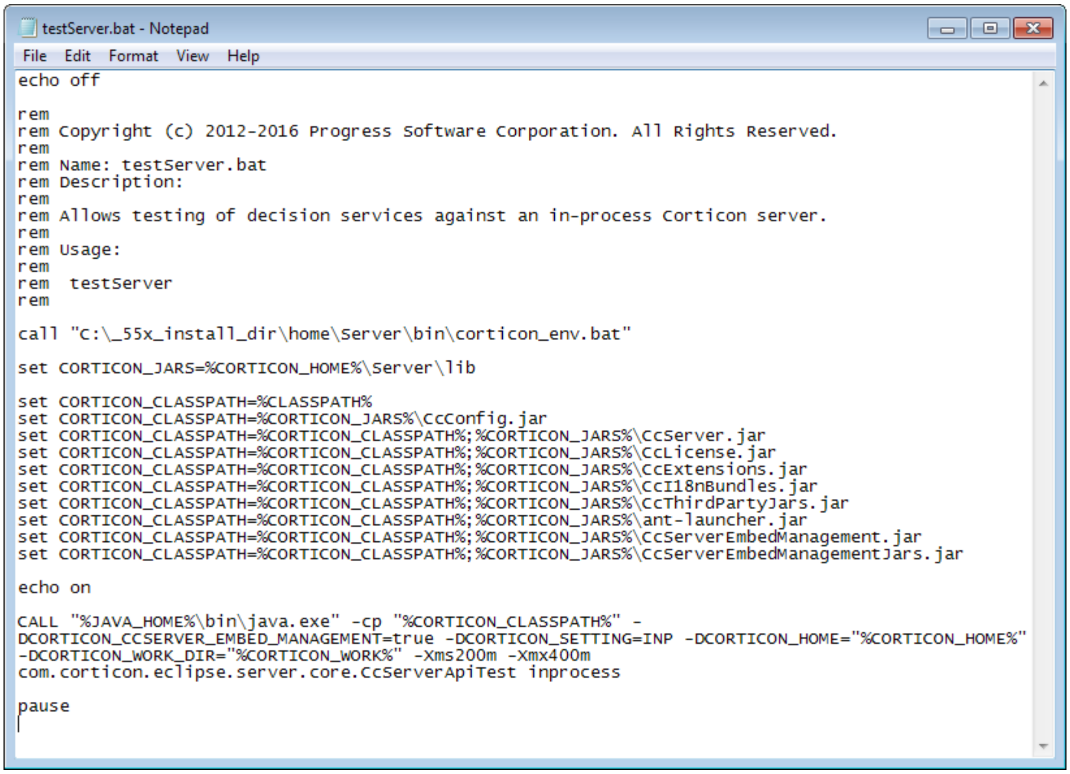
Testing the installed Corticon Server as in-process Java classes

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



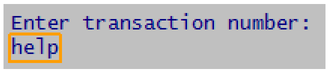
```
echo off

rem
rem Copyright (c) 2012-2016 Progress Software Corporation. All Rights Reserved.
rem
rem Name: testServer.bat
rem Description:
rem
rem Allows testing of decision services against an in-process Corticon server.
rem
rem Usage:
rem
rem  testServer
rem

call "C:\_55x_install_dir\home\Server\bin\corticon_env.bat"

set CORTICON_JARS=%CORTICON_HOME%\Server\lib

set CORTICON_CLASSPATH=%CLASSPATH%
set CORTICON_CLASSPATH=%CORTICON_JARS%\CcConfig.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcServer.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcLicense.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcExtensions.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcI18nBundles.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcThirdPartyJars.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\ant-launcher.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcServerEmbedManagement.jar
set CORTICON_CLASSPATH=%CORTICON_CLASSPATH%;%CORTICON_JARS%\CcServerEmbedManagementJars.jar

echo on

CALL "%JAVA_HOME%\bin\java.exe" -cp "%CORTICON_CLASSPATH%" -
DCORTICON_CCSERVER_EMBED_MANAGEMENT=true -DCORTICON_SETTING=INP -DCORTICON_HOME="%CORTICON_HOME%"
-DCORTICON_WORK_DIR="%CORTICON_WORK%" -Xms200m -Xmx400m
com.corticon.eclipse.server.core.CcServerApiTest inprocess

pause
```

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 5.5** > **Corticon Command Prompt**
2. At the command prompt, enter `testServer`.

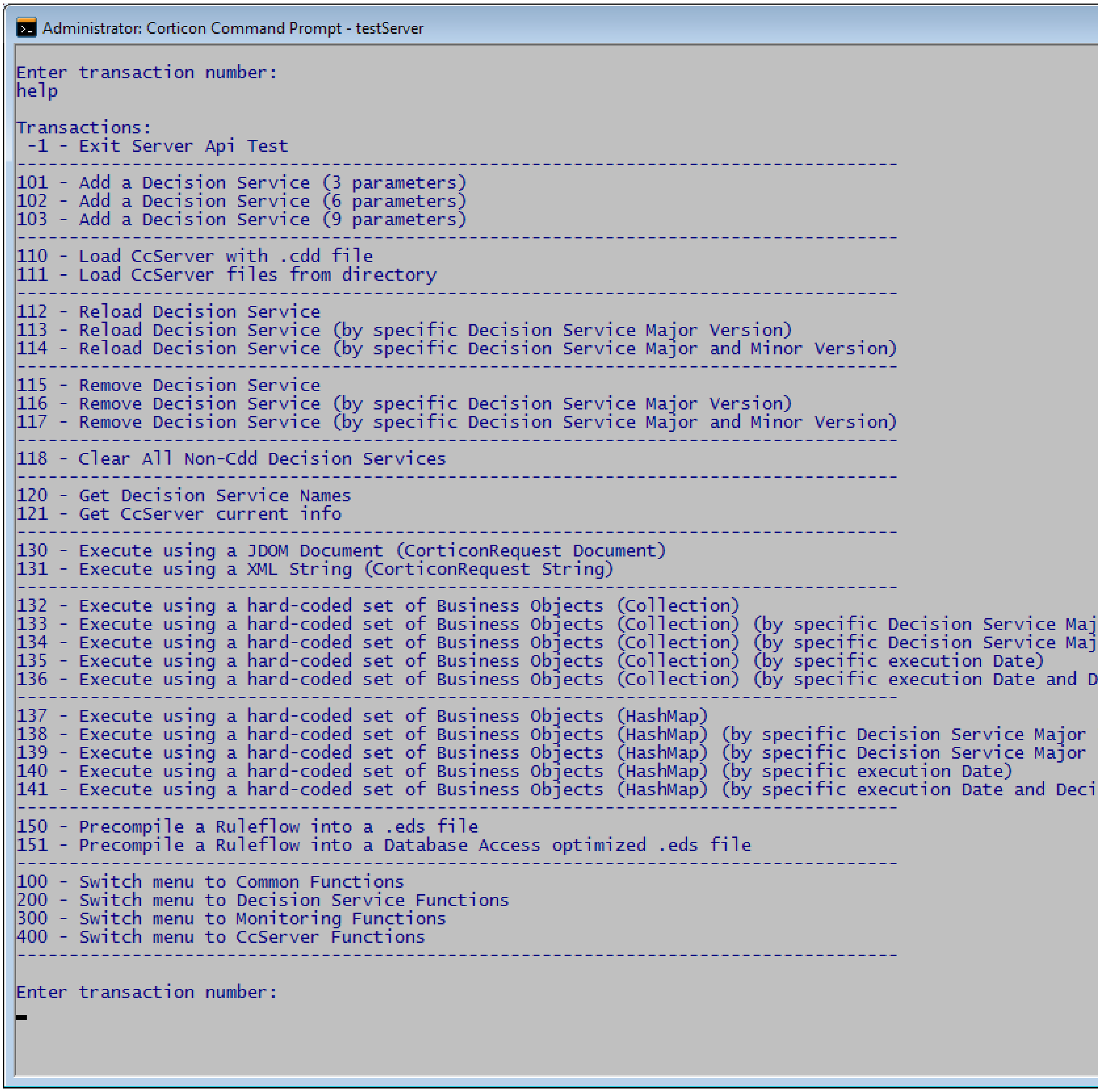
   Note: When you launch Corticon Command Prompt, it calls `corticon_env.bat`, adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts by name from several locations, and then relocates the prompt to the root of the Corticon work directory.

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



```
Enter transaction number:
help
```

4. The 100 series commands are listed.



```
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 Maj
134 - Execute using a hard-coded set of Business Objects (Collection) (by specific Decision Service Maj
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 D
------------------------------------------------------------------------------------
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
139 - Execute using a hard-coded set of Business Objects (HashMap) (by specific Decision Service Major
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 Deci
------------------------------------------------------------------------------------
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.

6

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 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 Tutorial.
- **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 Tutorial. 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 Tutorial: Advanced Rule Modeling in Corticon Studio 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.

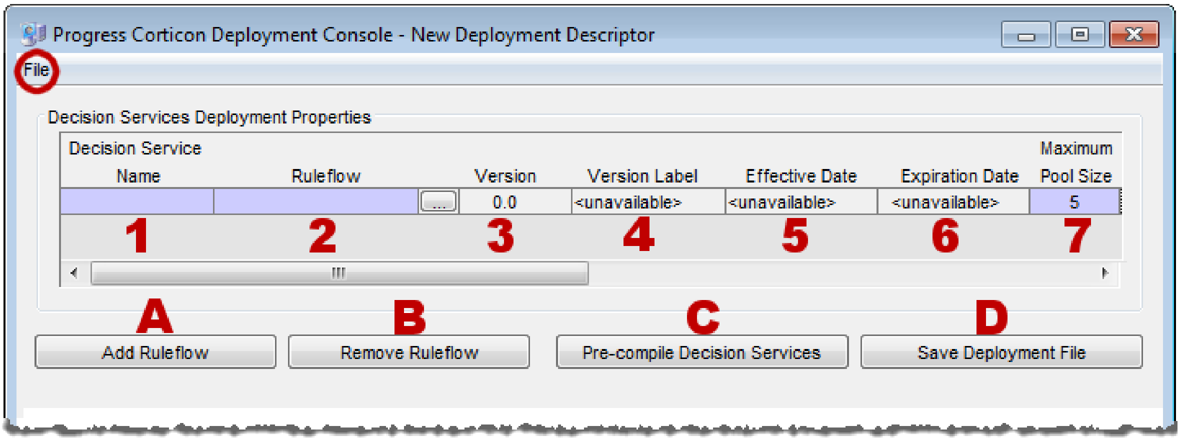
Deployment Descriptors are easily created using the Deployment Console, which is installed only by the Corticon Server installers.

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 5.5** > **Corticon Deployment Console** to launch the script file `\Server\deployConsole.bat`.

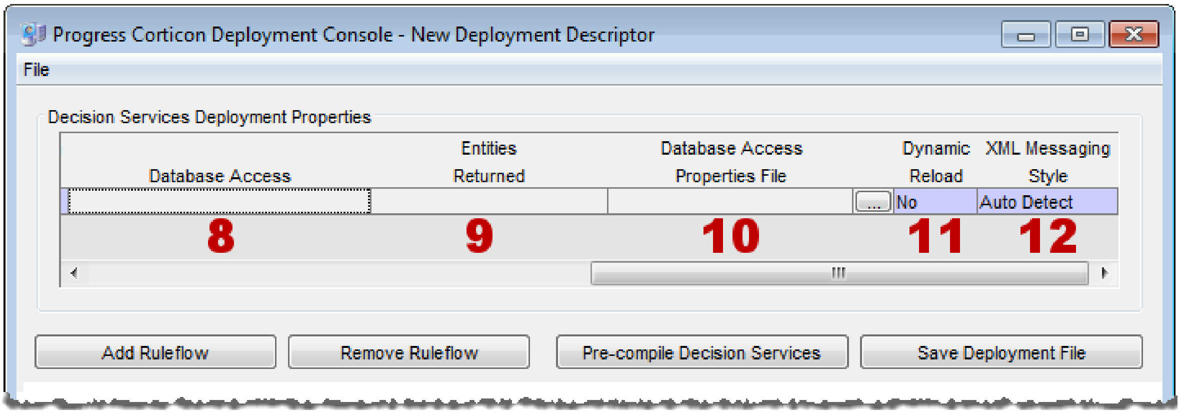
The Deployment Console is divided into two sections. Because the Deployment Console is a rather wide window, its columns are shown as two screen captures in the following figures. The **red** identifiers are the topics listed below.

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


Progress Corticon Deployment Console - New Deployment Descriptor
File
Decision Services Deployment Properties
1
2
3
4
5
6
7
A
B
C
D
Add Ruleflow
Remove Ruleflow
Pre-compile Decision Services
Save Deployment File

Decision Service Name	Ruleflow	Version	Version Label	Effective Date	Expiration Date	Maximum Pool Size
	...	0.0	<unavailable>	<unavailable>	<unavailable>	5

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


Progress Corticon Deployment Console - New Deployment Descriptor
File
Decision Services Deployment Properties
8
9
10
11
12
Add Ruleflow
Remove Ruleflow
Pre-compile Decision Services
Save Deployment File

Database Access	Entities Returned	Database Access Properties File	Dynamic Reload	XML Messaging Style
		...	No	Auto Detect

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 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** - *Active if your Corticon license enables EDC* - 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** - *Active if your Corticon license enables EDC* - 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. **Database Access Properties File** - *Active if your Corticon license enables EDC* - 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 database access properties 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 Decision Service Contract Structures section of the Integration chapter 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.

7

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

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.

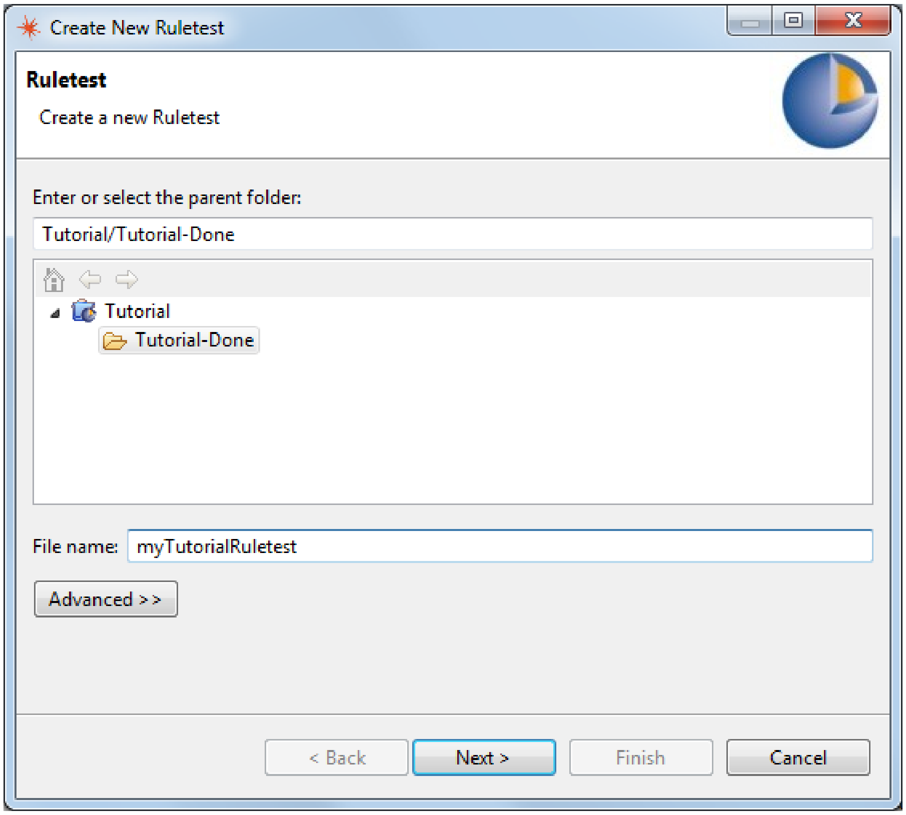
Path 1: Using Corticon Studio as a SOAP client to consume a Decision Service

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**.



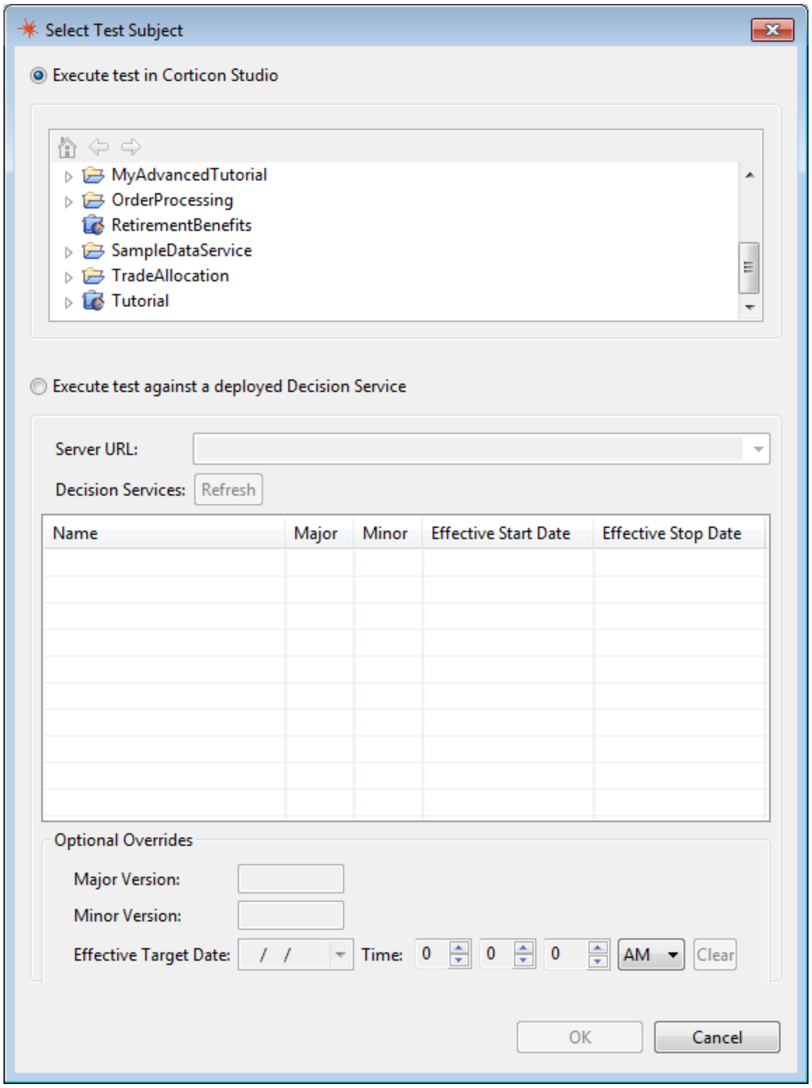
Ruletest

Create a new Ruletest

Enter or select the parent folder: Tutorial/Tutorial-Done

File name: myTutorialRuletest

2. Select the **parent folder** for the new Ruletest.
3. Enter a file name for the new Ruletest.
4. Click **Next** to continue and display the **Select Test Subject** dialog box, as shown:



Select Test Subject

● Execute test in Corticon Studio

- MyAdvancedTutorial
- OrderProcessing
- RetirementBenefits
- SampleDataService
- TradeAllocation
- Tutorial

○ Execute test against a deployed Decision Service

Server URL:

Decision Services: Refresh

Name	Major	Minor	Effective Start Date	Effective Stop Date

Optional Overrides

Major Version:

Minor Version:

Effective Target Date: / / Time: 0 0 0 AM Clear

OK Cancel

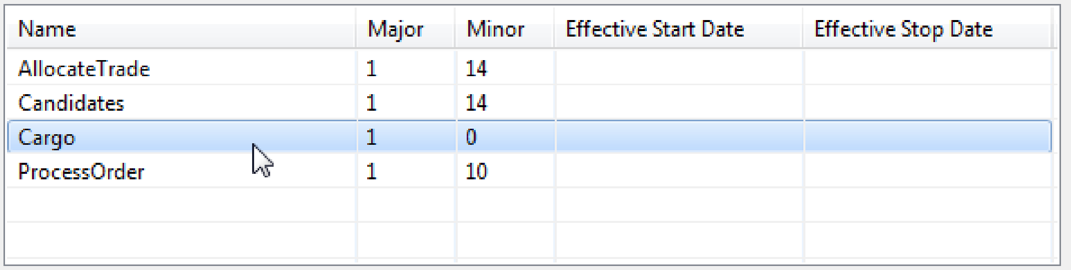
5. Click **Execute test against deployed Decision Service**.

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*.

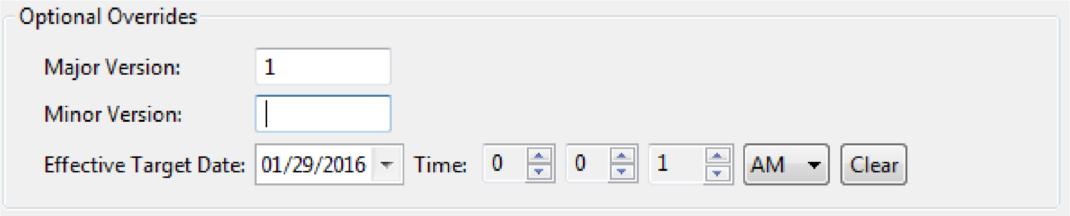
   Click **Refresh** to populate the list of deployed Decision Services on that server.

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 might want to click **OK** at this point.
9. When the selected Decision Service was deployed with a date range defined -- active from the effective date through the expiration date -- you might want to apply overrides to 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:



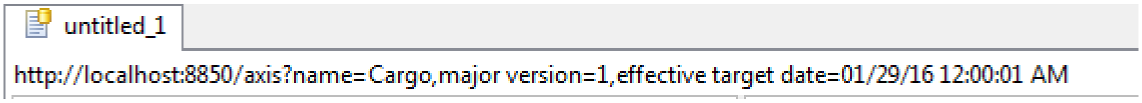
Optional Overrides

Major Version: 1

Minor Version:

Effective Target Date: 01/29/2016 Time: 0 0 1 AM

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



untitled_1

http://localhost:8850/axis?name=Cargo,major version=1,effective target date=01/29/16 12:00:01 AM

11. Run the Ruletest.

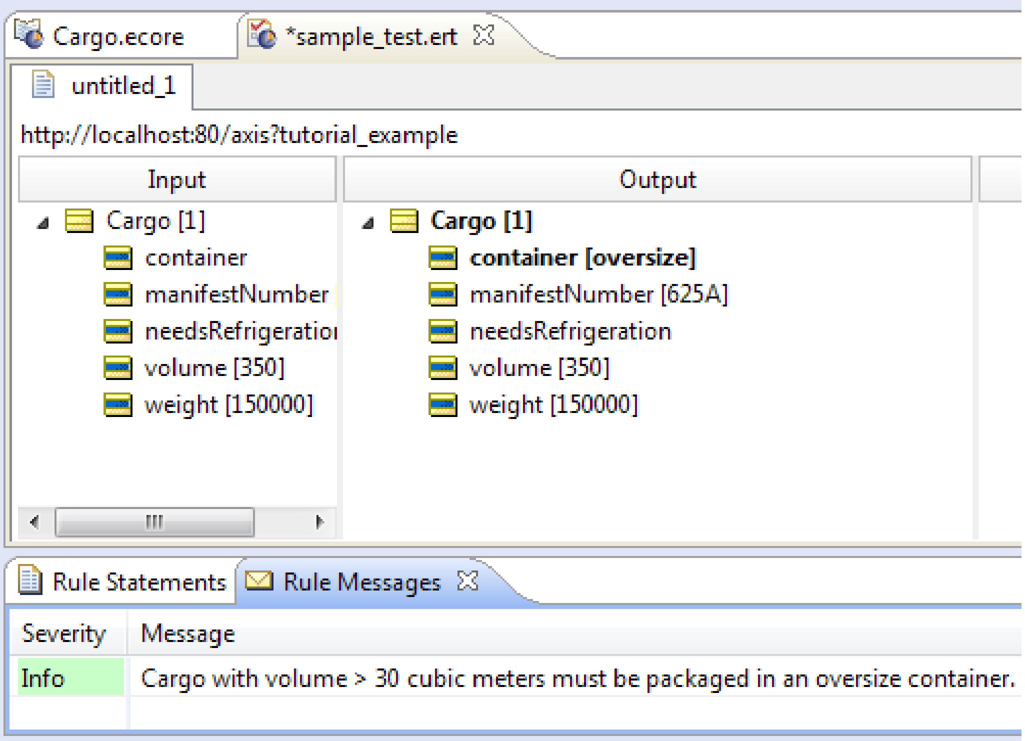
The test executes against the specified Decision Service on the selected Java web server using the overrides you entered.

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



http://localhost:80/axis?tutorial_example

Input	Output
Cargo [1]	**Cargo [1]**
container	**container [oversize]**
manifestNumber	manifestNumber [625A]
needsRefrigeratio	needsRefrigeration
volume [350]	volume [350]
weight [150000]	weight [150000]

Severity	Message
Info	Cargo with volume > 30 cubic meters must be packaged in an oversize container.

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

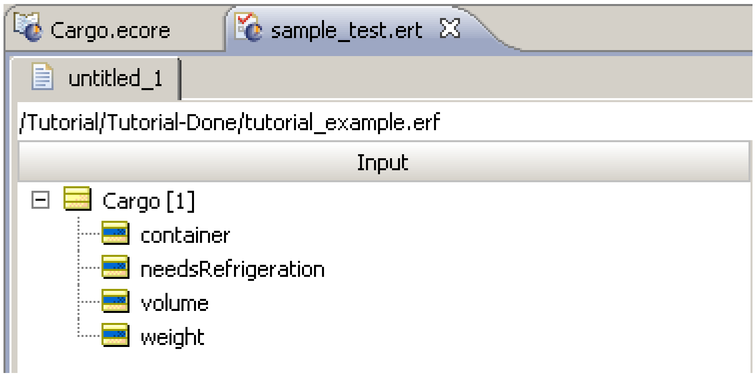
Creating a Request Message for a Decision Service

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



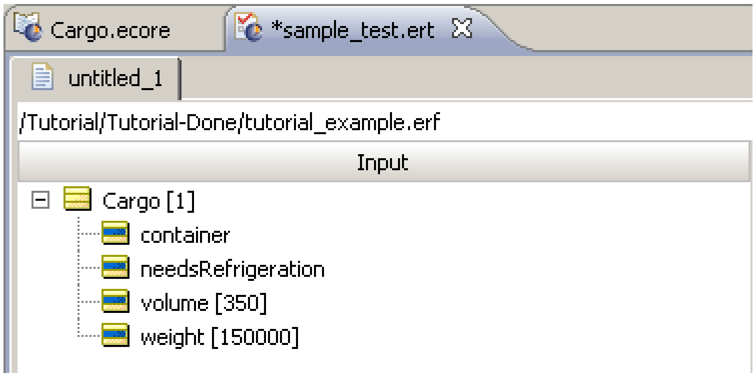
/Tutorial/Tutorial-Done/tutorial_example.erf

Input

- Cargo [1]
  - container
  - needsRefrigeration
  - volume
  - weight

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



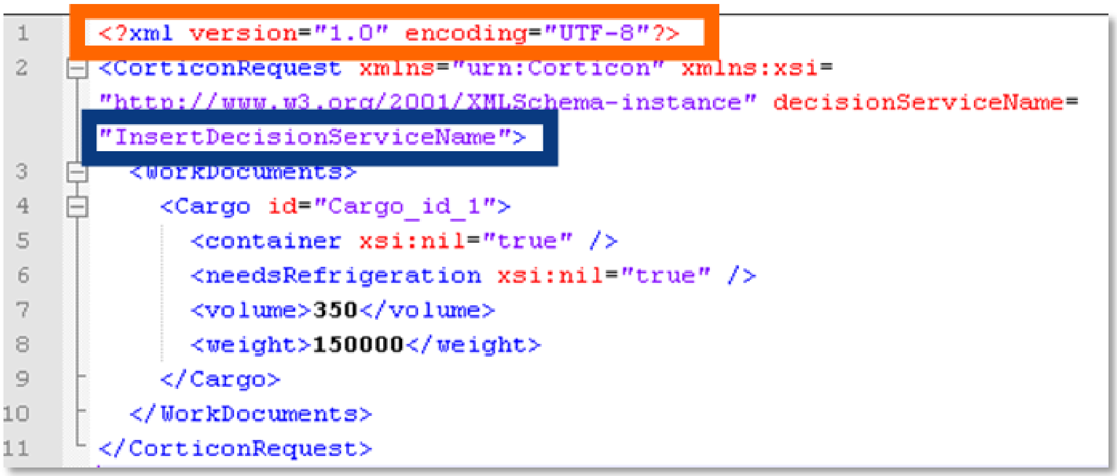
/Tutorial/Tutorial-Done/tutorial_example.erf

Input

- Cargo [1]
  - container
  - needsRefrigeration
  - volume [350]
  - weight [150000]

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

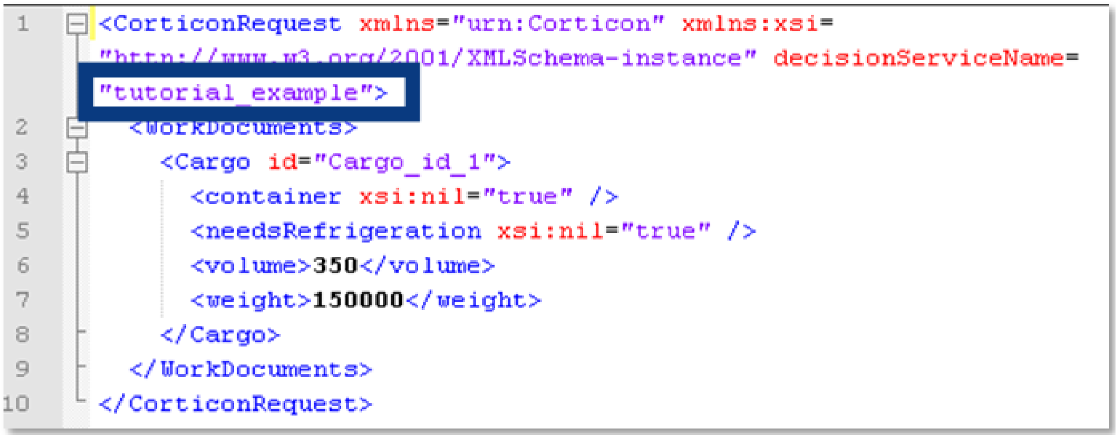
Figure 10: Sample XML File Exported from a Studio Test



```
<?xml version="1.0" encoding="UTF-8"?>
<CorticonRequest xmlns="urn:Corticon" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" decisionServiceName=
"InsertDecisionServiceName">
  <WorkDocuments>
    <Cargo id="Cargo_id_1">
      <container xsi:nil="true" />
      <needsRefrigeration xsi:nil="true" />
      <volume>350</volume>
      <weight>150000</weight>
    </Cargo>
  </WorkDocuments>
</CorticonRequest>
```

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



```
<CorticonRequest xmlns="urn:Corticon" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" decisionServiceName=
"tutorial_example">
  <WorkDocuments>
    <Cargo id="Cargo_id_1">
      <container xsi:nil="true" />
      <needsRefrigeration xsi:nil="true" />
      <volume>350</volume>
      <weight>150000</weight>
    </Cargo>
  </WorkDocuments>
</CorticonRequest>
```

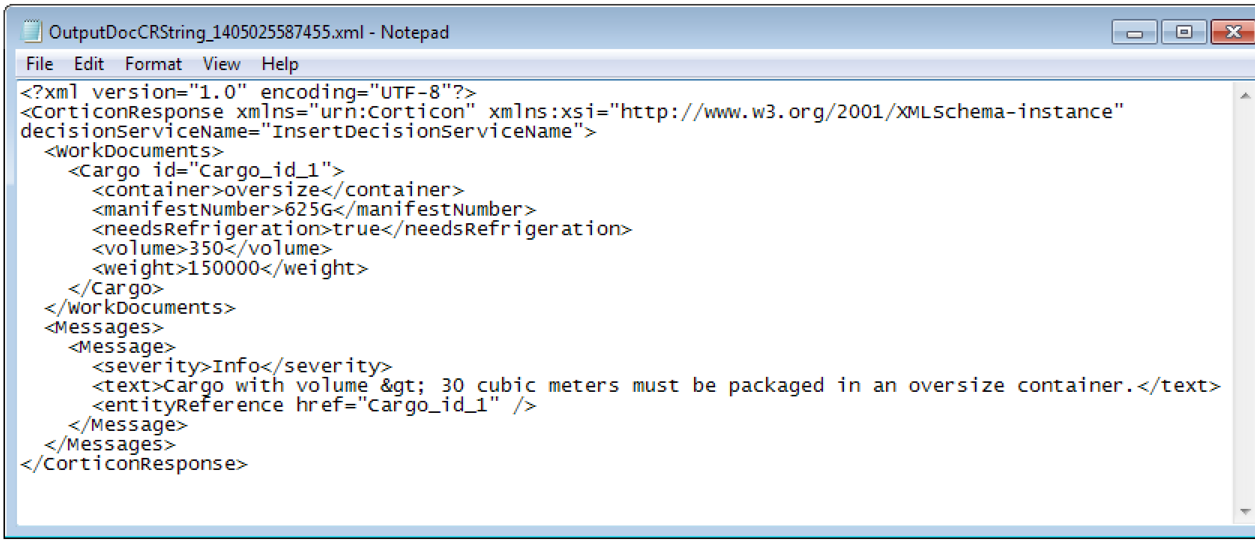
10. **Save** your changes to the XML file and exit your text editor.

Sending a request message to Corticon Server

The Corticon Server test scripts, described in Testing the installed Corticon Server on page 25, 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. In this Tutorial, our test XML file is `sample.xml`. When you press **Enter** the request is processed and a response 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



```
<?xml version="1.0" encoding="UTF-8"?>
<CorticonResponse xmlns="urn:Corticon" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
decisionServiceName="InsertDecisionServiceName">
  <WorkDocuments>
    <Cargo id="Cargo_id_1">
      <container>oversize</container>
      <manifestNumber>625G</manifestNumber>
      <needsRefrigeration>true</needsRefrigeration>
      <volume>350</volume>
      <weight>150000</weight>
    </Cargo>
  </WorkDocuments>
  <Messages>
    <Message>
      <severity>Info</severity>
      <text>Cargo with volume &gt; 30 cubic meters must be packaged in an oversize container.</text>
      <entityReference href="Cargo_id_1" />
    </Message>
  </Messages>
</CorticonResponse>
```

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 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

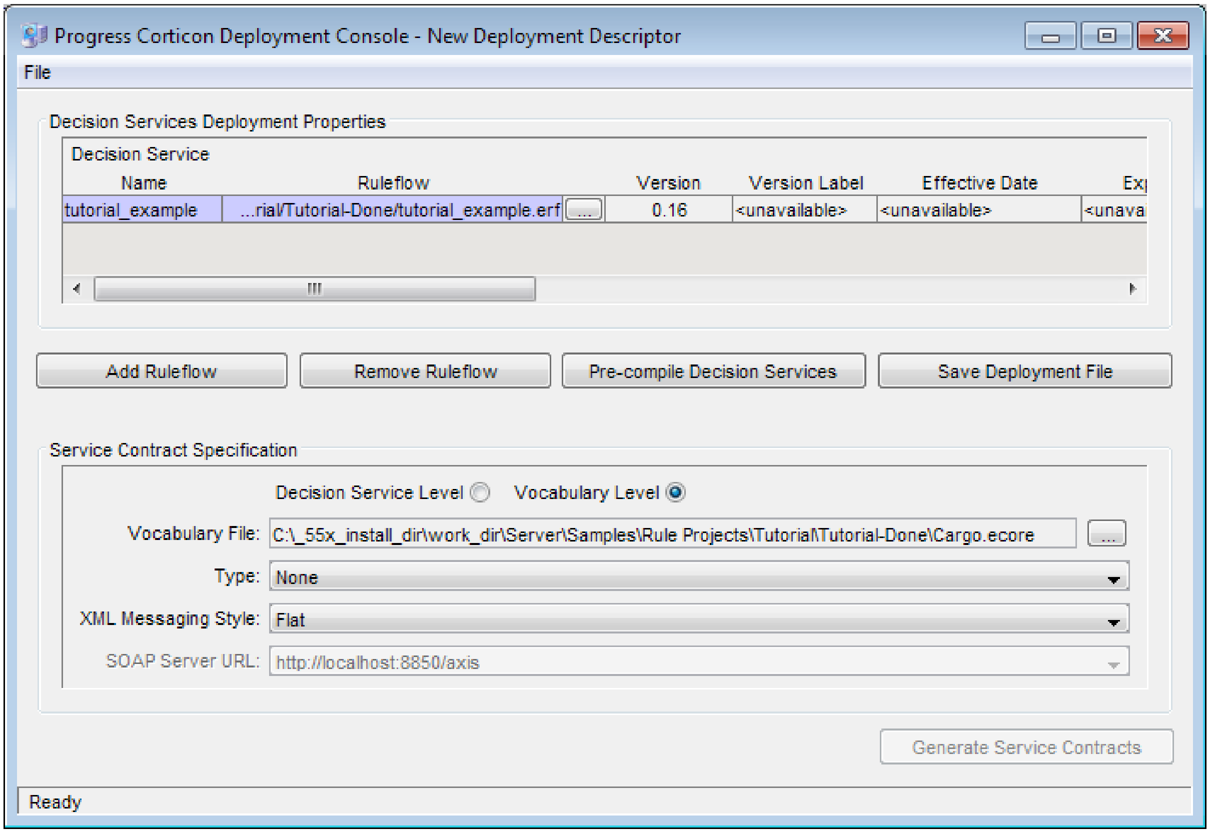
There are also two types of SOAP messaging styles commonly used in web services:

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

Figure 13: Deployment Console's Service Contract Specification Window


Progress Corticon Deployment Console - New Deployment Descriptor
File
Decision Services Deployment Properties
Decision Service
Add Ruleflow
Remove Ruleflow
Pre-compile Decision Services
Save Deployment File
Service Contract Specification
Generate Service Contracts
Ready

Name	Ruleflow	Version	Version Label	Effective Date	Ex
tutorial_example	...rial/Tutorial-Done/tutorial_example.erf	0.16	<unavailable>	<unavailable>	<unava

Decision Service Level ○ Vocabulary Level ●

Vocabulary File: C:_55x_install_dir\work_dir\Server\Samples\Rule Projects\Tutorial\Tutorial-Done\Cargo.ecore

Type: None

XML Messaging Style: Flat

SOAP Server URL: http://localhost:8850/axis

Launch the **Deployment Console**, and then perform the following steps to generate a service contract. All the **Deployment Console** options are also described in detail in the *Corticon Server: Integration & Deployment Guide*.

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 section of this tutorial.

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 (similar to the SOAP `.xml` sample) and a test script that runs the sample.

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

```
{"Objects": [{
    "total": null,
    "myItems": [
        {
            "product": "Ball",
            "price": "10.000000",
            "quantity": "20",
            "subtotal": null,
            "__metadata": {
                "#id": "Item_id_1",
                "#type": "Item"
            }
        },
        {
            "product": "Racket",
            "price": "20.000000",
            "quantity": "1",
            "subtotal": null,
            "__metadata": {
                "#id": "Item_id_2",
                "#type": "Item"
            }
        },
        {
            "product": "Wrist Band",
            "price": "5.250000",
            "quantity": "2",
            "subtotal": null,
            "__metadata": {
                "#id": "Item_id_3",
                "#type": "Item"
            }
        }
    ],
    "shipped": null,
    "shippedOn": null,
    "__metadata": {
        "#id": "Order_id_1",
        "#type": "Order"
    },
    "dueDate": "1/1/2008 12:00:00 AM",
    "note": null
}]}
```

To run the JSON sample:

1. Start Corticon Server.
2. Open a command prompt window at `[CORTICON_HOME]\Server\bin`.
3. Enter `testServerREST.bat`. The command transaction list is displayed:

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

Transactions:
 -1 - Exit REST API Test
----------------------------------------------------------------------------

  0 - Change Connection Parameters
----------------------------------------------------------------------------
142 - Execute JSON REST request
143 - Execute JSON REST request (by specific Decision Service Major Version)
144 - Execute JSON REST request (by specific Decision Service Major and Minor
 Version)
145 - Execute JSON REST request (by specific execution Date)
146 - Execute JSON REST request (by specific execution Date and Decision
Service Major Version)
----------------------------------------------------------------------------
Enter transaction number
```

4. Enter `142`.
5. When prompted for **Input JSON File Path**, enter (or copy) the path to the sample:

```
C:\Users\{user}\Progress\CorticonWork_5.5\Samples\Rule
Projects\OrderProcessing\OrderProcessingPayload.json
```

6. When prompted for **Input Decision Service Name**, enter (or copy) the name of the Decision Service that is the sample's target:

```
ProcessOrder
```

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": "Item_id_3",
                "text": "The subtotal of line item for Wrist Band is
10.500000.",
                "severity": "Info",
                "__metadata": {"#type": "#RuleMessage"}
            },
            {
                "entityReference": "Item_id_2",
                "text": "The subtotal of line item for Racket is 20.000000.",

                "severity": "Info",
                "__metadata": {"#type": "#RuleMessage"}
            },
            {
                "entityReference": "Item_id_1",
                "text": "The subtotal of line item for Ball is 200.000000.",
                "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
12:00:00 AM",
                "severity": "Warning",
                "__metadata": {"#type": "#RuleMessage"}
            }
        ],
        "__metadata": {"#type": "#RuleMessages"},
        "version": "0.0"
    },
    "Objects": [{
        "total": 230.5,
        "myItems": [
            {
                "product": "Ball",
                "price": "10.000000",
                "quantity": "20",
                "subtotal": 200,
                "__metadata": {
                    "#id": "Item_id_1",
                    "#type": "Item"
                }
            },
            {
                "product": "Racket",
                "price": "20.000000",
                "quantity": "1",
                "subtotal": 20,
                "__metadata": {
                    "#id": "Item_id_2",
                    "#type": "Item"
                }
            },
            {
                "product": "Wrist Band",
                "price": "5.250000",
                "quantity": "2",
                "subtotal": 10.5,
                "__metadata": {
                    "#id": "Item_id_3",
                    "#type": "Item"
                }
            }
        ],
        "shipped": true,
        "shippedOn": "12/1/2008 12:00:00 AM",
        "__metadata": {
            "#id": "Order_id_1",
            "#type": "Order"
        },
        "dueDate": "1/1/2008 12:00:00 AM",
        "note": "This Order was shipped late"
    }]
}
```

Troubleshooting Java server

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

For more information, see *"Corticon Deployment Console properties" in the Integration and Deployment Guide.*

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

8

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. **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. **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

The Java Server Console

Corticon's Java Server Console is a browser-based administration tool for monitoring and managing installations of Corticon Server for Java. The Java Server Console is accessed through a browser connecting to the URL (for a Java Web Service)`http://{server_hostname}:8850/axis`, and then using either of the default modeler credentials:

- Username `modeler1` with its password `modeler1`
- Username `modeler2` with its password `modeler2`

Note: See the next set of topics, Using the Corticon Java Server Console on page 58 for detailed information about all the Server Console panels, tabs, and functions. Consult with your administrator for other credentials that provide more (or less) rights with the Server Console.

For details, see the following topics:

- Telling the Java server where to find Deployment Descriptor files
- Creating and deploying the sample Decision Services
- Modifying and deploying the sample Decision Services
- How Test Decision Services differ from Production (live) Decision Services
- Determining which Decision Service to execute against based on Versions and Effective/Expires Dates

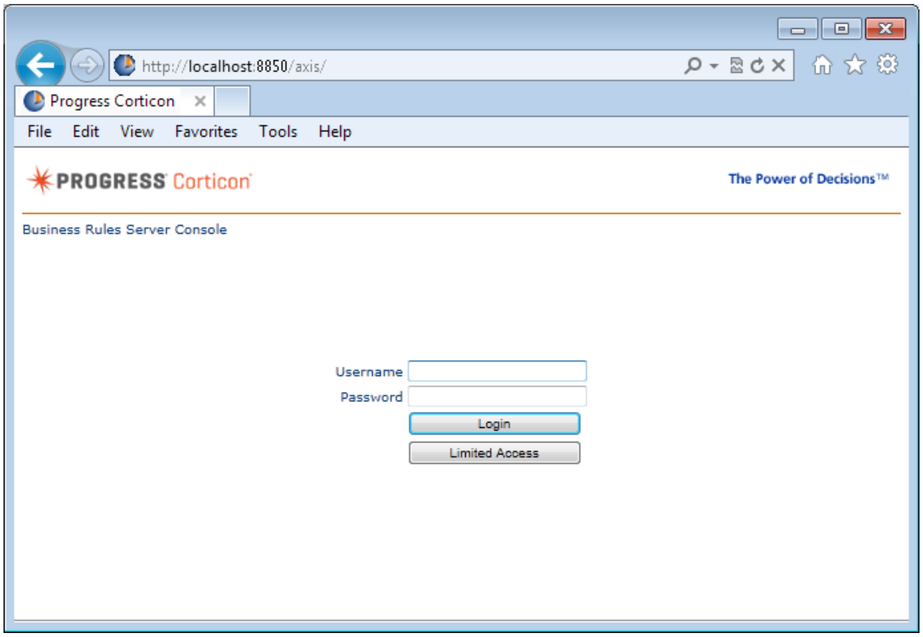
Using the Corticon Java Server Console

Corticon's Server Console is a browser-based administration tool for monitoring and managing installations of Corticon Server for Java.

Launching and logging in to Corticon Server Console

When your Corticon Java Server is running, open a web browser to the URL `http://<yourServerURL>:<yourPort>/axis` in a web browser. By default, `axis.war` is installed in the bundled Progress Application Server, and the default URL is `http://localhost:8850/axis`. The Corticon Server Console login page opens, as shown:

Figure 14: Server Console Login Page



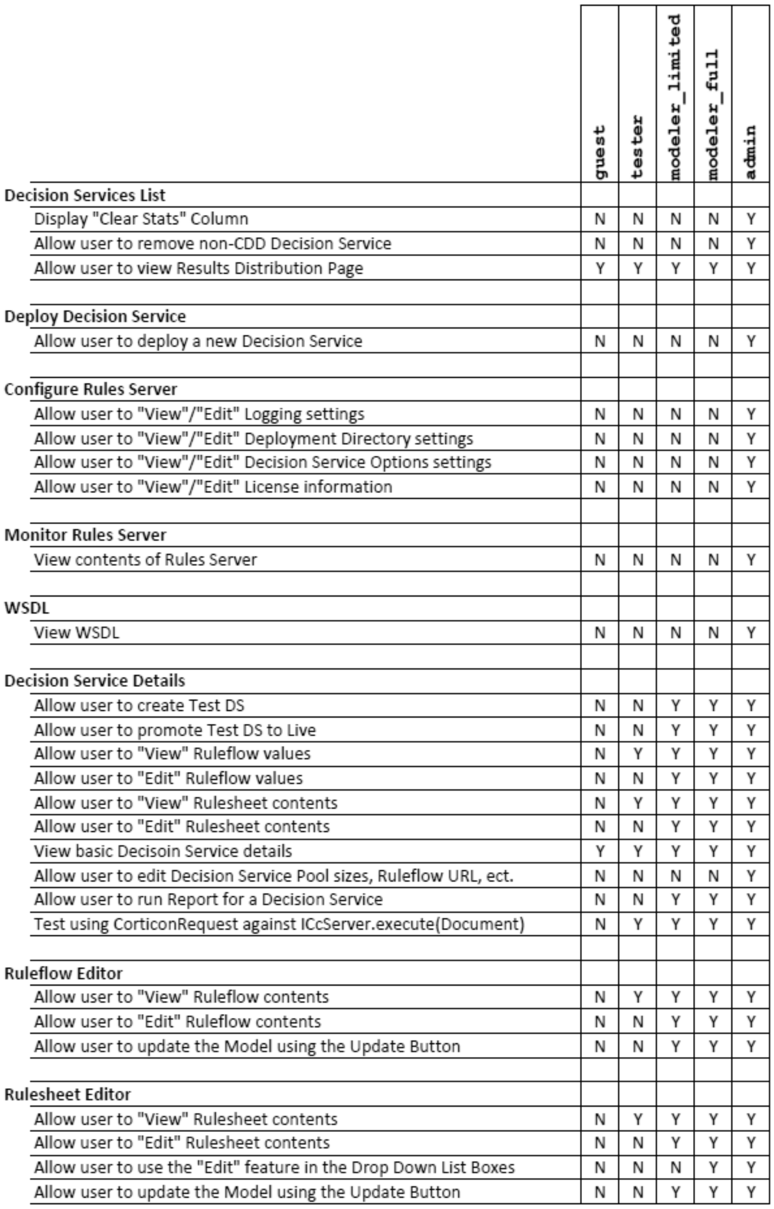
As installed, the Corticon Server has six Server Console login usernames defined, listed here in descending order of strength:

Table 1: Server Console Default Credentials

Username	Password
`admin`	`admin`
`administrator`	`changeme`
`modeler2`	`modeler2`
`modeler1`	`modeler1`
`tester`	`tester`
`<empty>`	`<empty>`

The **rights** assigned to a username specify the view, read, and write functions that are enabled for the user, as follows:

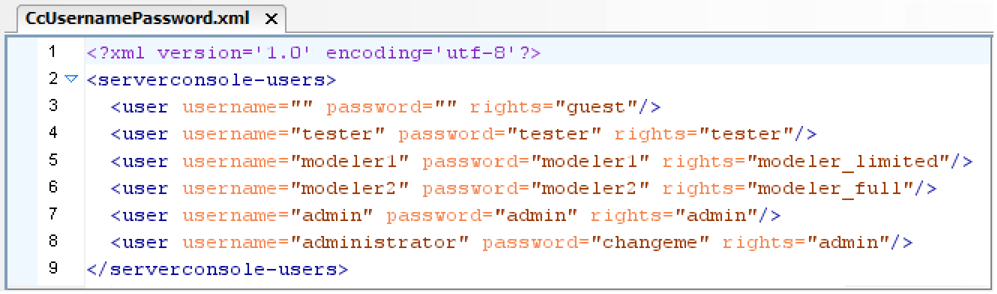
Figure 15: Server Console User Rights



	guest	tester	modeler_limited	modeler_full	admin
Decision Services List					
Display "Clear Stats" Column	N	N	N	N	Y
Allow user to remove non-CDD Decision Service	N	N	N	N	Y
Allow user to view Results Distribution Page	Y	Y	Y	Y	Y
Deploy Decision Service					
Allow user to deploy a new Decision Service	N	N	N	N	Y
Configure Rules Server					
Allow user to "View"/"Edit" Logging settings	N	N	N	N	Y
Allow user to "View"/"Edit" Deployment Directory settings	N	N	N	N	Y
Allow user to "View"/"Edit" Decision Service Options settings	N	N	N	N	Y
Allow user to "View"/"Edit" License information	N	N	N	N	Y
Monitor Rules Server					
View contents of Rules Server	N	N	N	N	Y
WSDL					
View WSDL	N	N	N	N	Y
Decision Service Details					
Allow user to create Test DS	N	N	Y	Y	Y
Allow user to promote Test DS to Live	N	N	Y	Y	Y
Allow user to "View" Ruleflow values	N	Y	Y	Y	Y
Allow user to "Edit" Ruleflow values	N	N	Y	Y	Y
Allow user to "View" Rulesheet contents	N	Y	Y	Y	Y
Allow user to "Edit" Rulesheet contents	N	N	Y	Y	Y
View basic Decisoin Service details	Y	Y	Y	Y	Y
Allow user to edit Decision Service Pool sizes, Ruleflow URL, ect.	N	N	N	N	Y
Allow user to run Report for a Decision Service	N	N	Y	Y	Y
Test using CorticonRequest against ICcServer.execute(Document)	N	Y	Y	Y	Y
Ruleflow Editor					
Allow user to "View" Ruleflow contents	N	Y	Y	Y	Y
Allow user to "Edit" Ruleflow contents	N	N	Y	Y	Y
Allow user to update the Model using the Update Button	N	N	Y	Y	Y
Rulesheet Editor					
Allow user to "View" Rulesheet contents	N	Y	Y	Y	Y
Allow user to "Edit" Rulesheet contents	N	N	Y	Y	Y
Allow user to use the "Edit" feature in the Drop Down List Boxes	N	N	N	Y	Y
Allow user to update the Model using the Update Button	N	N	Y	Y	Y

The XML file that maintains these credentials is the following:

Figure 16: CcUsernamePassword.xml



```
<?xml version='1.0' encoding='utf-8'?>
<serverconsole-users>
  <user username="" password="" rights="guest"/>
  <user username="tester" password="tester" rights="tester"/>
  <user username="modeler1" password="modeler1" rights="modeler_limited"/>
  <user username="modeler2" password="modeler2" rights="modeler_full"/>
  <user username="admin" password="admin" rights="admin"/>
  <user username="administrator" password="changeme" rights="admin"/>
</serverconsole-users>
```

To add, modify, or delete Server Console users, access CcConfig.jar in your installation directory, either at \Server\lib (in-process use) or \Server\pas\corticon\webapps\axis\WEB-INF\lib (Progress App Server use.) Edit the file CcUsernamePassword.xml and update it to suit your preferences. Save the updated file in its CcConfig.jar.

If you login using the <empty> account, or if you choose the **Limited Access** login option, your use of the Server Console will be limited to read-only mode. In this mode, you will not be able to deploy Decision Services or configure or monitor the Server. If you try to view the Deploy Decision Service, Configure Rules Server, or Monitor Rules Server pages (see below), you get appropriate alerts that "User does not have rights to...".

Try logging in as admin or administrator to expose the complete set of features and functionality that are documented in this chapter. Or, follow the instructions in the following topic to move your authentication to your LDAP domains, and then login as a user that has admin rights.

Using LDAP authentication in Server Console

Instead of the built-in user and rights definitions, Corticon Server Console lets you choose to use Lightweight Directory Access Protocol (LDAP) domains for role-based authentication, so that you can control access to Corticon Server Console and define roles in your current user management systems, such as Microsoft's Active Directory.

Note: The interface to LDAP evaluates all credentials (username and password) and properties (name and value) as case-sensitive.

Properties for LDAP

The name-value lines in a well-formed ldap.properties file are as follows:

Table 2: LDAP properties for Server Console authentication

Property	Description
ldap.base.provider.url	URL for connection to a directory service.
ldap.base.dn	Parent Domain for the specified directory.
ldap.security.principal	Qualified username or e-mail address of admin user.
ldap.security.credentials	Password of the user
ldap.user.base	Location to search for users.
ldap.group.base	Location to search for groups.
ldap.user.search	Search field for user.
ldap.group.search	Search field for group.
ldap.group.mappings	Mapping of Corticon Server Console Roles with Active Directory groups.

Property	Description
`ldap.check.all.directories`	Property to authenticate user against all specified directories in `ldap.server.ids`
`ldap.server.ids`	List all directory services.

Configuration Examples

The following three configurations cover most use cases:

- Lookup in one directory service
- Lookup in multiple specified directory services
- Lookup across a list of directory services

The following topics detail each of these use cases.

Lookup in one directory service

The following block shows the content of an `ldap.properties` file with minimal required properties.

```
# Active Directory Configuration Details
ldap.base.provider.url=ldap://{hostname}:389/
ldap.base.dn=dc=test,dc=local
ldap.security.principal=admin@test.local
ldap.security.credentials=password

# Defaults search base to ldap.base.dn if user/group base is not specified
#ldap.user.base=cn=Users,dc=test,dc=local
#ldap.group.base=ou=Roles,dc=test,dc=local

ldap.user.search=sAMAccountName
ldap.group.search=sAMAccountName
ldap.group.mappings=admin\=group1,group3\nmodeler_full\=group2\ntester\=group4
```

The syntax for the username field on the login page is `username`. For example, `testuser`

Lookup in multiple directory services

In this authentication type, user has to give domain name and the username to authenticate against the specified directory. There are two regions in this example: `americas` and `apac`.

```
# Specify multiple domains(Active Directory) configurations

# Default Settings
ldap.base.provider.url=ldap://{hostname}:{port}/
ldap.base.dn=dc=test,dc=local
ldap.security.principal=Administrator@test.local
ldap.security.credentials=password
ldap.user.search=sAMAccountName
ldap.group.search=sAMAccountName
#ldap.user.base=cn=Users,dc=test,dc=local
#ldap.group.base=ou=Roles,dc=test,dc=local
ldap.group.mappings=admin\=group1,group3\nmodeler_full\=group2

# americas settings
ldap.base.provider.url.americas=ldap://{hostname}:{port}/
ldab.base.dn.americas=dc=sample,dc=local
```

```
ldap.security.principal.americas=Administrator@sample.local
ldap.security.credentials.americas=password
ldap.user.search.americas=sAMAccountName
ldap.group.search.americas=sAMAccountName
ldap.user.base.americas=cn=Users,dc=sample,dc=local
ldap.group.base.americas=ou=Roles,dc=sample,dc=local
#ldap.group.mappings.americas=admin=group1,group3\nmodeler_full=group2
```

The syntax for the username field on the login page is `domainId\username`. For example, `apac\testuser`

Lookup across a list directory services

In this authentication type, a user is authenticated against a list of directory service . Authentication process stops when the user is successfully authenticated against any specified directory.

```
# Configuration to check for user in all listed directories

# Check all directories
ldap.check.all.directories=true
# LDAP Server IDs
ldap.server.ids=apac,americas,default

# apac Settings
ldap.base.provider.url.apac=ldap://{hostname}:{port}/
ldap.base.dn.apac=dc=test1,dc=local
ldap.security.principal.apac=admin@test1.local
ldap.security.credentials.apac=password
ldap.user.search.apac=sAMAccountName
ldap.user.base.apac=cn=Users,dc=test1,dc=local
ldap.group.search.apac=sAMAccountName
ldap.group.base.apac=ou=Roles,dc=test1,dc=local
ldap.group.mappings.apac=admin\=group1,group3\nmodeler_full\=group2

# americas settings
ldap.base.provider.url.americas=ldap://{hostname}:{port}/
ldab.base.dn.americas=dc=test2,dc=local
ldap.security.principal.americas=Administrator@test2.local
ldap.security.credentials.americas=password
ldap.user.search.americas=sAMAccountName
ldap.user.base.americas=cn=Users,dc=test2,dc=local
ldap.group.search.americas=sAMAccountName
ldap.group.base.americas=ou=Roles,dc=test2,dc=local
#ldap.group.mappings.americas=admin=group1,group3\nmodeler_full=group2

# Default Settings
ldap.base.provider.url=ldap://{hostname}:{port}/
ldap.base.dn=dc=test2,dc=local
ldap.security.principal=Administrator@test2.local
ldap.security.credentials=password
#ldap.user.search=sAMAccountName
#ldap.user.base=cn=Users,dc=test2,dc=local
#ldap.group.search=sAMAccountName
#ldap.group.base=ou=Roles,dc=test2,dc=local
ldap.group.mappings=admin\=group1,group3\nmodeler_full\=group2
```

The syntax for the username field on the login page is `username`. For example, `testuser`. If a user provides `domainId\username` to login, authenticatition will look only in the specified domain.

Deploying LDAP authentication

Once you have defined your LDAP properties, save the file as `ldap.properties` locally, or - if you are using clusters of servers - a network-accessible location so that all cluster members can access it.

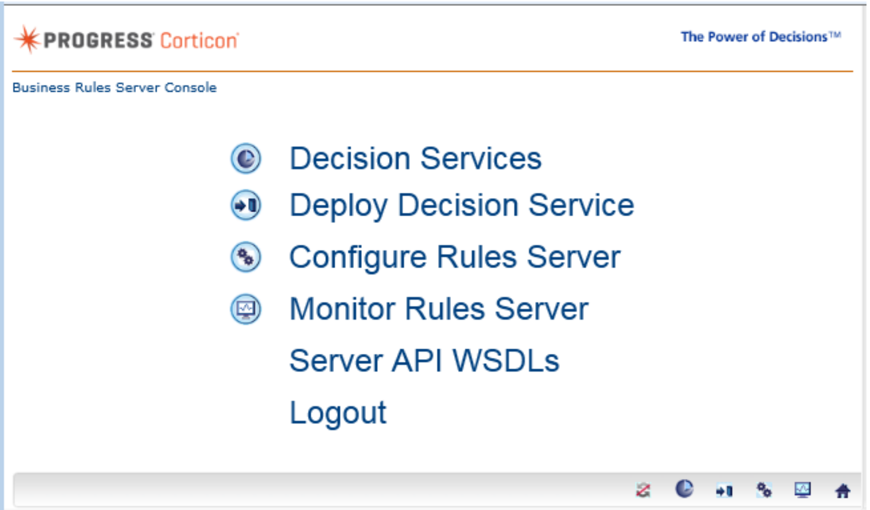
When you initiate or change the `ldap.properties` file, you need to restart the Corticon Server to implement the changes. Once the server restarts, user connections that were dropped will need to use LDAP credentials to connect their Server Console to the Corticon Server.

Note: You can revert to the non-LDAP authentication mechanisms by removing (or relocating) the `ldap.properties` file. Then, when you restart the Corticon Server to implement the changes, user connections that were dropped will need to use the local credentials (in `CcUsernamePassword.xml`) to connect their Server Console to the Corticon Server. See the preceding topic for more information.

Console main menu page

After a successful login, the Server Console opens to its **Home** page.

Figure 17: Server Console Home Page



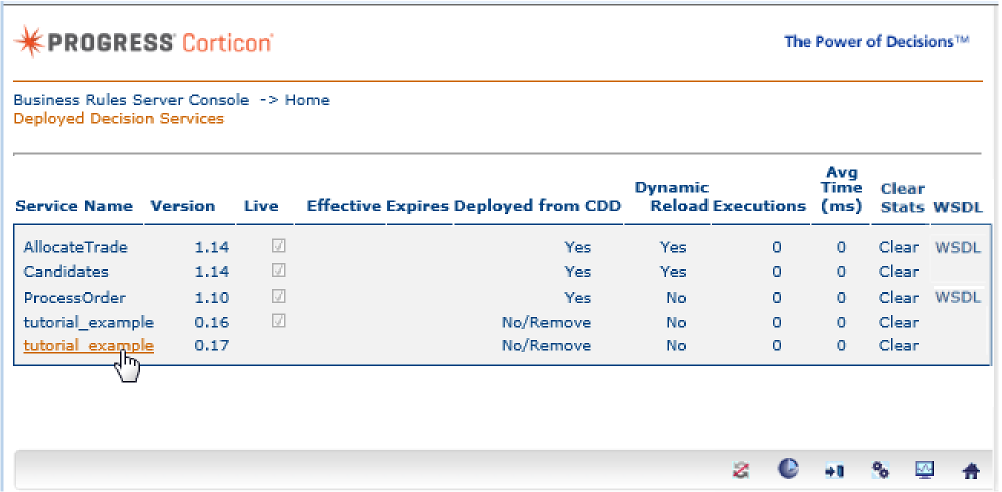
The lower right corner of these pages provide buttons to navigate to other top-level pages:

Icon	Description
🔄	Enable automatic Refresh of the Server Console display. When enabled, the Console updates every five seconds.
🚫	Disable refresh of the Server Console display. It is a good practice to do this when you are changing settings to avoid losing unsaved entries.
◐	Show the Decision Services menu.
➡	Show the Deploy Decision Service menu.
⚙	Show the Configure Rules Server menu.
🖥	Show the Monitor Rules Server menu.
🏠	Show the **Home** menu.

Decision Services page

Click **Decision Services** on the Home page (or ◐ from the icon bar) to open a page that lists the Decision Services currently deployed to the Corticon Server instance monitored by Server Console.

Figure 18: the Decision Services Page



Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats	WSDL
AllocateTrade	1.14	☑			Yes	Yes	0	0	Clear	WSDL
Candidates	1.14	☑			Yes	Yes	0	0	Clear	
ProcessOrder	1.10	☑			Yes	No	0	0	Clear	WSDL
tutorial_example	0.16	☑			No/Remove	No	0	0	Clear	
tutorial_example	0.17				No/Remove	No	0	0	Clear	

Each line item on the **Decision Services** page has these columns:

- **Service Name** - The name assigned to a deployed Decision Service, whether deployed by a .cdd file, or by data manually entered on the Deploy Decision Service page. When assigned via a .cdd file, the name is entered as the Decision Service Name in the Deployment Console. Each Service Name provides a link to the deployment properties of that Decision Service. Click on a Service Name to open its information tabs, as shown above.
- **Version** - The version identifier of the deployed Decision Service. A Decision Service's version number is set in the **Ruleflow** > **Properties** menu option of Corticon Studio for each Ruleflow (`.erf` file). See Decision Service Versioning and Effective Dating in this manual, and the *Rule Modeling Guide* for more information.
- **Live** - The deployment status of the Decision Service. When deployed and ready to receive invocations, the **Live** checkbox is checked. If a new version of a Decision Service has been created using the **Overview** tab, then it will not become live until promoted. This process is described in more detail in the *Rule Modeling Guide*.
- **Effective** - The date on which the selected Decision Service becomes active.
- **Expires** - The date on which the selected Decision Service ceases to be active.
- **Deployed from CDD** - Indicates whether a Decision Service has been deployed from a `.cdd` file (**Yes**) or from the Deploy Decision Service page (**No/Remove**). Note that **No/Remove** is hyperlinked, indicating that a Decision Service deployed using Server Console (from the Deploy Decision Service page) can also be removed from within Server Console by clicking the link. Decision Services deployed from a `.cdd` can only be removed by editing or removing the `.cdd`. Decision Services deployed via the Server Console update the `serverState.xml` file. Likewise, if a Decision Service is *removed* using the **No/Remove** hyperlink, then its corresponding entry is removed from `serverState.xml`.
- **Dynamic Reload** - Indicates whether Corticon Server's maintenance thread will watch for changes to the Decision Service and refresh the pool if updates are detected. For Decision Services deployed through a `.cdd`, this option is set in the Dynamic Reload field of the Deployment Console. If deployed via the Deploy Decision Service page of Server Console, Dynamic Reload is automatically **Yes**.
- **Executions** - The number of times the Decision Service has executed since statistics were last cleared. Click on its value to display its tabs and visualizations of execution average time and counts. This feature is enabled only when you turn **On** the optional **Decision Service Performance Monitoring Service**, located on the Decision Service Options tab on page 78.
- **Avg Time(ms)** - The amount of time, on average, required by the Decision Service to execute. This feature is enabled only when you turn **On** the optional **Decision Service Performance Monitoring Service**, located on the Decision Service Options tab on page 78.
- **Clear Stats** - Sets **Execution**, and **Average Time** -- as well as their **Performance** and **Distribution Chart** -- data to zero. You need administrator rights to perform the Clear Stats action.
- **WSDL** - Opens the generated WSDL file that has been attached to the selected Decision Service in a text editor. When you compile assets to create a Decision Service, the resulting `.eds` file will embed its Decision Service-level WSDL service contract. You can still generate Service Contracts from the Deployment Console as separate files, but you need to recompile them to get an embedded WSDL.

Note: Attachment of a WSDL to a Decision Service is an option, enabled by default, that can be turned off by setting the Server Property `com.corticon.server.compile.add.wsdl = false` in the server's `brms.properties` file. If you turn it off, subsequent Decision Services will not embed WSDL, while any `.eds` files that do have it will retain it and display their `WSDL` link.

Decision Service actions and information

The four buttons at the top of the page and the four tabs of each Decision Service's information page are discussed in this topic.

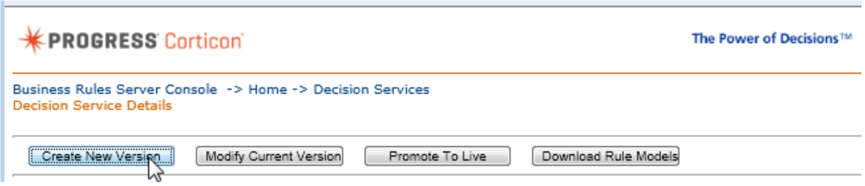
If the Decision Service was deployed by a `.cdd` file, then the contents of the `.cdd` are displayed on this page. If the Decision Service was deployed by the **Deploy Decision Service** page, then the information entered on that page (which is persisted in `serverstate.xml`) is displayed. The information displayed on the **Overview** page, shown in Overview Tab, is collected from three sources:

- From the `.cdd` file or from the information entered in the **Deploy Decision Service** page (depending on how the Decision Service was deployed). In both cases, the information reflects the state of the Decision Service at the time of deployment.
- From the Ruleflow file (`.erf`) itself, either from file `DateTime` stamps or from properties set in the **Ruleflow** > **Properties** menu of Corticon Studio.
- Retrieved real-time/dynamically from the Server Console. For example **Pool Settings|Available/Total Instances in Pool** reflects the current Reactor pool status for that Decision Service. The number of Reactors available and total, will change as Corticon Server grows/shrinks each Decision Service's pool to handle demand. For more information about Reactor pools, see Pool Size and Optimizing Pool Size.

BUTTONS

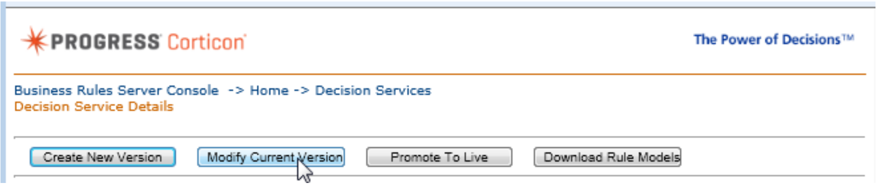
The four buttons at the top of a Decision Service information page let you perform the following actions:

- **Create New Version** - Click the button, as shown:



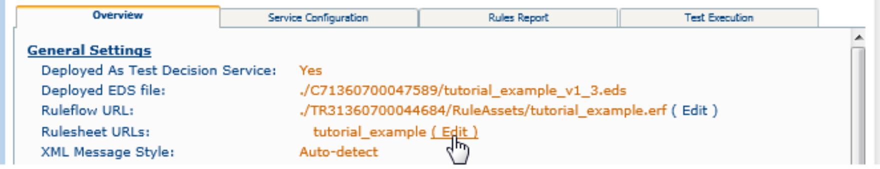
Opens a panel that defines a new major version of the Decision Service which you can edit.

- **Modify Current Version** - Click the button, as shown:



Opens a panel that defines a new minor version of the Decision Service which you can edit.

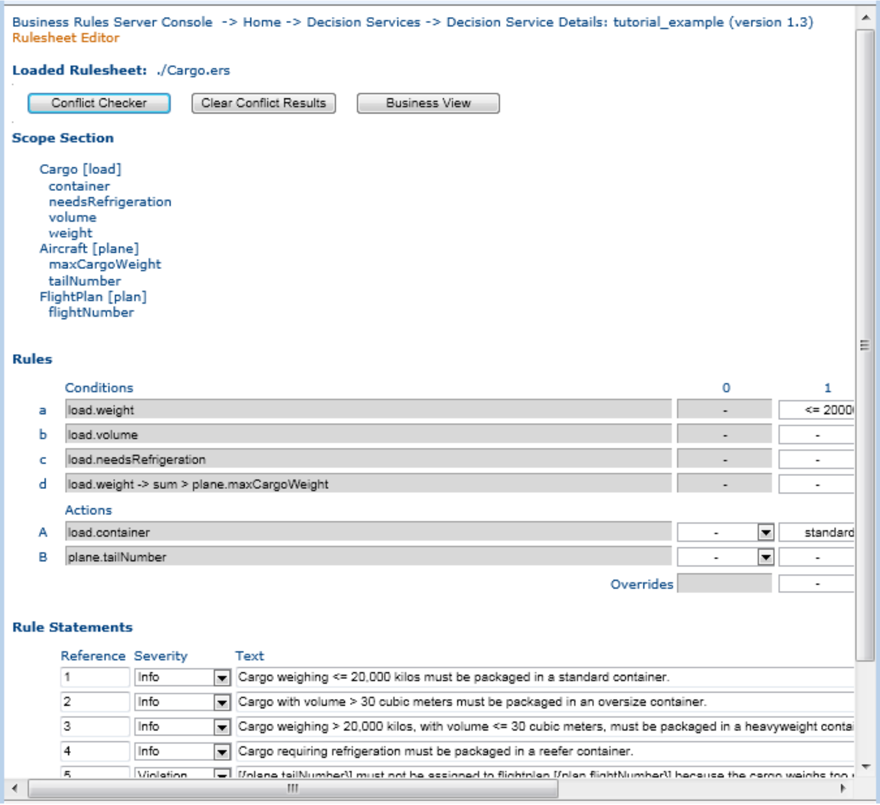
Click on an editable item to open its associated asset.



General Settings

Deployed As Test Decision Service:	Yes
Deployed EDS file:	./C71360700047589/tutorial_example_v1_3.eds
Ruleflow URL:	./TR31360700044684/RuleAssets/tutorial_example.erf (Edit)
Rulesheet URLs:	tutorial_example (Edit)
XML Message Style:	Auto-detect

On the selected asset -- a Rulesheet in this example -- adjust selected parameters, click to check for conflicts, and see the alternative Business View.



Business Rules Server Console -> Home -> Decision Services -> Decision Service Details: tutorial_example (version 1.3)
Rulesheet Editor

Loaded Rulesheet: ./Cargo.ers

Scope Section

Cargo [load]
container
needsRefrigeration
volume
weight
Aircraft [plane]
maxCargoWeight
tailNumber
FlightPlan [plan]
flightNumber

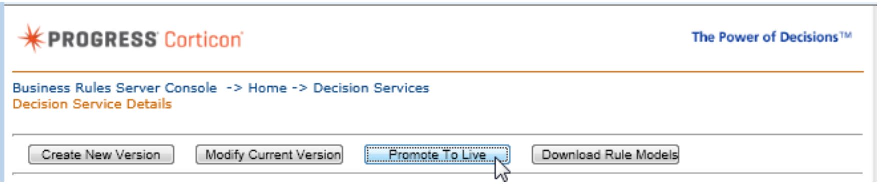
Rules

	Conditions	0	1
a	load.weight	-	<= 2000
b	load.volume	-	-
c	load.needsRefrigeration	-	-
d	load.weight -> sum > plane.maxCargoWeight	-	-
	Actions		
A	load.container	-	standard
B	plane.tailNumber	-	-
	Overrides		-

Rule Statements

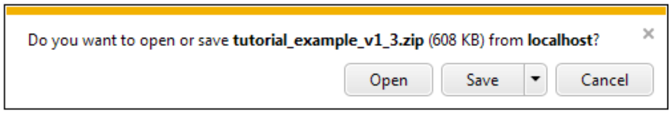
Reference	Severity	Text
1	Info	Cargo weighing <= 20,000 kilos must be packaged in a standard container.
2	Info	Cargo with volume > 30 cubic meters must be packaged in an oversize container.
3	Info	Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be packaged in a heavyweight contai
4	Info	Cargo requiring refrigeration must be packaged in a reefer container.
5	Violation	[{plane.tailNumber}] must not be assigned to flightplan [{plan.flightNumber}] because the cargo weighs too

- **Promote To Live -** Click the button, as shown to promote the changed service to live.:



Business Rules Server Console -> Home -> Decision Services
Decision Service Details

- **Download Rule Models -** Click the button to package the decision service and its assets into a .zip archive file.



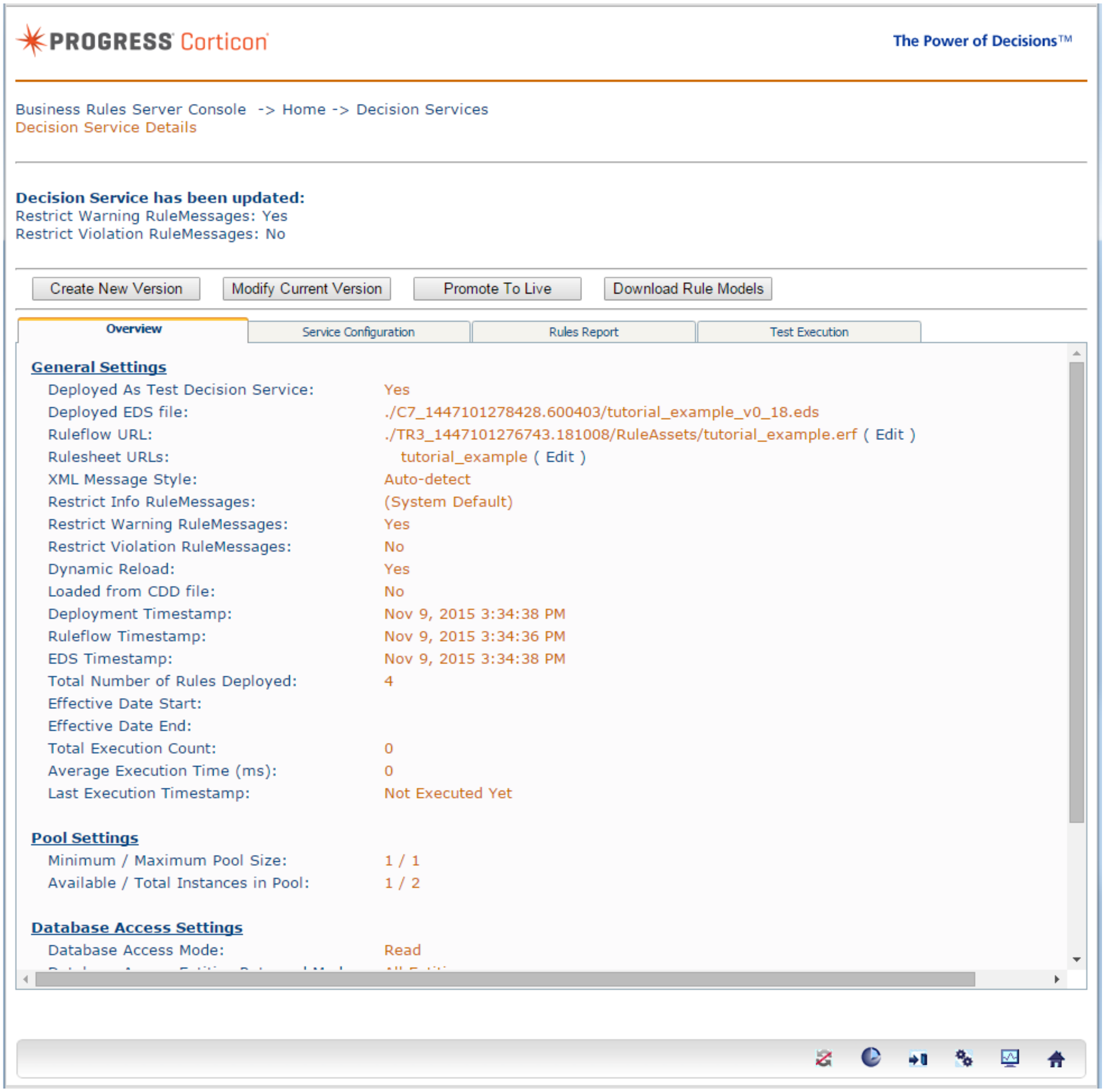
Do you want to open or save **tutorial_example_v1_3.zip** (608 KB) from **localhost**?

TABS

Overview tab

The selected decision service, `tutorial_example`, opens to its **Overview** tab, as shown:

Figure 19: Overview Tab



Business Rules Server Console -> Home -> Decision Services
Decision Service Details

Decision Service has been updated:
Restrict Warning RuleMessages: Yes
Restrict Violation RuleMessages: No

General Settings

Deployed As Test Decision Service:	Yes
Deployed EDS file:	./C7_1447101278428.600403/tutorial_example_v0_18.eds
Ruleflow URL:	./TR3_1447101276743.181008/RuleAssets/tutorial_example.erf (Edit)
Rulesheet URLs:	tutorial_example (Edit)
XML Message Style:	Auto-detect
Restrict Info RuleMessages:	(System Default)
Restrict Warning RuleMessages:	Yes
Restrict Violation RuleMessages:	No
Dynamic Reload:	Yes
Loaded from CDD file:	No
Deployment Timestamp:	Nov 9, 2015 3:34:38 PM
Ruleflow Timestamp:	Nov 9, 2015 3:34:36 PM
EDS Timestamp:	Nov 9, 2015 3:34:38 PM
Total Number of Rules Deployed:	4
Effective Date Start:	
Effective Date End:	
Total Execution Count:	0
Average Execution Time (ms):	0
Last Execution Timestamp:	Not Executed Yet

Pool Settings

Minimum / Maximum Pool Size:	1 / 1
Available / Total Instances in Pool:	1 / 2

Database Access Settings

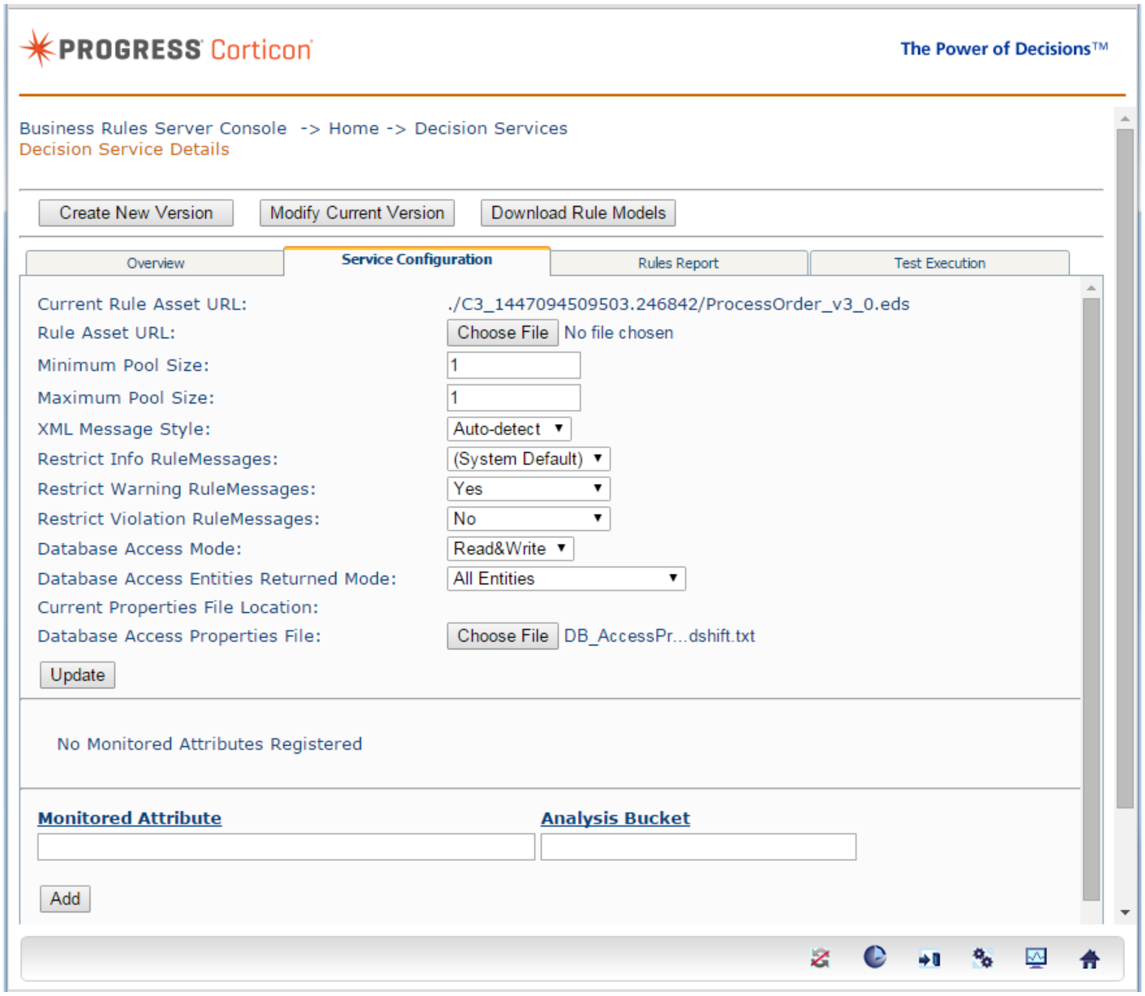
Database Access Mode:	Read

The **Overview** tab lets Server Console users with write permissions to perform limited management and updates to the deployed Ruleflows and constituent Rulesheets. See the *Rule Modeling Guide* for details on editing those Corticon assets.

Service Configuration tab

Click the **Service Configuration** tab to open that panel, as shown:

Figure 20: Service Configuration Tab



Current Rule Asset URL: ./C3_1447094509503.246842/ProcessOrder_v3_0.eds
Rule Asset URL: Choose File No file chosen
Minimum Pool Size: 1
Maximum Pool Size: 1
XML Message Style: Auto-detect
Restrict Info RuleMessages: (System Default)
Restrict Warning RuleMessages: Yes
Restrict Violation RuleMessages: No
Database Access Mode: Read&Write
Database Access Entities Returned Mode: All Entities
Current Properties File Location:
Database Access Properties File: Choose File DB_AccessPr...dshift.txt

No Monitored Attributes Registered

Monitored Attribute
Analysis Bucket

This panel has two sections:

Update the Decision Service's Deployment Properties

If the Decision Service was deployed by the **Deploy Decision Service** page, then the deployment properties can be changed on the **Service Configuration** tab using the **Update** button. Enter your changes, and then click **Update** to re-deploy the changes to the Decision Service.

If the selected Decision Service was deployed by a .cdd file, then deployment properties cannot be changed on this tab – they must be changed in the .cdd file itself. See the topics in *"Packaging and deploying Decision Services" in the Integration and Deployment Guide* for more information about .cdd files and the Deployment Console. This function is accessible to admin users.

Monitored Attribute and Analysis Buckets

Server Console lets you monitor specific attributes in a deployed Decision Service. By choosing attributes to monitor, you can view the statistical breakdown of attribute values over the course of many Decision Service executions.

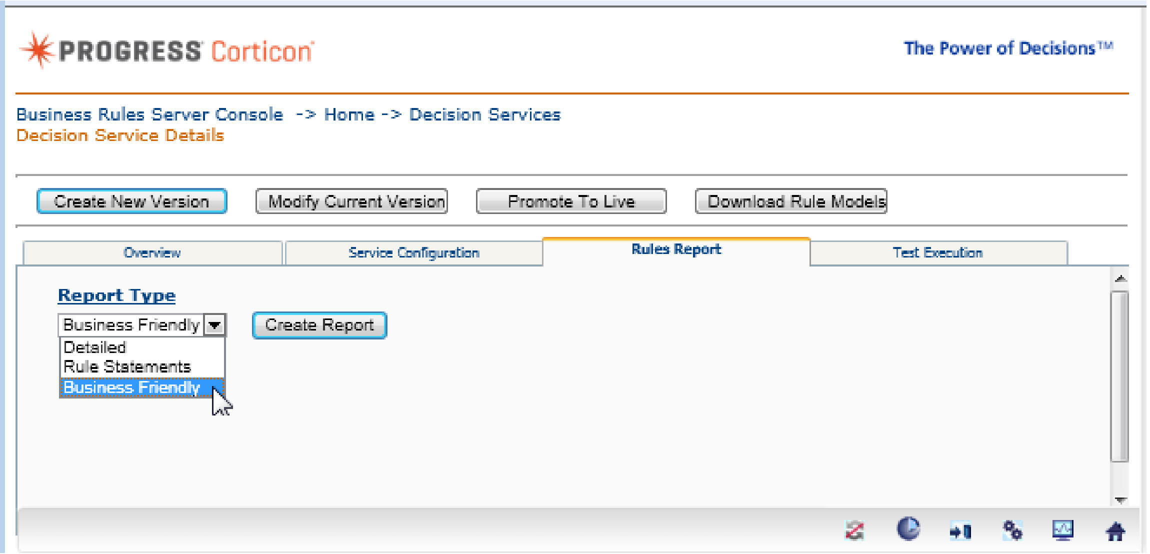
For example, the Ruleflow created in the Tutorial: Basic Rule Modeling in Corticon Studio assigns values such as oversize and reefer to attribute Cargo.container. To monitor this attribute, enter the fully qualified attribute name (including the entity) in the **Monitored Attribute** box and enter the exact value or values in the **Analysis Buckets** box (separated by a comma).

After entering your attribute and its values, click **Add** to append the data to the monitored list. The results of attribute monitoring can be viewed in the Distribution Chart tab, described below.

Rules Report Tab

Click the **Rules Report** tab to open that panel, as shown:

Figure 21: Rules Report Tab

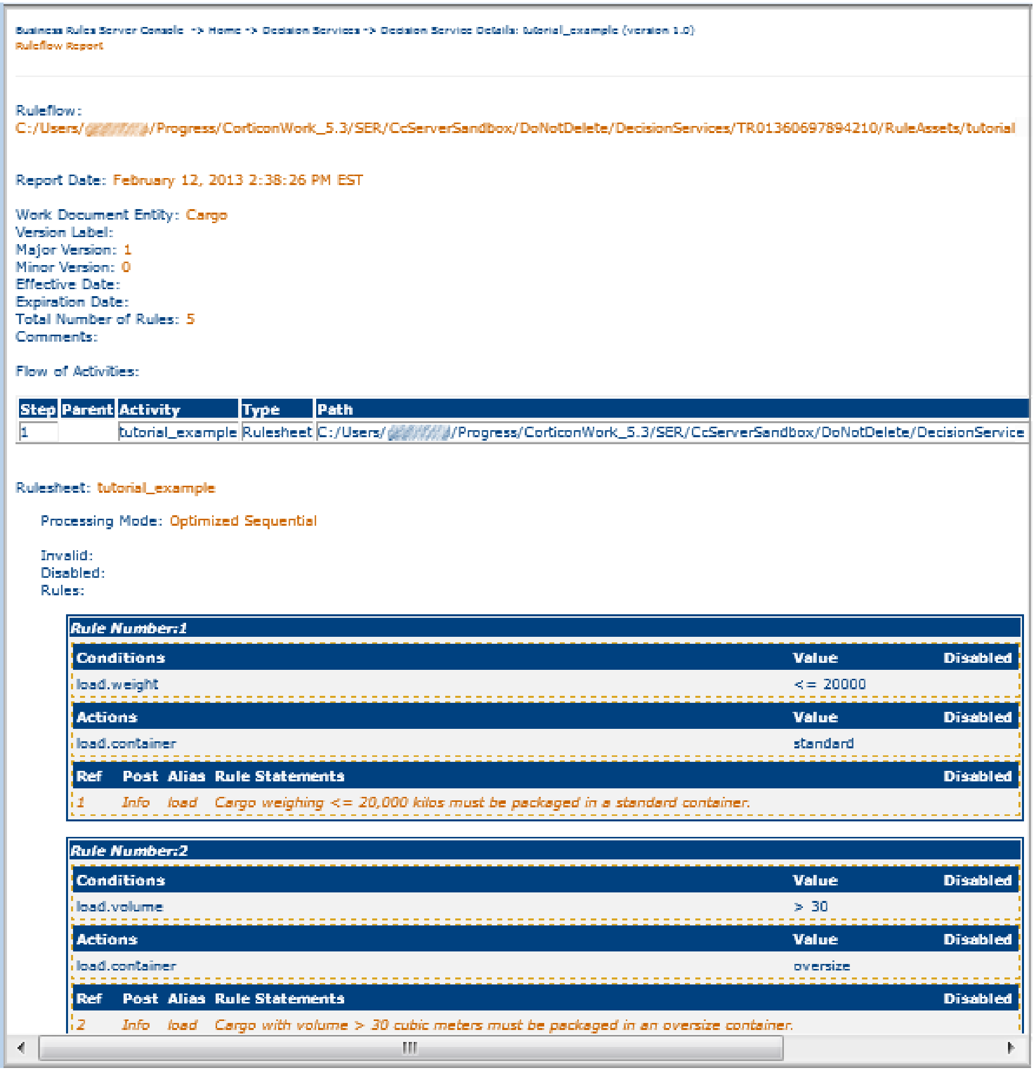


Rulesheet reports can be generated directly from this tab. See the *Rule Modeling Guide* for more information about Corticon's reporting framework. Its options are:

- **Detailed** reports display all elements of the Rulesheet, including Scope, Filters, and other sections.
- **Rule Statements** reports display only those Rule Statements entered for rules. If a rule has no Rule Statement, then that rule will not appear in this report style.
- **Business Friendly** reports display the Natural Language equivalents (if present) of the rules.

The following figure is a Business-Friendly report on the selected decision service:

Figure 22: Business Friendly Report



Business Rules Server Console -> Home -> Decision Services -> Decision Service Details: tutorial_example (version 1.0)
Ruleflow Report

Ruleflow :
C:/Users/[redacted]/Progress/CorticonWork_5.3/SER/CcServerSandbox/DoNotDelete/DecisionServices/TR01360697894210/RuleAssets/tutorial

Report Date: February 12, 2013 2:38:26 PM EST

Work Document Entity: Cargo
Version Label:
Major Version: 1
Minor Version: 0
Effective Date:
Expiration Date:
Total Number of Rules: 5
Comments:

Flow of Activities:

Step	Parent	Activity	Type	Path
1		tutorial_example	Rulesheet	C:/Users/[redacted]/Progress/CorticonWork_5.3/SER/CcServerSandbox/DoNotDelete/DecisionService

Rulesheet: tutorial_example

Processing Mode: Optimized Sequential

Invalid:
Disabled:
Rules:

Rule Number:1

Conditions	Value	Disabled
load.weight	<= 20000	

Actions	Value	Disabled
load.container	standard	

Ref	Post	Alias	Rule Statements	Disabled
1	*Info*	*load*	*Cargo weighing <= 20,000 kilos must be packaged in a standard container.*	

Rule Number:2

Conditions	Value	Disabled
load.volume	> 30	

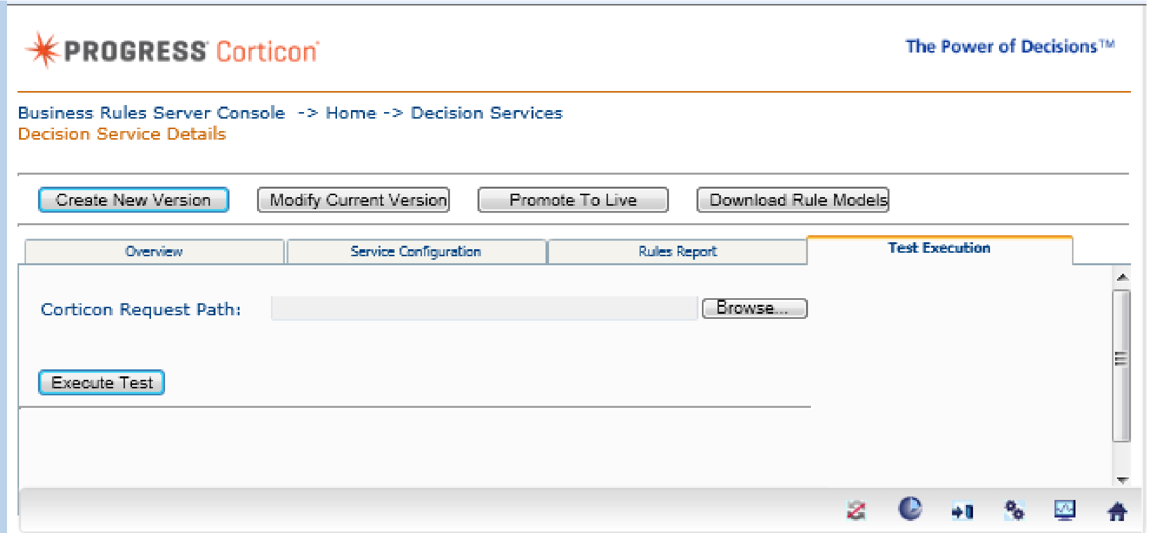
Actions	Value	Disabled
load.container	oversize	

Ref	Post	Alias	Rule Statements	Disabled
2	*Info*	*load*	*Cargo with volume > 30 cubic meters must be packaged in an oversize container.*	

Test Execution Tab

Click the **Test Execution** tab to open that panel, as shown:

Figure 23: Test Execution Tab



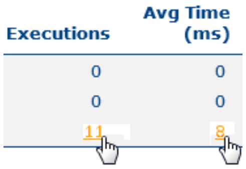
Business Rules Server Console -> Home -> Decision Services
Decision Service Details

Corticon Request Path:

If you have a compliant Corticon Request XML message file, then you can use this tab to send it to Corticon Server, and then view its Response. Navigate to the XML message file using the **Browse…** button, then click **Execute Test** to invoke Corticon Server. To be compliant, your XML Request message must follow the same rules as described in Path 2 in the *Corticon Server: Deploying Web Services*.

Execution and Distribution Visualizations

To access the visualizations of execution and distribution data, click on a decision service line's Execution or Avg Time value, as shown:

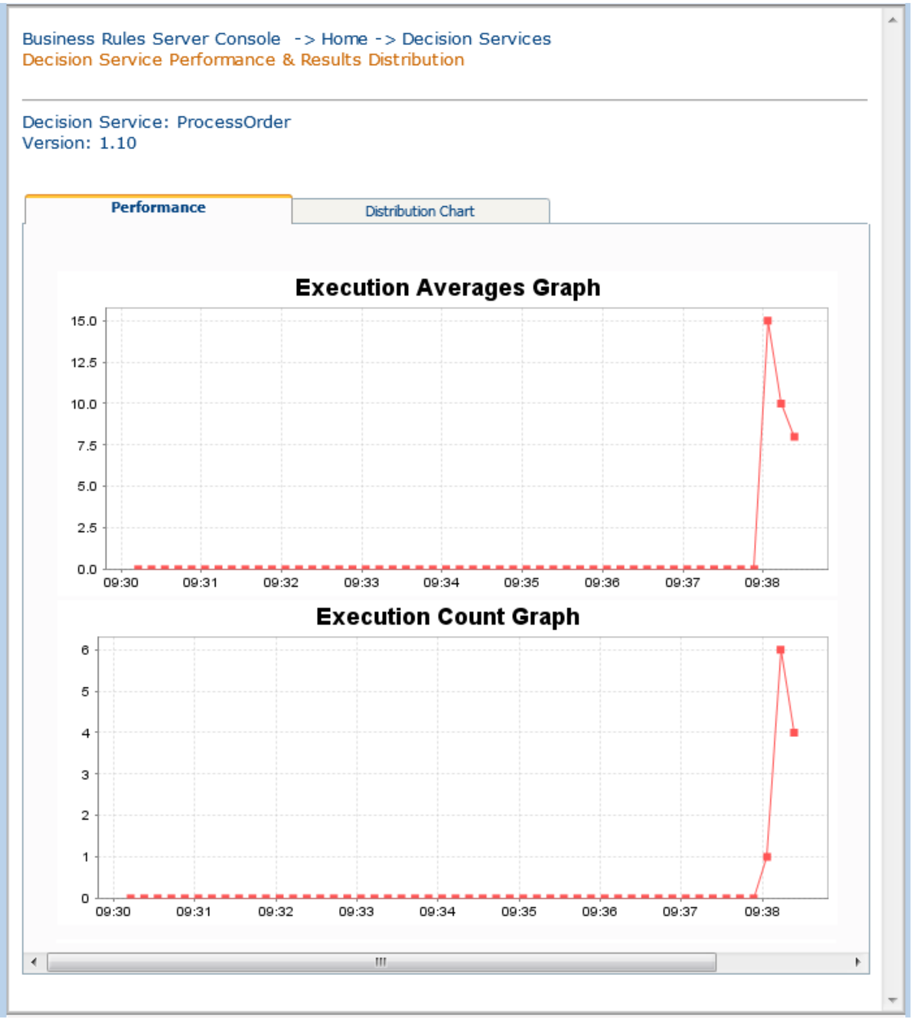


Executions	Avg Time (ms)
0	0
0	0
11	8

Performance Tab

The selected Decision Service records the number of executions since statistics were last cleared, and then computes the average time in milliseconds. The **Executions** tabs are:

Figure 24: Performance Tab: Execution Average Time & Count Graphs



Business Rules Server Console -> Home -> Decision Services
Decision Service Performance & Results Distribution

Decision Service: ProcessOrder
Version: 1.10

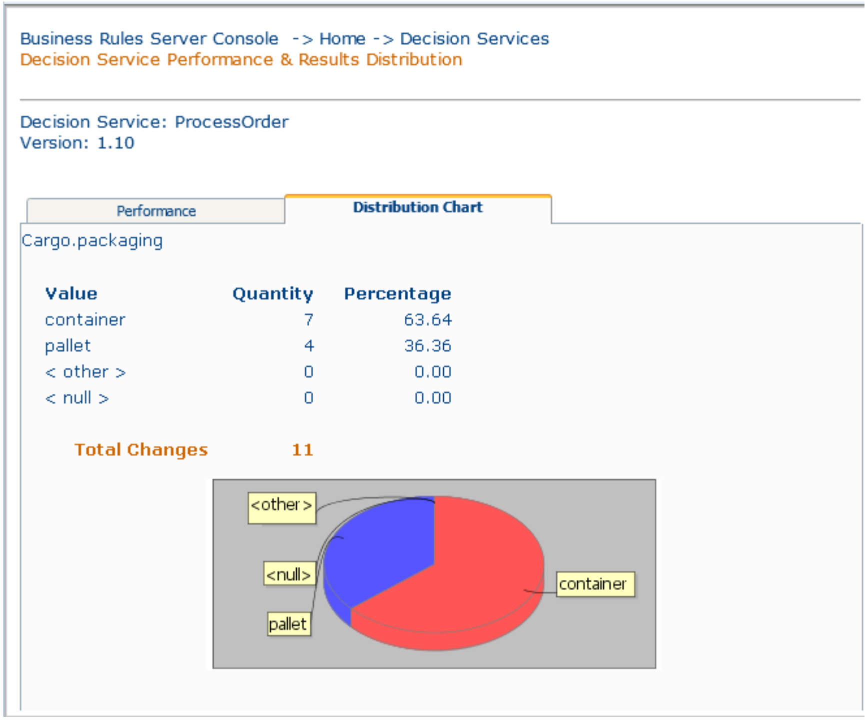
The Decision Service Performance Monitoring Service and Decision Service Time Interval Performance Analysis Service must be **On** for this feature to work. To turn these services **On**, go to the **Configure Decision Service** page, **Decision Service Options** tab.

The top graph displays average execution time for all executions performed within Server Console's recording interval. The lower graph displays the number of executions performed within Server Console's recording interval. Server Console's recording interval is 10 seconds by default in the Server properties, yet can be overidden in your `brms.properties` file.

Distribution Chart Tab

This tab displays a pie chart of the distribution of values of attributes monitored in the Service Configuration tab.

Figure 25: Distribution Chart Tab: Results of Monitored Attributes



Business Rules Server Console -> Home -> Decision Services
Decision Service Performance & Results Distribution

Decision Service: ProcessOrder
Version: 1.10

Performance | Distribution Chart

Cargo.packaging

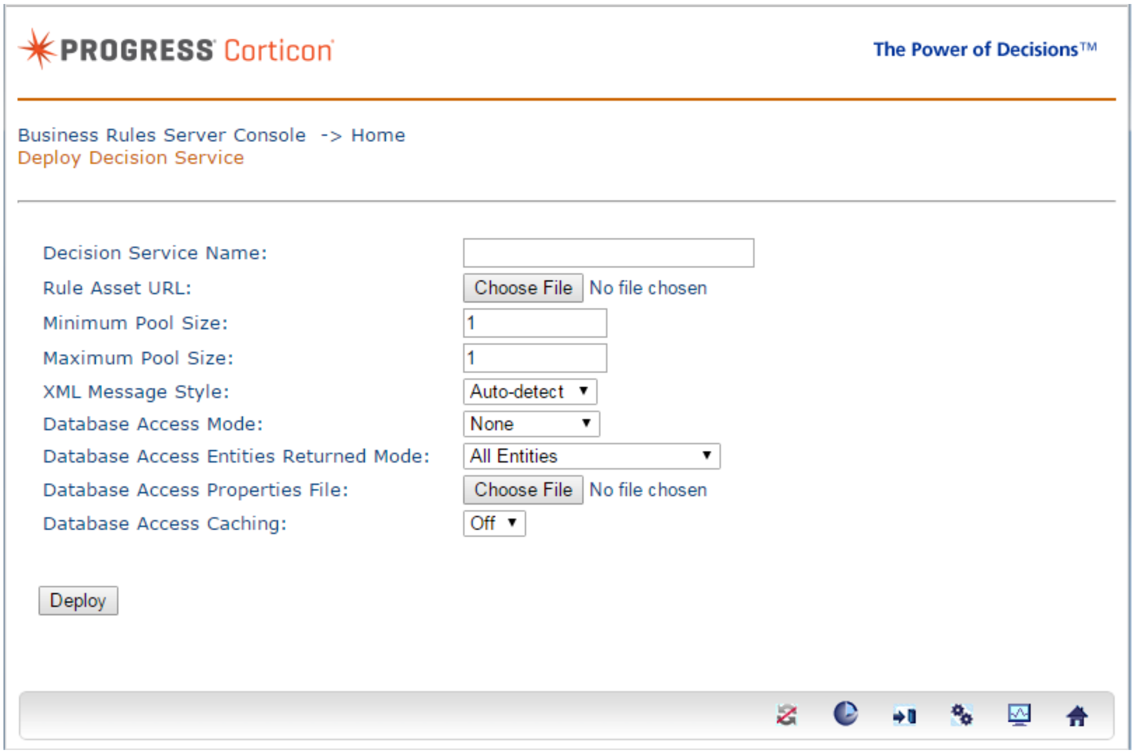
Value	Quantity	Percentage
container	7	63.64
pallet	4	36.36
< other >	0	0.00
< null >	0	0.00
Total Changes	11	

The Decision Service Results Distribution Analysis Service must be **On** for this feature to work. To turn **On**, go to the **Configure Decision Service** page, **Decision Service Options** tab.

Deploy Decision Service page

Click **Deploy Decision Service** on the Home page (or ➡ on the icon bar) to open a page that enables you to deploy Decision Services directly from the Java Server Console. To deploy Decision Services from the Java Server Console, you must first pre-compile the Ruleflow in the Deployment Console to create an `.eds` file.

Figure 26: Deploy Decision Service Page



Decision Service Name:	
Rule Asset URL:	Choose File No file chosen
Minimum Pool Size:	1
Maximum Pool Size:	1
XML Message Style:	Auto-detect ▼
Database Access Mode:	None ▼
Database Access Entities Returned Mode:	All Entities ▼
Database Access Properties File:	Choose File No file chosen
Database Access Caching:	Off ▼

Choose your `.eds` file as the **Rule Asset URL**. Enter the appropriate deployment information, and then click **Deploy**.

Note: These fields are described in Using the Java Server's Deployment Console Decision Services on page 34 and the *Using EDC Guide*.

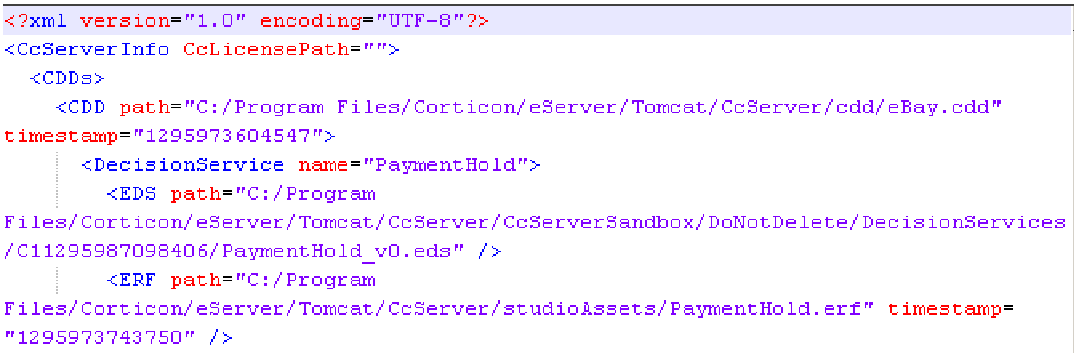
The **Deploy Decision Service** page captures the same deployment information as a `.cdd`, and stores it in `ServerState.xml` file. Corticon Server persists the deployment state of all Decision Services, regardless of how deployed, in its `ServerState.xml` file, located in `[CORTICON_WORK_DIR]\SER\CcServerSandbox\DoNotDelete`. When Corticon Server restarts, it reads the `ServerState.xml` file and redeploys any Decision Services included in it.

To update a Decision Service, you must change the Ruleflow's compiled `.eds` file and then redeploy using the Service Configuration tab's **Update** button. Therefore, a Dynamic Reload setting isn't necessary.

Once deployed, Corticon Server creates a "local" copy of the `.eds` file in its local `CcServerSandbox` directory structure. The location path can be seen in the `ServerState.xml` file in the following figure. It does this so that it always has access to the executable code (stored inside the `.eds` file) when it attempts to reload/redeploy the Decision Service.

If you want to change the Ruleflow, we recommend using the **Update** button on the Service Configuration tab of the **Decision Service** page. We do not recommend editing the local copy of the `.erf` file because the two files will not be identical. Using the `Update` method ensures the original and local copies are always the same.

Figure 27: ServerState.xml showing Local Storage of Deployed Decision Service



```
<?xml version="1.0" encoding="UTF-8"?>
<CcServerInfo CcLicensePath="">
  <CDDs>
    <CDD path="C:/Program Files/Corticon/eServer/Tomcat/CcServer/cdd/eBay.cdd"
timestamp="1295973604547">
      <DecisionService name="PaymentHold">
        <EDS path="C:/Program
Files/Corticon/eServer/Tomcat/CcServer/CcServerSandbox/DoNotDelete/DecisionServices
/C11295987098406/PaymentHold_v0.eds" />
        <ERF path="C:/Program
Files/Corticon/eServer/Tomcat/CcServer/studioAssets/PaymentHold.erf" timestamp=
"1295973743750" />
```

All other entries in the **Deploy Decision Service** page are visible above in the `<NonCDDs>` section of the document.

Configure Rules Server page

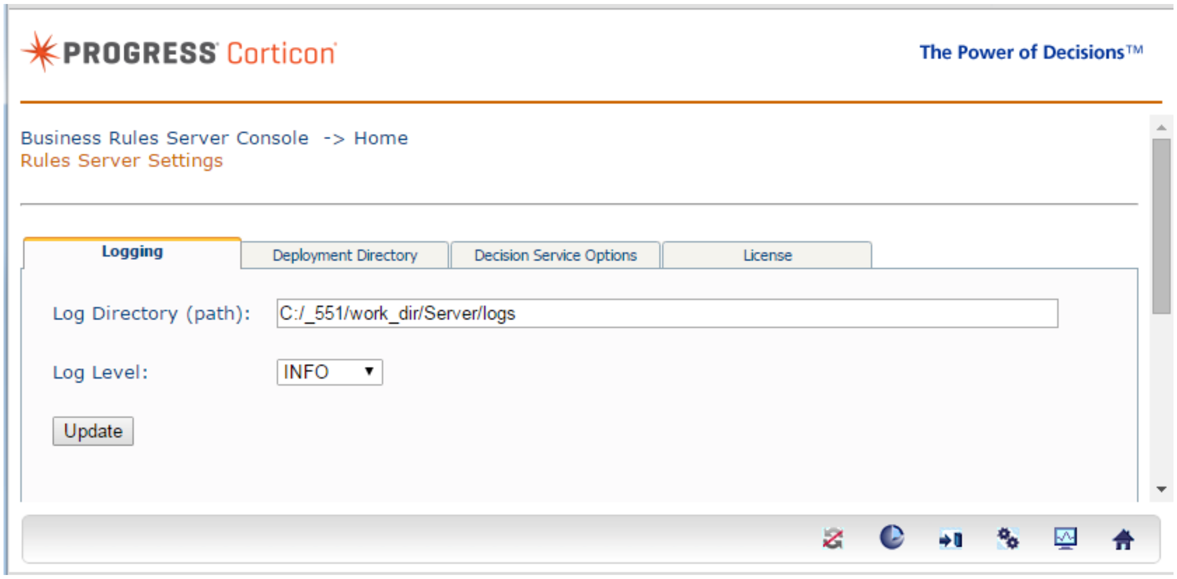
Click **Configure Rules Server** on the Home page (or from the icon bar) to open the configuration page.

Note: Setting these configuration properties in the Server Console (or in corresponding API methods) applies them immediately to the running server, and then persists them to `ServerState.xml` so that they take effect each time Corticon Server is started. These settings apply AFTER your override properties file is loaded. It is recommended that you opt for *"Using the override file, brms.properties" as described in the Integration and Deployment Guide.*

Logging tab

The **Logging** tab displays the current settings of `logpath` and `loglevel` properties, as shown:

Figure 28:



Business Rules Server Console -> Home
Rules Server Settings

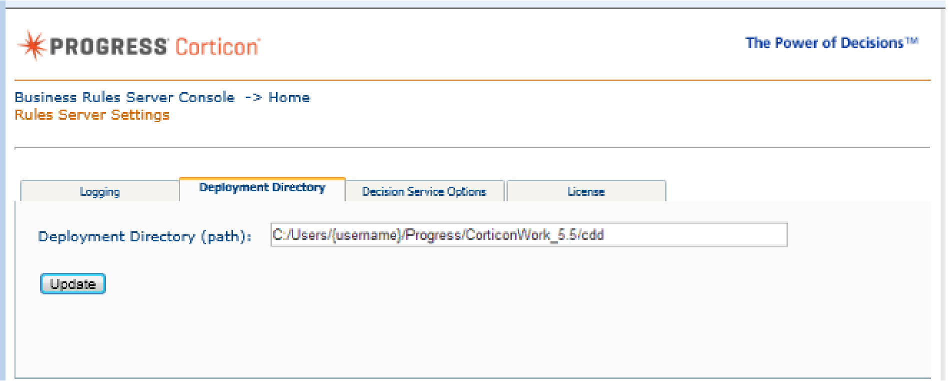
Log Directory (path): C:/_551/work_dir/Server/logs

Log Level: INFO

Deployment Directory tab

The **Deployment Directory** tab displays the current value of the `autoloaddir` property, as shown:

Figure 29: Deployment Directory tab

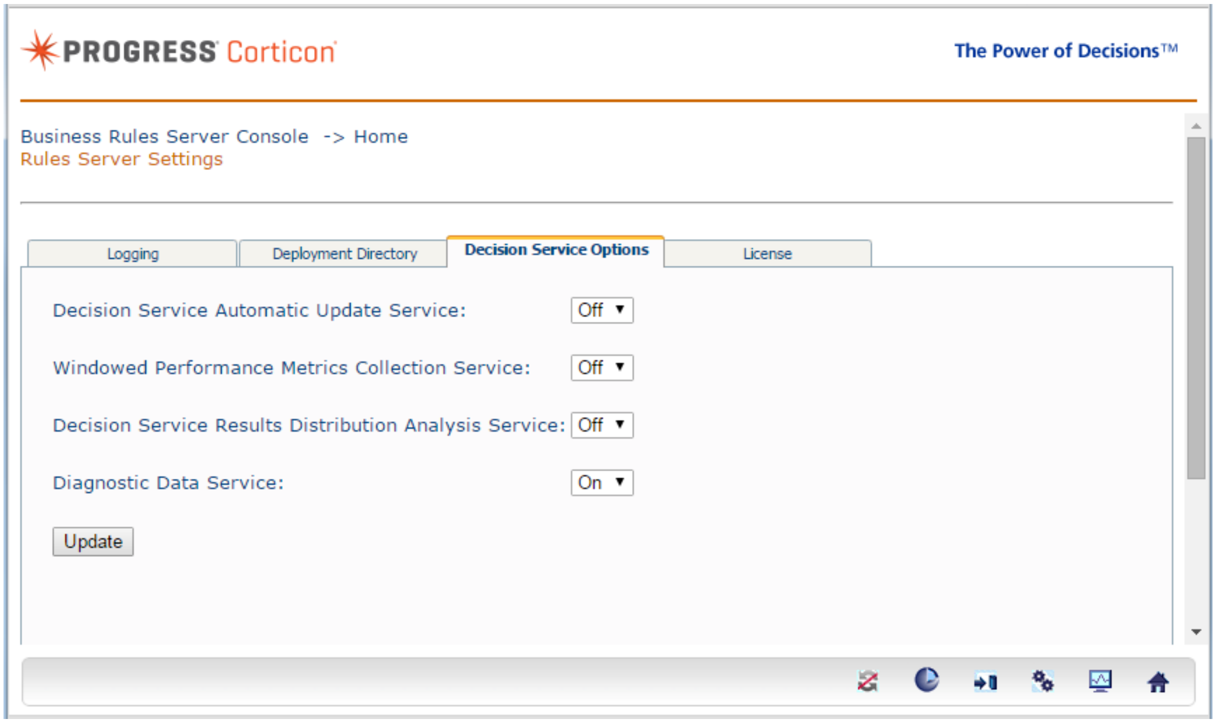


Deployment Directory (path): C:/Users/{username}/Progress/CorticonWork_5.5/cdd

Decision Service Options tab

Server Console lets you enable and disable four Corticon Server services on this tab, as shown:

Figure 30: Decision Service Options tab



Decision Service Automatic Update Service: Off

Windowed Performance Metrics Collection Service: Off

Decision Service Results Distribution Analysis Service: Off

Diagnostic Data Service: On

By default, the first three these settings are **Off**. Set your preferred value (**On** or **Off**) for each feature, and then click **Update** to write the new values to `ServerState.xml`.

Decision Service Automatic Update Service

This service, also known as the Dynamic Update Monitoring Service, is performed by Corticon Server's maintenance thread. The com.corticon.ccserver.dynamicupdatemonitor.autoactivate property in Server properties maintains the permanent value of this property, but it can also be controlled during Corticon Server's session through this Server Console option. The option chosen is recorded in ServerState.xml.

Windowed Performance Metrics Collection Service

This service is responsible for generating the Executions and Avg Time (ms) values displayed on the Decision Services page, as well as the Execution Averages and Execution Count graphs on the Performance tab. The option chosen is recorded in ServerState.xml.

Decision Service Results Distribution Analysis Service

This service is responsible for tracking and recording the attribute values you designated in the **Decision Services** page's Service Configuration tab. The option chosen is recorded in ServerState.xml.

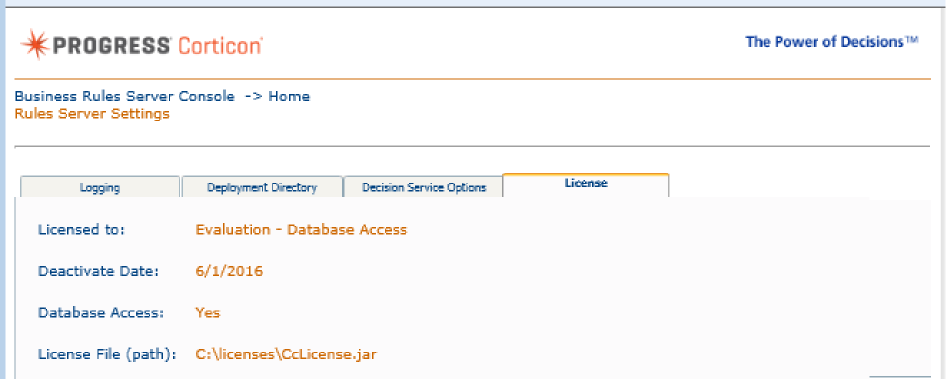
Diagnostic Data Service

This service is responsible for tracking and recording key functions at regular time intervals. The option chosen is recorded in ServerState.xml.

License tab

The Corticon license file at the specified License File (path) is decrypted from the CcLicense.jar currently in use by the running instance of Corticon Server. Three important fields are presented to the user: **Licensed To**, **Deactivate Date**, and **Database Access**, as shown:

Figure 31: License tab



Licensed to: Evaluation - Database Access

Deactivate Date: 6/1/2016

Database Access: Yes

License File (path): C:\licenses\CcLicense.jar

This page is informational only.

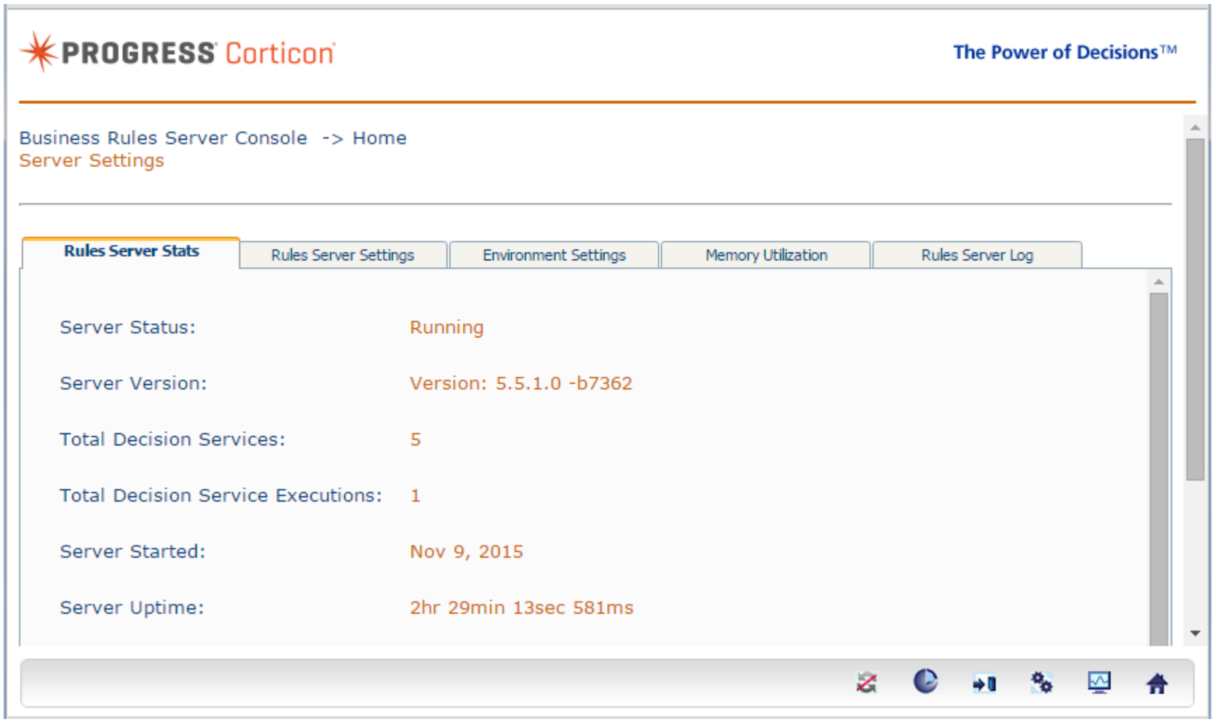
Monitor Rules Server page

Click **Monitor Rules Server** on the Home page (or 🖥 from the icon bar) to open the monitoring page.

Rules Server Stats tab

This tab displays a summary view of basic Corticon Server information, including version/build, total number of Decision Services deployed, session start time, and uptime duration, as shown:

Figure 32: Rules Server Stats tab

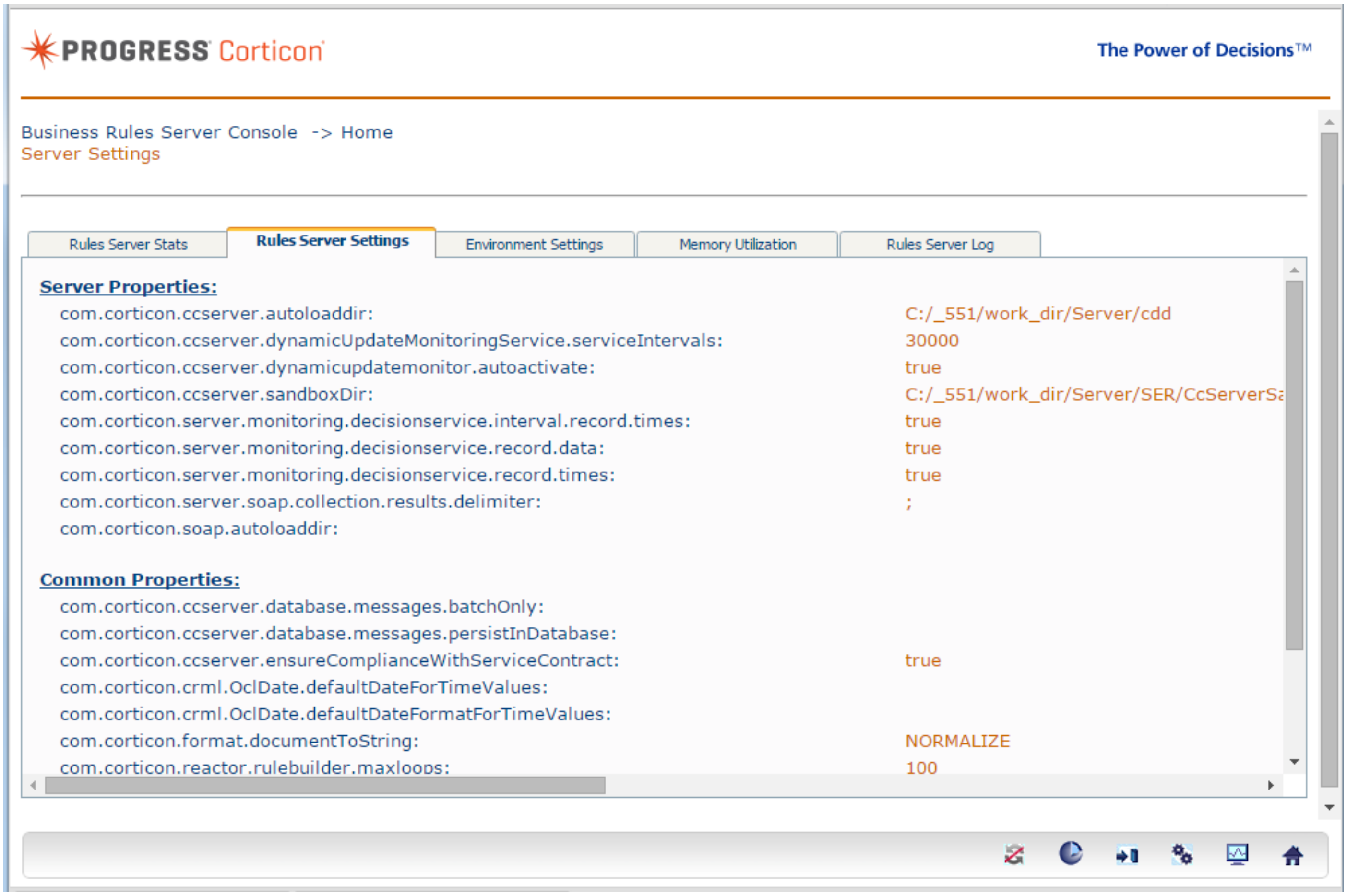


Server Status:	Running
Server Version:	Version: 5.5.1.0 -b7362
Total Decision Services:	5
Total Decision Service Executions:	1
Server Started:	Nov 9, 2015
Server Uptime:	2hr 29min 13sec 581ms

Rules Server Settings tab

This tab displays a view of many of the property settings.

Figure 33: Rules Server Settings tab



Server Properties:

com.corticon.ccserver.autoloaddir:	C:/_551/work_dir/Server/cdd
com.corticon.ccserver.dynamicUpdateMonitoringService.serviceIntervals:	30000
com.corticon.ccserver.dynamicupdatemonitor.autoactivate:	true
com.corticon.ccserver.sandboxDir:	C:/_551/work_dir/Server/SER/CcServerSa
com.corticon.server.monitoring.decisionservice.interval.record.times:	true
com.corticon.server.monitoring.decisionservice.record.data:	true
com.corticon.server.monitoring.decisionservice.record.times:	true
com.corticon.server.soap.collection.results.delimiter:	;
com.corticon.soap.autoloaddir:	

Common Properties:

com.corticon.ccserver.database.messages.batchOnly:	
com.corticon.ccserver.database.messages.persistInDatabase:	
com.corticon.ccserver.ensureComplianceWithServiceContract:	true
com.corticon.crml.OclDate.defaultDateForTimeValues:	
com.corticon.crml.OclDate.defaultDateFormatForTimeValues:	
com.corticon.format.documentToString:	NORMALIZE
com.corticon.reactor.rulebuilder.maxloops:	100

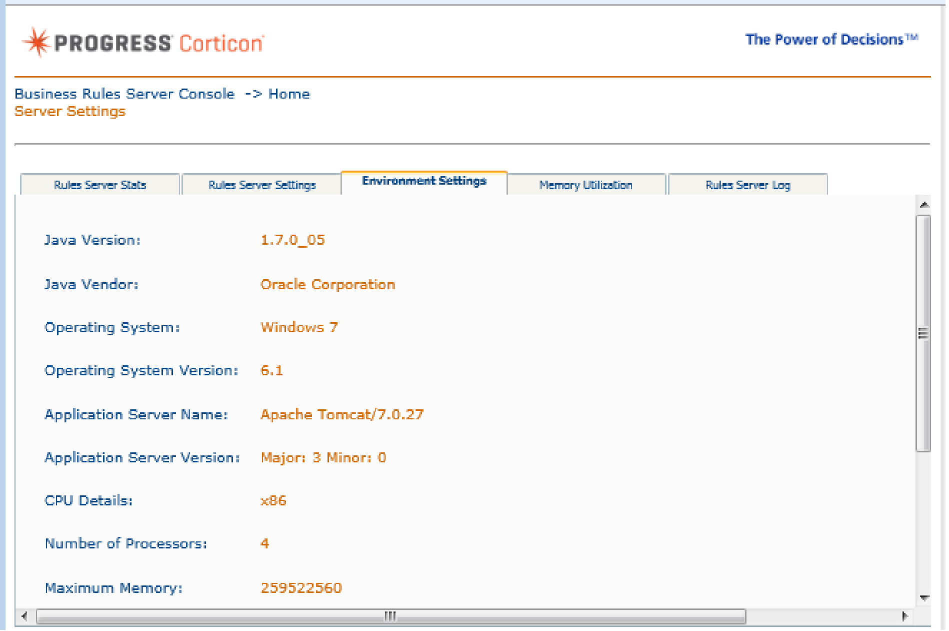
These settings are maintained in `CcConfig.jar` and user overrides are loaded last through the `brms.properties` file. (See *"Configuring Corticon properties and settings" in the Integration and Deployment Guide* for more information about this file). If a property setting (such as `logLevel`) has been overridden by a setting in Server Console (and persisted in `ServerState.xml`), then the Server Console value is displayed.

The list of properties displayed on this tab is controlled by `CcServerConsoleProperties.properties`, which is located in `Server\pas\corticon\webapps\axis\WEB-INF` in your Corticon Server installation directory.

Environment Settings tab

This tab displays basic information about host operating system, application server, and Java virtual machine (JVM), as shown:

Figure 34: Environment Settings tab



Java Version: 1.7.0_05

Java Vendor: Oracle Corporation

Operating System: Windows 7

Operating System Version: 6.1

Application Server Name: Apache Tomcat/7.0.27

Application Server Version: Major: 3 Minor: 0

CPU Details: x86

Number of Processors: 4

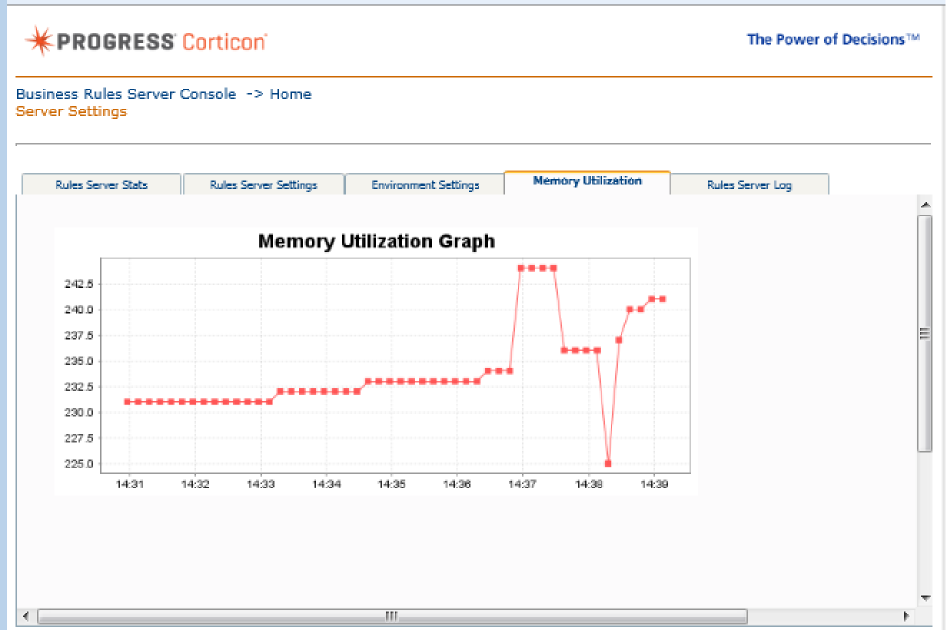
Maximum Memory: 259522560

This information may be useful to Progress technical support in the event you need to contact us.

Memory Utilization tab

This tab displays a graph of JVM memory utilization, as shown:

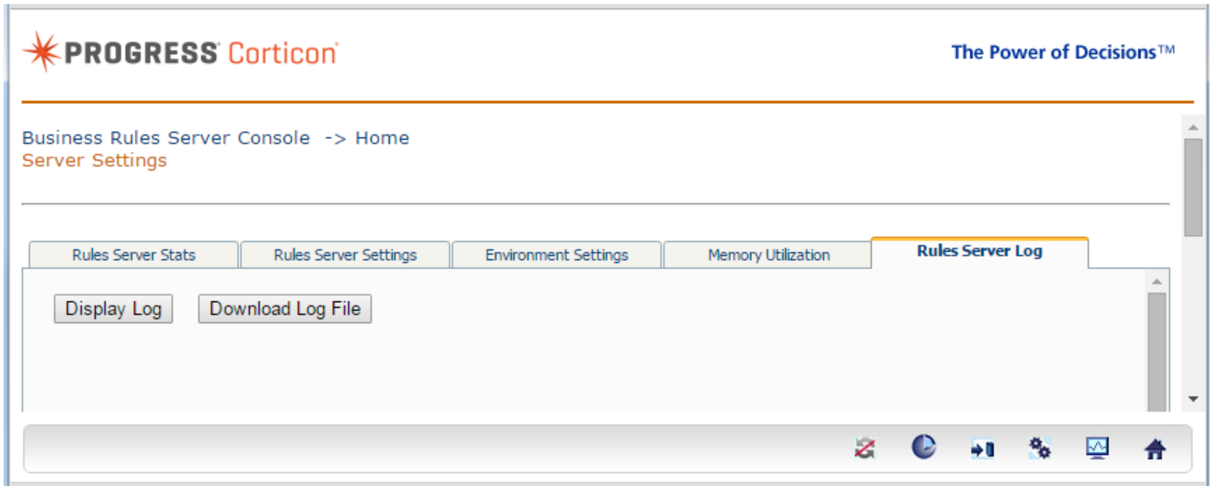
Figure 35: Memory Utilization tab



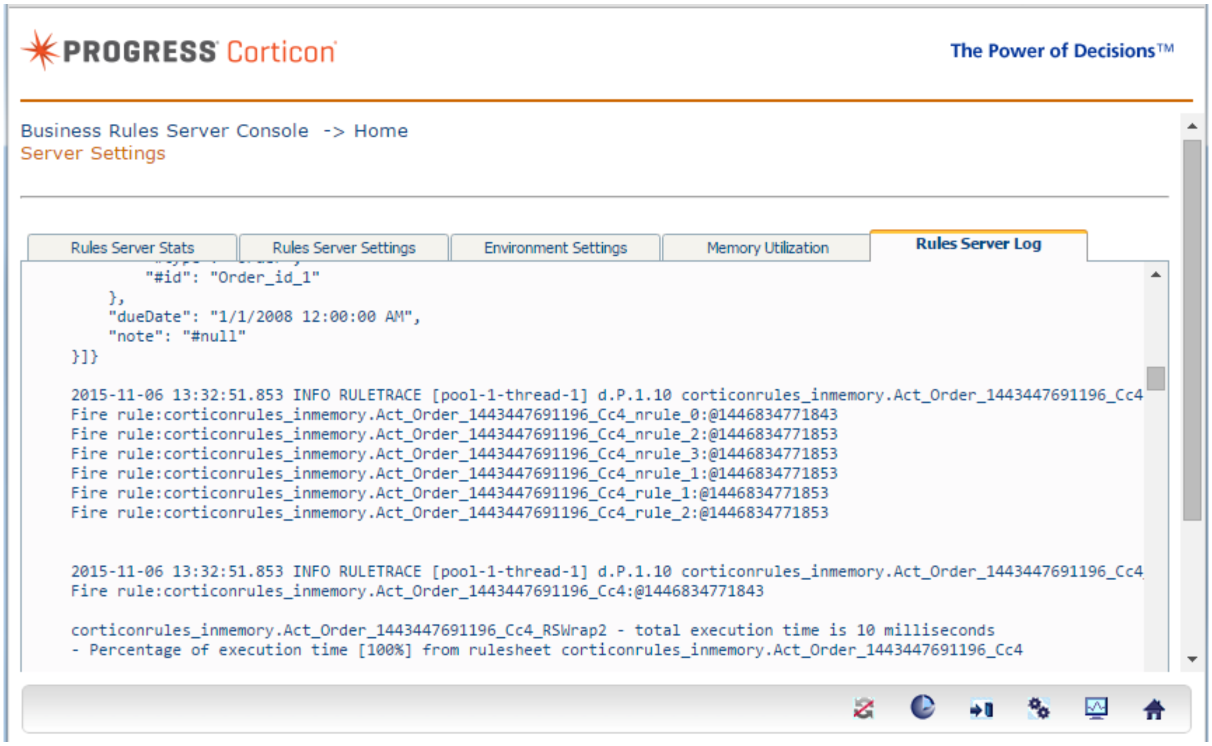
This graph will help you understand if Corticon Server's host environment is under-resourced, and may also be useful to Progress technical support in the event you need to contact us about memory problems.

Rules Server Log tab

This tab reads and displays lines from the current Corticon Server log file, as shown:



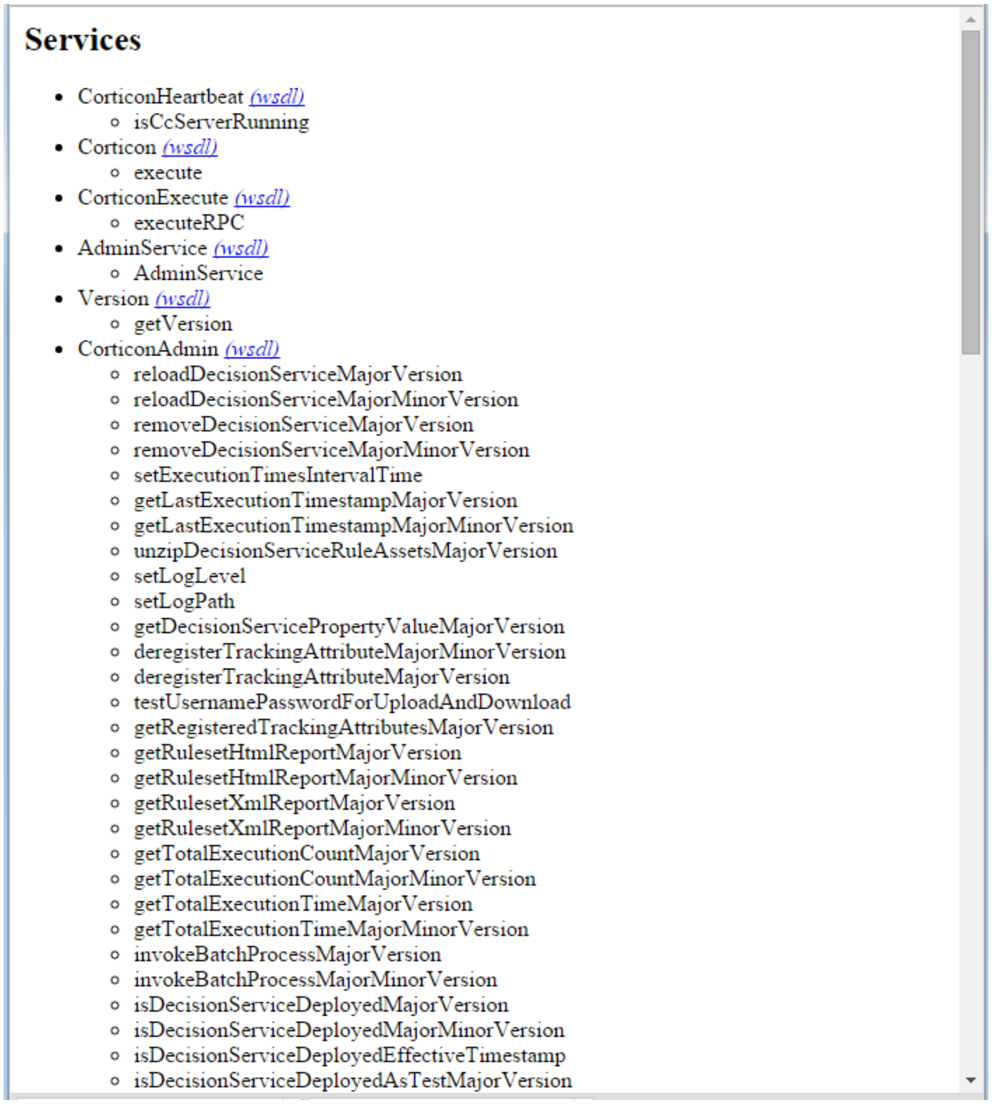
This tab lets you read recent log entries, as illustrated:



```
            "#id": "Order_id_1"
        },
        "dueDate": "1/1/2008 12:00:00 AM",
        "note": "#null"
    }]}

    2015-11-06 13:32:51.853 INFO RULETRACE [pool-1-thread-1] d.P.1.10 corticonrules_inmemory.Act_Order_1443447691196_Cc4
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_nrule_0:@1446834771843
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_nrule_2:@1446834771853
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_nrule_3:@1446834771853
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_nrule_1:@1446834771853
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_rule_1:@1446834771853
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4_rule_2:@1446834771853


    2015-11-06 13:32:51.853 INFO RULETRACE [pool-1-thread-1] d.P.1.10 corticonrules_inmemory.Act_Order_1443447691196_Cc4
    Fire rule:corticonrules_inmemory.Act_Order_1443447691196_Cc4:@1446834771843

    corticonrules_inmemory.Act_Order_1443447691196_Cc4_RSWrap2 - total execution time is 10 milliseconds
    - Percentage of execution time [100%] from rulesheet corticonrules_inmemory.Act_Order_1443447691196_Cc4
```

WSDLs

This page provides access to WSDL files for all Corticon APIs, as shown:



Services

- CorticonHeartbeat (wsdl)
  - isCcServerRunning
- Corticon (wsdl)
  - execute
- CorticonExecute (wsdl)
  - executeRPC
- AdminService (wsdl)
  - AdminService
- Version (wsdl)
  - getVersion
- CorticonAdmin (wsdl)
  - reloadDecisionServiceMajorVersion
  - reloadDecisionServiceMajorMinorVersion
  - removeDecisionServiceMajorVersion
  - removeDecisionServiceMajorMinorVersion
  - setExecutionTimesIntervalTime
  - getLastExecutionTimestampMajorVersion
  - getLastExecutionTimestampMajorMinorVersion
  - unzipDecisionServiceRuleAssetsMajorVersion
  - setLogLevel
  - setLogPath
  - getDecisionServicePropertyValueMajorVersion
  - deregisterTrackingAttributeMajorMinorVersion
  - deregisterTrackingAttributeMajorVersion
  - testUsernamePasswordForUploadAndDownload
  - getRegisteredTrackingAttributesMajorVersion
  - getRulesetHtmlReportMajorVersion
  - getRulesetHtmlReportMajorMinorVersion
  - getRulesetXmlReportMajorVersion
  - getRulesetXmlReportMajorMinorVersion
  - getTotalExecutionCountMajorVersion
  - getTotalExecutionCountMajorMinorVersion
  - getTotalExecutionTimeMajorVersion
  - getTotalExecutionTimeMajorMinorVersion
  - invokeBatchProcessMajorVersion
  - invokeBatchProcessMajorMinorVersion
  - isDecisionServiceDeployedMajorVersion
  - isDecisionServiceDeployedMajorMinorVersion
  - isDecisionServiceDeployedEffectiveTimestamp
  - isDecisionServiceDeployedAsTestMajorVersion

You can access this page directly at the address:
`http://<yourServer>:<yourPort>/axis/servlet/AxisServlet`

Lifecycle management through the Java Server Console

Modifying rules within "live" Decision Services already deployed to Corticon Server requires more considerations than just updating rules in Progress Corticon Studio. In Progress Corticon Studio, the rules in Rulesheets are still just design-time assets, perhaps not even tested yet or packaged in a Ruleflow, let alone deployed to a Server.

But rules in a live Decision Service are available for invocation *right at that moment*, so we need to take a few extra precautions to ensure we do not interfere with clients trying to use our deployed Decision Services.

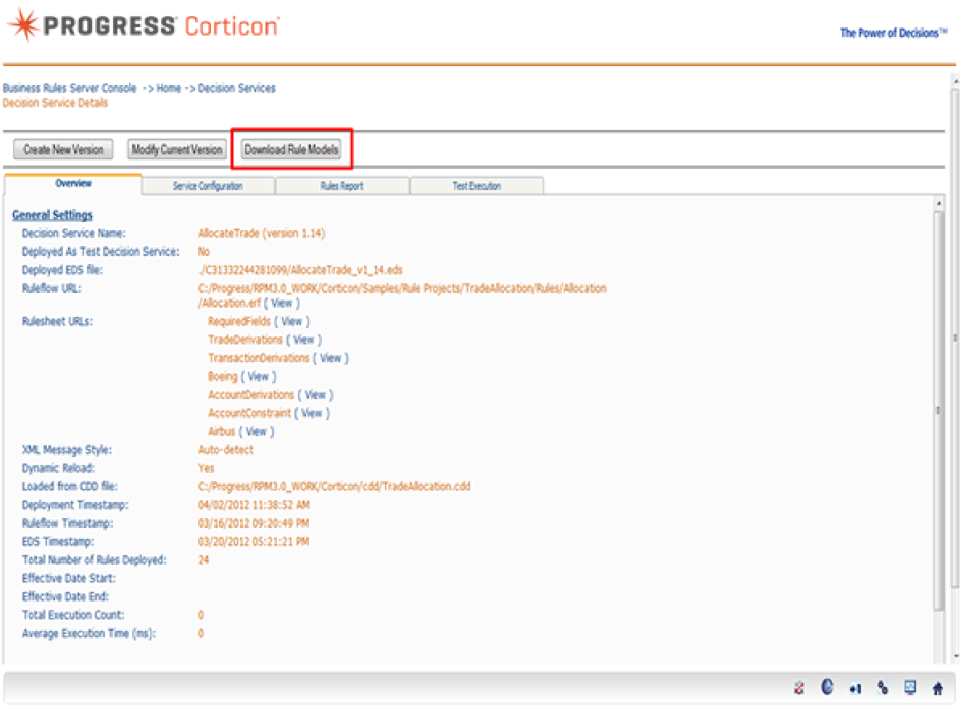
Using the Java Server Console to Download Decision Services

You can use the Corticon Server Console to download Decision Services. Click the **Download Rule Models** button to download a ZIP file to your local machine, which contains all the assets belonging to a Decision Service. The ZIP file preserves the folder structure and the relative relationships between the assets. The ZIP file can be extracted to the Eclipse workspace, and the assets can be opened and edited without any errors.

To download Decision Services using the Server Console:

1. In a browser, connect to the Corticon Server Console (`http://localhost:8850/axis`). Log in.
2. Click **Decision Services** to show the list of all the deployed Decision Services.
3. Click the Decision Service name that you want to download.
4. Click **Download Rule Models**, as shown:

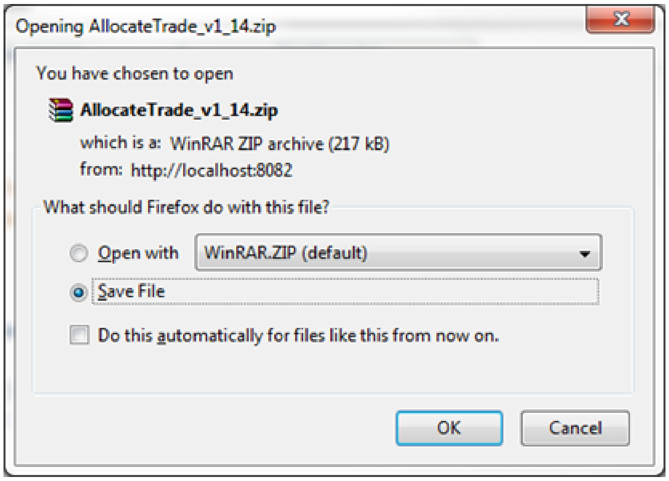
   Figure 36: Downloading Decision Service from the Server Console



Decision Service Name:	AllocateTrade (version 1.14)
Deployed As Test Decision Service:	No
Deployed EDS file:	./C31332244281099/AllocateTrade_v1_14.eds
Ruleflow URL:	C:/Progress/RPM3.0_WORK/Corticon/Samples/Rule Projects/TradeAllocation/Rules/Allocation /Allocation.erf (View)
Rulesheet URLs:	RequiredFields (View) TradeDerivations (View) TransactionDerivations (View) Boeing (View) AccountDerivations (View) AccountConstraint (View) Airbus (View)
XML Message Style:	Auto-detect
Dynamic Reload:	Yes
Loaded from CDD file:	C:/Progress/RPM3.0_WORK/Corticon/cdd/TradeAllocation.cdd
Deployment Timestamp:	04/02/2012 11:38:52 AM
Ruleflow Timestamp:	03/16/2012 09:20:49 PM
EDS Timestamp:	03/20/2012 05:21:21 PM
Total Number of Rules Deployed:	24
Effective Date Start:	
Effective Date End:	
Total Execution Count:	0
Average Execution Time (ms):	0

The Decision Service downloads the selected Decision Service to the local machine, packaged as a ZIP file that you can choose to open or save, as shown:

Figure 37: Downloading Decision Services to the local machine



Opening AllocateTrade_v1_14.zip

You have chosen to open

AllocateTrade_v1_14.zip

which is a: WinRAR ZIP archive (217 kB)

from: http://localhost:8082

What should Firefox do with this file?

○ Open with WinRAR.ZIP (default)

● Save File

□ Do this automatically for files like this from now on.

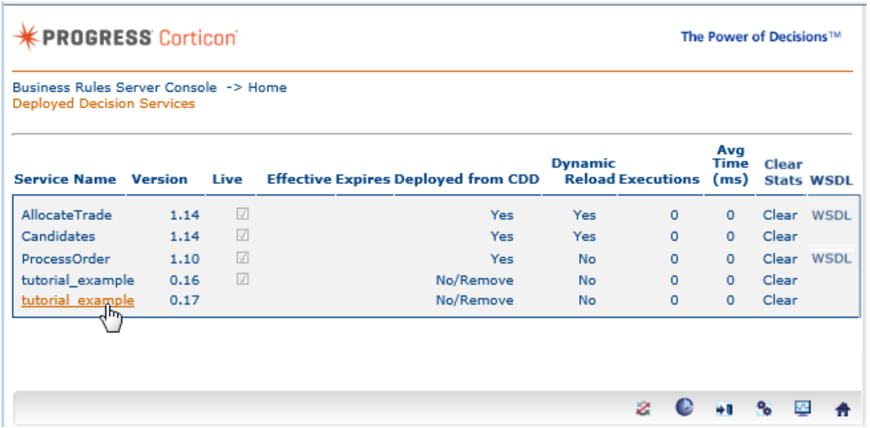
Creating a new decision service version

Before any rule changes can be made from within Server Console, a new version of the Decision Service must be created first. A new version can be created for a Decision Service deployed via a `.cdd`, or for a Decision Service deployed via the Server Console.

To create a new Decision Service version:

1. Login to Server Console
2. Select Decision Services from the Server Console Main Menu
3. Click the name of the Decision Service you want to modify. Each Decision Service listed in the Service Name column is a hyperlink.
4. Select **Create New Version** button at the top of the page. See the Decision Service Versioning and Effective Dating chapter of the *Server Integration & Deployment Guide* for more information about versions and how to use them during Decision Service invocation.
5. You will return to the Decision Services page, where you should see an additional Decision Service listed in the **Service Name** column. The Version should be incremented by 1. In the figure shown below, a new version of `tutorial_example` has been created.

Figure 38: Server Console with a new version of tutorial_example shown



Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats	WSDL
AllocateTrade	1.14	☑			Yes	Yes	0	0	Clear	WSDL
Candidates	1.14	☑			Yes	Yes	0	0	Clear	
ProcessOrder	1.10	☑			Yes	No	0	0	Clear	WSDL
tutorial_example	0.16	☑			No/Remove	No	0	0	Clear	
tutorial_example	0.17				No/Remove	No	0	0	Clear	

Notice in the figure that `tutorial_example` version .17 is not yet "live" (the **Live** column checkbox is unchecked). This means we can make changes to it.

Opening the new version

To make changes to the new version (or any other "non-live" Decision Service):

1. Click on the Decision Service you want in the Service Name column. The **Overview** tab opens.
2. Under the **General Settings** header, select the **(Edit)** hyperlink following the `.erf` or `.ers` you want to modify:
   a) Selecting the **(Edit)** hyperlink following the `.erf` opens a page that lets you modify effective/expiration dates of the Decision Service, or increase its Version number.
   b) Selecting the **(Edit)** hyperlink following a `.ers` opens a page that lets you modify that Rulesheet.

Modifying the new version

Server Console allows the following types of changes to a Rulesheet:

- Changing values in a Condition or Action cell, including creating new values in the existing set
- Adding/removing/changing Overrides for rule columns
- Changing Rule Statements for those rule columns that have them, including Text and Serverity changes.
- Re-applying the Conflict Checker following any changes to ensure new no new conflicts have been introduced by your changes

Server Console does not allow the following types of changes to a Rulesheet:

- Adding/removing/changing any Scope used by any rules
- Adding/removing/changing any Filters used by the Rulesheet
- Adding/removing/changing Condition or Action expressions
- Adding/removing rule columns to the decision table
- Adding/removing Rule Statements
- Adding/removing/changing and looping or advanced Inferencing features in a Rulesheet
- Enabling or disabling any Rulesheet rows or columns

The Rulesheet change page also includes a **Business View/Technical View** toggle button that shows/hides Rulesheet Scope and Filters, if used.

Once you have updated the Rulesheet with any changes, click the **Save** button at the bottom of the page. A message will display the updated Decision Service timestamp and advise that a slight delay may occur before the Decision Service is ready to be promoted to "live" status. This delay is due to the time required to compile the Decision Service "on the fly" by Corticon Server. For more information about Decision Service compilation, see the *Server Integration & Deployment Guide*.

Promoting the new version to live

After saving the new version of the Rulesheet (and after making any other changes to the Ruleflow or other Rulesheets), return to the Decision Services:Overview tab for the new version.

Clicking the **Promote to Live** button causes the new version to deploy to "live" status, and will update the **Live** checkbox on the Decision Services page.

Removing the version from live

To remove a new Decision Service version (or any other Decision Service deployed via the Server Console):

1. Connect to the Server Console with administrative user rights.

   - Username `admin` with its password `admin`
   - Username `administrator` with its password `changeme`

2. Click the **No/Remove** hyperlink in the **Deployed from CDD** column of the Decision Services page.

Telling the Java server where to find Deployment Descriptor files

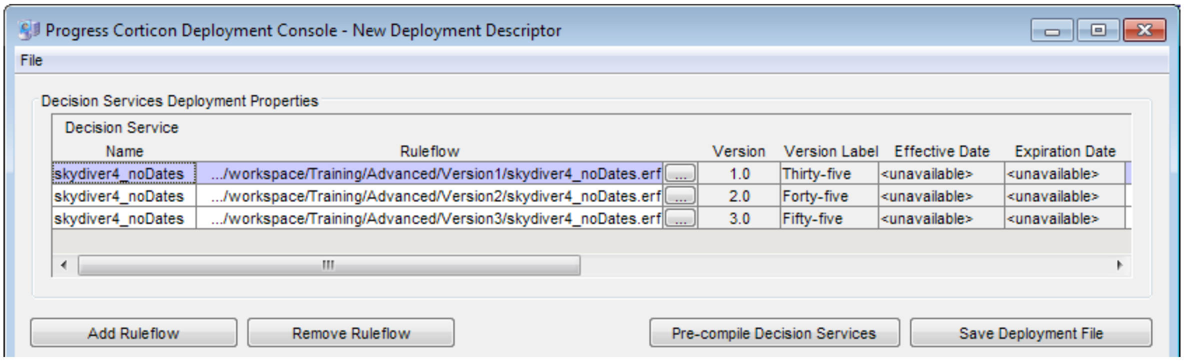
You can do direct deployment of Decision Services without having to use a `.cdd` or code a Java method call. This method of deployment can only be used with pre-compiled `.eds` files, not with uncompiled `.erf` files.

Creating and deploying the sample Decision Services

To create and deploy the sample Ruleflows:

1. Open the Deployment Console, and then add the three Ruleflows we created, all under the same Decision Service name, as shown:

   Figure 39: Deployment Console with the Sample Ruleflows

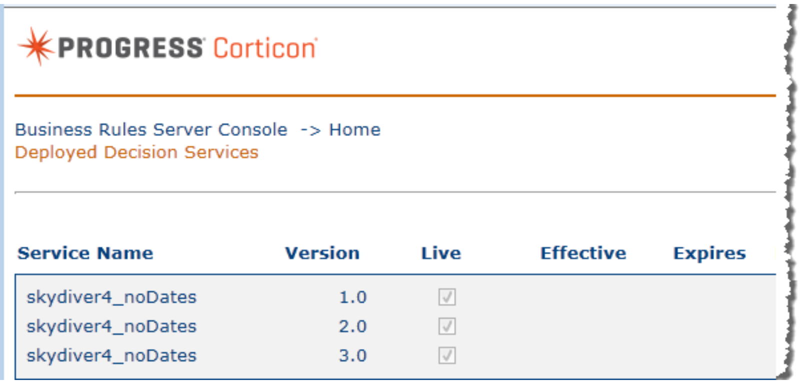
   Progress Corticon Deployment Console - New Deployment Descriptor

   Decision Services Deployment Properties

Decision Service Name	Ruleflow		Version	Version Label	Effective Date	Expiration Date
skydiver4_noDates	.../workspace/Training/Advanced/Version1/skydiver4_noDates.erf	...	1.0	Thirty-five	<unavailable>	<unavailable>
skydiver4_noDates	.../workspace/Training/Advanced/Version2/skydiver4_noDates.erf	...	2.0	Forty-five	<unavailable>	<unavailable>
skydiver4_noDates	.../workspace/Training/Advanced/Version3/skydiver4_noDates.erf	...	3.0	Fifty-five	<unavailable>	<unavailable>

2. Click **Pre-compile Decision Services**, and save the files in a local output folder.
3. Copy the three `.eds` files you created to a network-accessible directory, such as `C:\Compiled_EDS_Files`.
4. Start the Corticon Server.
5. In a browser, go the URL `http://localhost:8850/axis/`, and then log in with the user name and password `admin`.
6. Click **Deploy Decision Services**, and then browse to each of the `.eds` file you staged to add it to the deployment, as shown:

Figure 40: Server Console deploying the Sample Decision Services

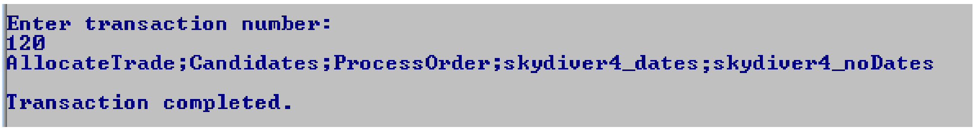


Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires
skydiver4_noDates	1.0	☑		
skydiver4_noDates	2.0	☑		
skydiver4_noDates	3.0	☑		

7. Confirm the deployment, by starting `testServerAxis.bat` (at `[CORTICON_HOME]\Server\bin`, and entering `120 - Get Decision Service Names`. The results are as shown:

   Figure 41: Confirming Deployment of Decision Service skydiver4



```
Enter transaction number:
120
AllocateTrade;Candidates;ProcessOrder;skydiver4_dates;skydiver4_noDates

Transaction completed.
```

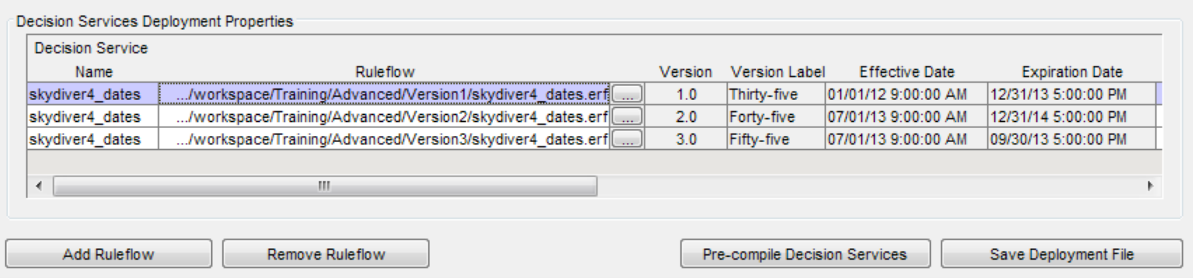
8. Notice that the Decision Service `skydiver4_noDates` shows up only **once** in the list, even though there are actually *three* different versions of this Decision Service deployed and loaded on Corticon Server. This is normal. Corticon Server "summarizes" all deployed versions of the Decision Service as a single entry on the list, even though each entry on the list may have multiple versions. So rather than thinking of these as three *different* Decision Services – think of them as three *versions* of the *same* Decision Service.

Modifying and deploying the sample Decision Services

To create and deploy the sample Ruleflows:

1. Open the Deployment Console, and then add the three Ruleflows we created, all under the same Decision Service name, as shown:

   Figure 42: Deployment Console with the Sample Ruleflows



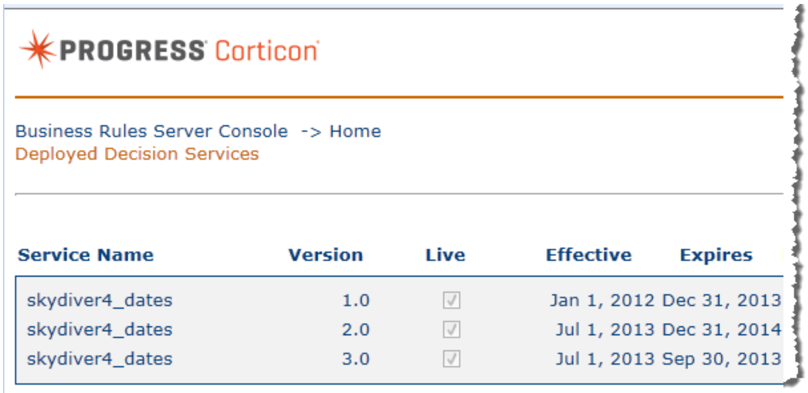
Decision Services Deployment Properties

Decision Service Name	Ruleflow	Version	Version Label	Effective Date	Expiration Date
skydiver4_dates	.../workspace/Training/Advanced/Version1/skydiver4_dates.erf	1.0	Thirty-five	01/01/12 9:00:00 AM	12/31/13 5:00:00 PM
skydiver4_dates	.../workspace/Training/Advanced/Version2/skydiver4_dates.erf	2.0	Forty-five	07/01/13 9:00:00 AM	12/31/14 5:00:00 PM
skydiver4_dates	.../workspace/Training/Advanced/Version3/skydiver4_dates.erf	3.0	Fifty-five	07/01/13 9:00:00 AM	09/30/13 5:00:00 PM

2. Click **Pre-compile Decision Services**, and save the files in a local output folder.
3. Copy the three `.eds` files you created to, in this example, the subdirectory of the default Progress Application Server included in the Corticon Server installation, `[CORTICON_WORK_DIR]\cdd`.
4. Start the Corticon Server.

5. In a browser, go the URL `http://localhost:8850/axis/`, and then log in with the user name and password `admin`.
6. Click **Deploy Decision Services**, and then browse to each of the `.eds` file you staged to add it to the deployment, as shown:

Figure 43: Server Console deploying the Sample Decision Services

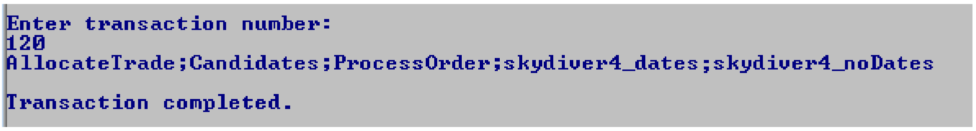


Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires
skydiver4_dates	1.0	☑	Jan 1, 2012	Dec 31, 2013
skydiver4_dates	2.0	☑	Jul 1, 2013	Dec 31, 2014
skydiver4_dates	3.0	☑	Jul 1, 2013	Sep 30, 2013

7. Confirm the deployment, by starting `testServerAxis.bat` (at `[CORTICON_HOME]\Server\bin\`, and entering `120 - Get Decision Service Names`. The results are as shown:

Figure 44: Confirming Deployment of Decision Service skydiver4_dates



```
Enter transaction number:
120
AllocateTrade;Candidates;ProcessOrder;skydiver4_dates;skydiver4_noDates

Transaction completed.
```

8. Notice that the Decision Service `skydiver4_dates` shows up only **once** in the list, even though there are actually *three* different versions of this Decision Service deployed and loaded on Corticon Server. This is normal. Corticon Server "summarizes" all deployed versions of the Decision Service as a single entry on the list, even though each entry on the list may have multiple versions. So rather than thinking of these as three *different* Decision Services – think of them as three *versions* of the *same* Decision Service.

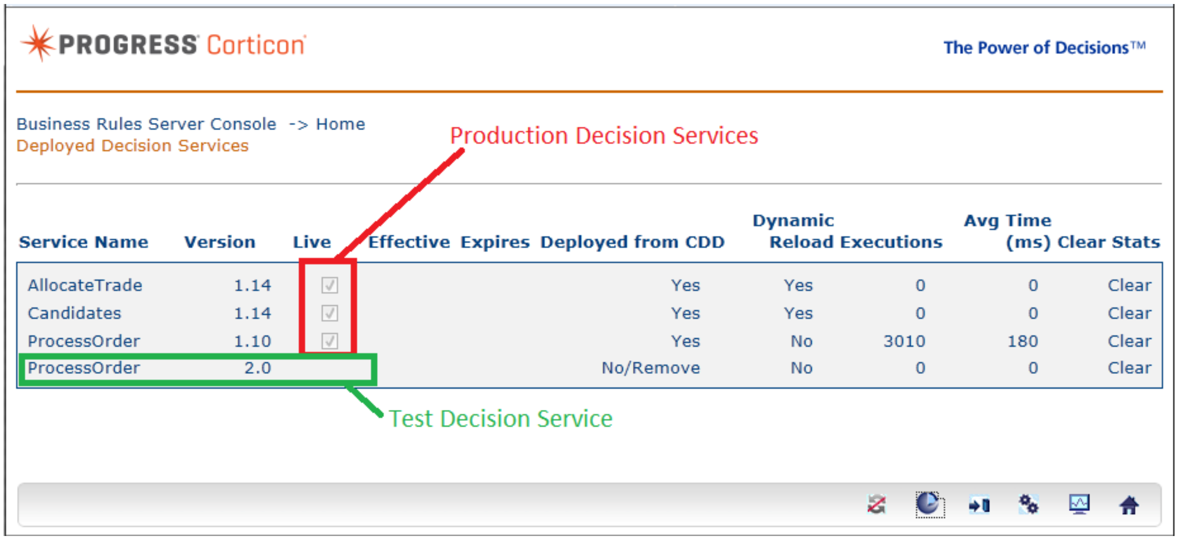
How Test Decision Services differ from Production (live) Decision Services

Corticon Server: Executing against Decision Services based on Production vs. Test and the use of Versioning and Effective Dating

Difference between Production and Live Decision Services:

You can use the Corticon Server Console to modify Decision Services that have been deployed on the Corticon Server. Production Decision Services are differentiated from Test Decision Services - that's because only Test Decision Services can be modified. A Decision Service marked as Production has its “Edit” capabilities disabled.

For example:



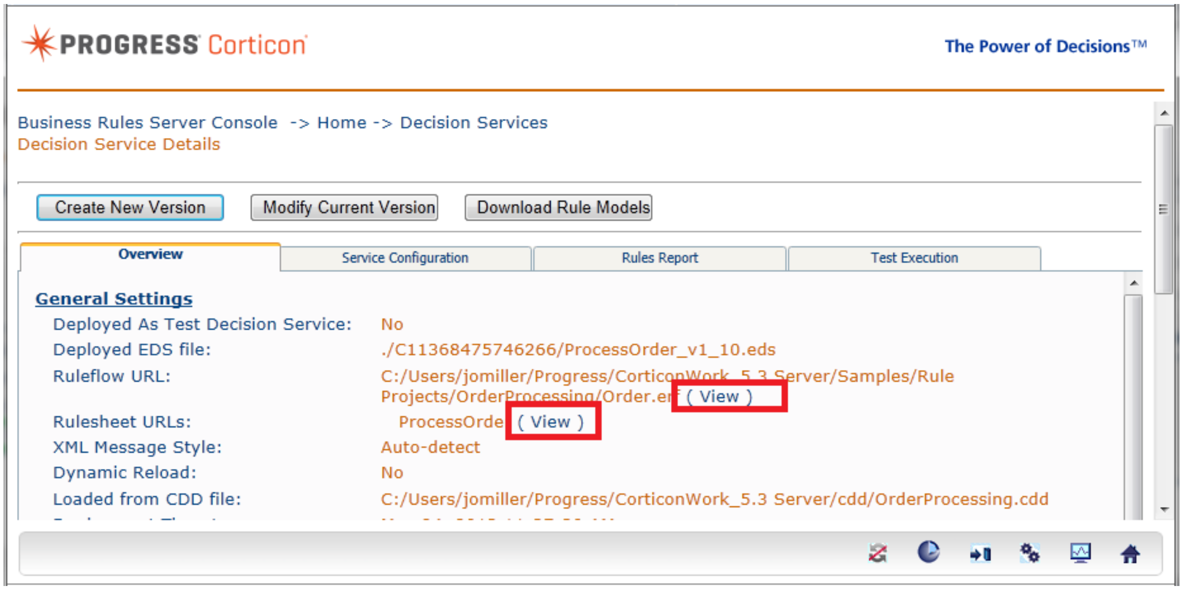
Business Rules Server Console -> Home
Deployed Decision Services

Production Decision Services

Service Name	Version	Live	Effective	Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats
AllocateTrade	1.14	☑			Yes	Yes	0	0	Clear
Candidates	1.14	☑			Yes	Yes	0	0	Clear
ProcessOrder	1.10	☑			Yes	No	3010	180	Clear
ProcessOrder	2.0				No/Remove	No	0	0	Clear

Test Decision Service

When we look at the Decision Service Detail Screen for ProcessOrder 1.10, we see that you can only “View” the Rulesheets:



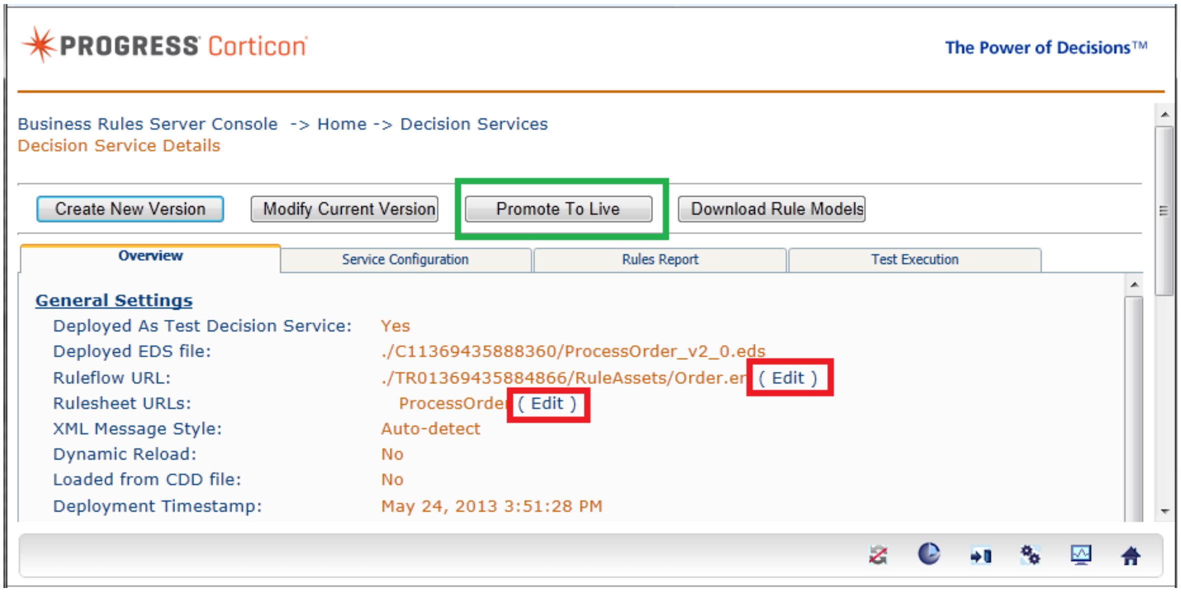
Business Rules Server Console -> Home -> Decision Services
Decision Service Details

General Settings

Deployed As Test Decision Service: No
Deployed EDS file: ./C11368475746266/ProcessOrder_v1_10.eds
Ruleflow URL: C:/Users/jomiller/Progress/CorticonWork_5.3 Server/Samples/Rule Projects/OrderProcessing/Order.erf (View)
Rulesheet URLs: ProcessOrder (View)
XML Message Style: Auto-detect
Dynamic Reload: No
Loaded from CDD file: C:/Users/jomiller/Progress/CorticonWork_5.3 Server/cdd/OrderProcessing.cdd

When we look at the Decision Service Detail Screen for ProcessOrder 2.0, we see that you can only “Edit” the Rulesheets.

Also, after you complete your changes, you can commit the changes to Live (Production) by clicking **Promote to Live**.



Business Rules Server Console -> Home -> Decision Services
Decision Service Details

General Settings

Deployed As Test Decision Service: Yes
Deployed EDS file: ./C11369435888360/ProcessOrder_v2_0.eds
Ruleflow URL: ./TR01369435884866/RuleAssets/Order.er
Rulesheet URLs: ProcessOrde
XML Message Style: Auto-detect
Dynamic Reload: No
Loaded from CDD file: No
Deployment Timestamp: May 24, 2013 3:51:28 PM

Determining which Decision Service to execute against based on Versions and Effective/Expires Dates

When a user supplies non-specific information about which Decision Service they want to execute against, the Corticon Server applies an algorithm to determine what is most suitable. When this algorithm is applied, all Test Decision Services are filtered out. In a production setting, the user cannot execute against a Test Decision Service unless the CorticonRequest is very specific about the Decision Service Name, Major Version Number, and Minor Version Number. The following examples use a mixture of ProcessOrder deployments to evalaute requests, first for only versions (or none), and then for effective/expires dates that might have version specified as well.

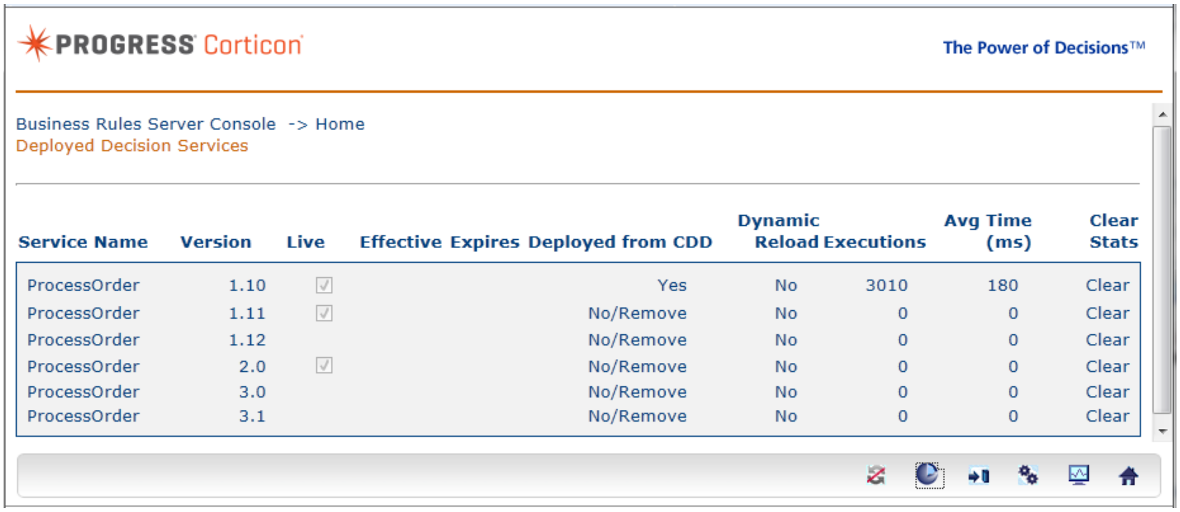
The algorithm evaluates versioning as follows:

1. If the user specifies the Decision Service Name (ProcessOrder), but does not specify the Major or Major/Minor Version Number, the Corticon Server tries to determine which version of ProcessOrder it should use.
2. If Decision Service Name only: Find the highest Production Major/Minor version for that Decision Service Name.
3. If Decision Service Name and Major Version Number: Find the highest Production Minor version for that Decision Service Name and Major Version Number.
4. If Decision Service Name and Major and Minor Version: Find that specific combination…regardless of whether it is Production or Test. If the user actually specifies the Major and Minor Version Number, assume that they know what they are doing and allow them to go against a Test Decision Service.
5. If no Decision Service is found based on this algorithm, then the Corticon Server throws a `CcServerDecisionServiceNotRegisteredException`.

The following two examples show how this algorithm applies in a variety of requests.

Example 1: Requests use versions but not effective dates

The deployed Decision Services are as shown:



PROGRESS Corticon

The Power of Decisions™

Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats
ProcessOrder	1.10	☑			Yes	No	3010	180	Clear
ProcessOrder	1.11	☑			No/Remove	No	0	0	Clear
ProcessOrder	1.12				No/Remove	No	0	0	Clear
ProcessOrder	2.0	☑			No/Remove	No	0	0	Clear
ProcessOrder	3.0				No/Remove	No	0	0	Clear
ProcessOrder	3.1				No/Remove	No	0	0	Clear

Test 1: User only specifies the Decision Service Name in the payload

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder">
        <WorkDocuments>
```

The Algorithm will find the largest Live Major Version and then the largest Minor Version for that Major Version.

In this case, this would be version 2.0.

Test 2a: User specifies the Decision Service Name and the Major Version Number in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder" decisionServiceTargetVersion="1">
        <WorkDocuments>
```

The Algorithm will find the largest Live Minor Version for that Major Version Number.

In this case, this would be version 1.11. Version 1.12 is ignored because it is not Live.

Test 2b: User specifies the Decision Service Name and the Major Version Number in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
decisionServiceName="ProcessOrder"
        decisionServiceTargetVersion="3">
        <WorkDocuments>
```

The Algorithm will find the largest Live Minor Version for that Major Version Number. However,

in this case, there is no Live Decision Service with Major Version 3. The Corticon Server will throw a `CcServerDecisionServiceNotRegisteredException`.

Test 3a: User specifies the Decision Service Name and the Major and Minor Version Number in the payload:

```
<CorticonRequest
        xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       decisionServiceName="ProcessOrder" decisionServiceTargetVersion="3.1">

        <WorkDocuments>
```

The Algorithm will try and find that specific version regardless if it is Production or Test.

In this case, there is a match and 3.1 would be used.

Test 3b: User specifies the Decision Service Name and the Major and Minor Version Number in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       decisionServiceName="ProcessOrder" decisionServiceTargetVersion="3.2">

        <WorkDocuments>
```

The Algorithm will try and find that specific version regardless if it is Production or Test.

In this case, there is Decision Service with Major and Minor Version 3.2. The Corticon Server will throw a `CcServerDecisionServiceNotRegisteredException`.

Example 2: Requests use effective dates and sometimes versions as well

The Test Cases above did not have Effective Dates in place. When Effective Dates are involved, the Algorithm to determine which Decision Service to use gets a more complex. This example describes how Effective and Expires Dates play a role in the Algorithm in determining the appropriate Decision Service to use during execution.

- When a Decision Service has no Effective or Expires Date specified, this means this Decision Service is valid from the start of time till the end of time…there is no lower or upper bounds on when this Decision Service is valid. (infinity in both directions)
- When a Decision Service has no Effective but does have an Expires Date specified, that means this Decision Service is valid from the start of time up to the specified Expires Date.
- When a Decision Service has an Effective Date but does not have an Expires Date specified, that means this Decision Service is valid from that specified Effective Date till the end of time.

In Example 1, all the Decision Services were unbounded (no Effective or Expires Date specified). In this case, none of the Decision Services were filtered out by the Algorithm based on the implied EffectiveDate value inside the CorticonRequest.

The algorithm is expanded as follows:

1. If the user specifies the Decision Service Name (ProcessOrder), but does not specify the Major or Major/Minor Version Number, or CorticonRequest TargetDate, the Corticon Server tries to determine which version of ProcessOrder it should use.
2. If Decision Service Name only: Find the highest Production Major/Minor version for that Decision Service Name that satisfies `Decision Service Effective Date < today() < Decision Service Expires Date`.
3. If Decision Service Name and CorticonRequest Target Date: Find the highest Production Major/Minor version for that Decision Service Name that satisfies `Decision Service`

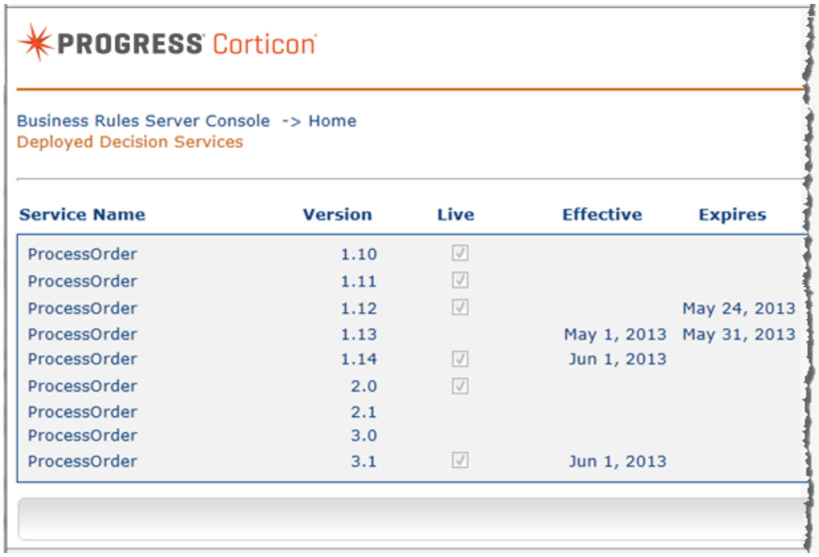
`Effective Date < CorticonRequest Target Date < Decision Service Expires Date`.

4. If Decision Service Name and Major Version Number: Find the highest Production Minor version for that Decision Service Name and Major Version Number that satisfies `Decision Service Effective Date < today() < Decision Service Expires Date`.
5. If Decision Service Name, Major Version Number, and CorticonRequest Target Date: Find the highest Production Minor version for that Decision Service Name and Major Version Number that satisfies `Decision Service Effective Date < CorticonRequest Target Date < Decision Service Expires Date .`

To simplify potential issues in determine what has priority between Major/Minor Version and Effective/Expiration Dates, an exception is thrown to the user if they specify Decision Service Name, Major Version, Minor Version, and Target Date.

Example 2

The deployed Decision Services are as shown:



Business Rules Server Console -> Home
Deployed Decision Services

Service Name	Version	Live	Effective	Expires
ProcessOrder	1.10	☑		
ProcessOrder	1.11	☑		
ProcessOrder	1.12	☑		May 24, 2013
ProcessOrder	1.13		May 1, 2013	May 31, 2013
ProcessOrder	1.14	☑	Jun 1, 2013	
ProcessOrder	2.0	☑		
ProcessOrder	2.1			
ProcessOrder	3.0			
ProcessOrder	3.1	☑	Jun 1, 2013	

Test 1a: User only specifies the Decision Service Name in the payload

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder">
        <WorkDocuments>
```

Since no Target Date was specified inside the CorticonRequest, `Target Date = today() = 8/1/2013`.

The Algorithm will first analyze all the Decision Services with name “ProcessOrder” and filter out those that don’t satisfy `Decision Service Effective Date < Target Date < Decision Service Expires Date`.

Decision Services that qualify include:

- Version 1.10
- Version 1.11
- Version 1.14
- Version 2.0

- Version 3.1

Test Decision Services are automatically excluded. These include Version 1.13, 2.1, and 3.0.

Now, find the largest Live Major Version and then the largest Minor Version for that Major Version from those versions that qualified.

In this case, this would be version 3.1.

Test 1b: User specifies the Decision Service Name in the payload and Effective Timestamp:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         decisionServiceName="ProcessOrder"
decisionServiceEffectiveTimestamp=”5/27/2013”>
         <WorkDocuments>
```

The Algorithm will first analyze all the Decision Services with name “ProcessOrder” and filter out those that don’t satisfy `Decision Service Effective Date < 5/27/2013 < Decision Service Expires Date`.

Decision Services that qualify include:

- Version 1.10
- Version 1.11
- Version 2.0

Version 1.12 expired 3 days prior.

Test Decision Services are automatically excluded. These include Version 1.13, 2.1, and 3.0.

Version 1.14 and 3.1 are not effective for another 4 days.

Now, find the largest Live Major Version and then the largest Minor Version for that Major Version from those versions that qualified.

In this case, this would be version 2.0.

Test 2a: User specifies the Decision Service Name and the Major Version Number in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         decisionServiceName="ProcessOrder" decisionServiceTargetVersion="1">
         <WorkDocuments>
```

Since no Target Date was specified inside the CorticonRequest, `Target Date = today() = 8/1/2013.`

The Algorithm will first analyze all the Decision Services with name “ProcessOrder” and filter out those that don’t satisfy `Decision Service Effective Date < Target Date < Decision Service Expires Date`.

Decision Services that qualify include:

- Version 1.10
- Version 1.11
- Version 1.14

All Decision Services that don’t have a Major Version Number = 1 are excluded from available list.

Test Decision Services are automatically excluded. This includes Version 1.13.

Now, find the largest Live Major Version and then the largest Minor Version for that Major Version from those versions that qualified.

In this case, this would be version 1.4.

Test 2b: User specifies the Decision Service Name, Major Version Number, and Effective Timestamp in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder" decisionServiceTargetVersion="1"
          decisionServiceEffectiveTimestamp=”5/27/2013”>
          <WorkDocuments>
```

The Algorithm will first analyze all the Decision Services with name “ProcessOrder” and filter out those that don’t satisfy `Decision Service Effective Date < 5/27/2013 < Decision Service Expires Date.`

Decision Services that qualify include:

- Version 1.10
- Version 1.11

All Decision Services that don’t have a Major Version Number = 1 are excluded from available list.

Test Decision Services are automatically excluded. This includes Version 1.13.

Now, find the largest Live Major Version and then the largest Minor Version for that Major Version from those versions that qualified.

In this case, this would be version 1.11.

Test 2c: User specifies the Decision Service Name, Major Version Number, and Effective Timestamp in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder"
decisionServiceTargetVersion="3”decisionServiceEffectiveTimestamp=”5/27/2013”>

        <WorkDocuments>
```

The Algorithm will first analyze all the Decision Services with name “ProcessOrder” and filter out those that don’t satisfy `Decision Service Effective Date < 5/27/2013 < Decision Service Expires Date`.

Decision Services that qualify include:

- none

All Decision Services that don’t have a Major Version Number = 3 are excluded from available list.

Test Decision Services are automatically excluded. This includes Version 3.0.

Version 3.1 isn’t effective for another 4 days.

The Corticon Server will throw a `CcServerDecisionServiceNotRegisteredException`.

Test 3a: User specifies the Decision Service Name, Major and Minor Version Numbers, and Effective Timestamp in the payload:

```
<CorticonRequest xmlns="urn:Corticon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        decisionServiceName="ProcessOrder" decisionServiceTargetVersion="3.1"

          decisionServiceEffectiveTimestamp=”5/27/2013”>
          <WorkDocuments>
```

The payload cannot have Major and Minor Version with an EffectiveTimestamp in the CorticonRequest. In this case, the Corticon Server will throw a `CcServerInvalidArgumentException`.

B

Access to Corticon knowledge resources

Complete online documentation for the current release

Corticon online tutorials:

- Tutorial: Basic Rule Modeling in Corticon Studio
- Tutorial: Advanced Rule Modeling in Corticon Studio
- Modeling Progress Corticon Rules to Access a Database using EDC
- Connecting a Progress Corticon Decision Service to a Database using EDC

Corticon guides (PDF):

- What's New in Corticon
- Corticon Installation Guide
- Corticon Studio: Rule Modeling Guide
- Corticon Studio: Quick Reference Guide
- Corticon Studio: Rule Language Guide
- Corticon Studio: Extensions Guide
- Corticon Server: Integration and Deployment Guide
- Corticon Server: Web Console Guide
- Corticon Server: Deploying Web Services with Java
- Corticon Server: Deploying Web Services with .NET

Corticon JavaDoc API reference (HTML):

- Corticon Foundation API
- Corticon Model API
- Corticon Server API

See also:

- Introducing the Progress® Application Server
- Corticon documentation for this release on the Progress download site: What's New Guide (PDF), Installation Guide (PDF), PDF download package, and the online Eclipse help installed with Corticon Studio.