

# Corticon Server: Deploying Web Services with Java



---

# Notices

---

## Copyright agreement

© 2014 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

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

Business Making Progress, Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Fathom, Making Software Work Together, OpenEdge, Powered by Progress, Progress, Progress Control Tower, Progress OpenEdge, Progress RPM, Progress Software Business Making Progress, Progress Software Developers Network, Rollbase, RulesCloud, RulesWorld, SequeLink, SpeedScript, Stylus Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. AccelEvent, AppsAlive, AppServer, BusinessEdge, Progress EasyI, DataDirect Spy, DataDirect SupportLink, EasyI, Future Proof, High Performance Integration, Modulus, OpenAccess, Pacific, ProDataSet, Progress Arcade, Progress Pacific, Progress Profiles, Progress Results, Progress RFID, Progress Responsive Process Management, Progress Software, ProVision, PSE Pro, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

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

<b>Preface.....</b>	<b>9</b>
Progress Corticon documentation.....	9
Overview of Progress Corticon.....	11
 <b>Chapter 1: Conceptual overview.....</b>	 <b>13</b>
What is a web service?.....	13
What is a Decision Service?.....	14
What is the Corticon Server with Java?.....	14
What is a web services consumer?.....	14
 <b>Chapter 2: Getting started.....</b>	 <b>15</b>
 <b>Chapter 3: Using the Corticon Server for Java installer.....</b>	 <b>17</b>
Downloading the Corticon Server for Java package.....	17
System requirements.....	18
Corticon Server for Java setup wizard.....	18
Registering your Corticon license for Java Server.....	24
Limits of the default evaluation license.....	25
 <b>Chapter 4: Starting Corticon Server for Java.....</b>	 <b>27</b>
 <b>Chapter 5: Enabling HTTPS.....</b>	 <b>31</b>
 <b>Chapter 6: Corticon Java Server files and API tools.....</b>	 <b>33</b>
Basic server classes.....	34
Setting up Corticon Server use cases.....	34
Installing Corticon Server as a J2EE SOAP servlet.....	35
Installing Corticon Server as a J2EE enterprise Java bean (EJB).....	35
Installing Corticon Server as Java classes in-process or in a custom Java container.....	36
The Corticon home and work directories.....	37
Configurations that use global environment settings.....	37
The Corticon Server Sandbox.....	38
Testing the installed Corticon Server.....	38
Testing the installed Corticon Server as a J2EE SOAP servlet.....	38
Testing the installed Corticon Server as in-process Java classes.....	42

---

## **Chapter 7: Deploying a Ruleflow to the Corticon Server.....45**

Creating a Ruleflow.....	46
Creating and installing a Deployment Descriptor file.....	46
Using the Deployment Console tool's Decision Services on the Java Server.....	46
Installing the Deployment Descriptor file.....	49
Hot re-deploying Deployment Descriptor files and Ruleflows.....	50

## **Chapter 8: Consuming a decision service.....51**

Integrating and testing a Decision Service.....	52
Path 1- Using Corticon Studio as a SOAP client to consume a Decision Service.....	53
Creating a new Java server test in Corticon Studio.....	53
Executing the remote test.....	55
Path 2 - Using bundled sample code to consume a Decision Service.....	55
Sending a request message to Corticon Server.....	57
Path 3 - Using SOAP client to consume a Decision Service.....	59
Web services SOAP messaging styles.....	59
Creating a service contract using the Deployment Console.....	60
Creating a SOAP request message for a Decision Service.....	61
Sending a SOAP request message to Corticon Server.....	61
Path 4 - Using JSON/RESTful client to consume a Decision Service on Java Server.....	62
Running the sample JSON Request.....	62
Troubleshooting.....	65

## **Chapter 9: Modeling and managing rules using Server Console.....67**

Lifecycle management Through the Server Console.....	68
Creating a new decision service version.....	68
Opening the new version.....	69
Modifying the new version.....	69
Promoting the new version to live.....	70
Removing the version from live.....	70
Telling the server where to find Deployment Descriptor files .....	70

## **Chapter 10: Using the Corticon Server Console.....71**

Launching and logging in to Corticon Server Console.....	71
Using LDAP authentication in Server Console.....	74
Console main menu page.....	77
Decision Services page.....	78
Decision Service actions and information.....	80
Execution and Distribution Visualizations.....	86
Deploy Decision Service page.....	88
Server state.....	88

---

Configure Rules Server.....	90
Logging tab.....	90
Deployment Directory tab.....	90
Decision Service Options tab.....	91
License tab.....	92
Monitor Rules Server page.....	92
Rules Server Stats tab.....	92
Rules Server Settings tab.....	93
Environment Settings tab.....	94
Memory Utilization tab.....	94
Rules Server Log.....	95
WSDLs.....	95

## **Chapter 11: Summary of Java samples.....97**

## **Appendix A: Pacific Application Server Administration.....99**

Administrative scripts.....	99
Administrative utilities.....	102
TCMAN.....	102
JMX and JConsole.....	102
Startup and shutdown sequences when using TCMAN.....	103
Working with Instances.....	107
Creating instances with TCMAN.....	109
Instance management with TCMAN.....	110
Installing and running an instance as a Windows service.....	111
Installing and running an instance as a UNIX daemon.....	112
Tomcat logging.....	112
Updating the core Tomcat server in Pacific Application Server.....	114
TCMAN Reference.....	114
The tcman command.....	114
Manager actions.....	116
List deployed applications (list).....	117
Display OS and server information (info).....	118
Deploy a Web application (deploy).....	118
Undeploy a Web application (undeploy).....	119
Reload a Web application (reload).....	120
Display detailed server status (status).....	121
Display memory leaks (leaks).....	122
Start a Web application (enable).....	123
Stop a Web application (disable).....	124
Display global server resources (resources).....	125
Display Web application HTTP sessions (sessions).....	125
Server actions.....	126

---

Create an instance (create).....	126
Delete an instance (delete).....	128
Display and manage an instance's configuration (config).....	129
Display or modify the server features of an instance (feature).....	131
Clean up or archive a server's log files (clean).....	132
Display a server's instances (instances).....	133
Register an instance for tracking (register).....	134
Stop tracking an instance (unregister).....	135
Start an instance (start).....	136
Stop an instance (stop).....	137
Display server, OS, and runtime version information (version).....	138
Test a server configuration (test).....	138
General actions.....	139
Display help (help).....	139
Display runtime environment information (env).....	140



---

# Preface

---

For details, see the following topics:

- [Progress Corticon documentation](#)
- [Overview of Progress Corticon](#)

## Progress Corticon documentation

The following documentation, as well as a *What's New in Corticon* document, is included with this Progress Corticon release:

Corticon Tutorials	
<i>Corticon Studio Tutorial: Basic Rule Modeling</i>	Introduces modeling, analyzing, and testing rules and decisions in Corticon Studio. Recommended for evaluators and users getting started. <i>See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's <b>Help</b> menu.</i>
<i>Corticon Studio Tutorial: Advanced Rule Modeling</i>	Provides a deeper look into Corticon Studio's capabilities by defining and testing vocabularies, scope, collections, messages, filters, conditions, transient data, and calculations in multiple rulesheets that are assembled into a Ruleflow. <i>See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's <b>Help</b> menu.</i>

<i>Corticon Tutorial: Using Enterprise Data Connector (EDC)</i>	Introduces Corticon's direct database access with a detailed walkthrough from development in Studio to deployment on Server. Uses Microsoft SQL Server to demonstrate database read-only and read-update functions.
<b>Corticon Studio Documentation: Defining and Modeling Business Rules</b>	
<i>Corticon Studio: Installation Guide</i>	Step-by-step procedures for installing Corticon Studio on computers running Microsoft Windows as a standalone installation and as a part of an existing Eclipse installation such as Progress Developer Studio for OpenEdge. Shows how to enable internationalization on Windows.
<i>Corticon Studio: Rule Modeling Guide</i>	Presents the concepts and purposes the Corticon Vocabulary, then shows how to work with it in Rulesheets by using scope, filters, conditions, collections, and calculations. Discusses chaining, looping, dependencies, filters and preconditions in rules. Presents the Enterprise Data Connector from a rules viewpoint, and then shows how database queries work. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting and many <i>Test Yourself</i> exercises.
<i>Corticon Studio: Quick Reference Guide</i>	Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.
<i>Corticon Studio: Rule Language Guide</i>	Reference information for all operators available in the Corticon Studio Vocabulary. A Rulesheet example is provided for many of the operators. Includes special syntax issues, handling arithmetic and character precedence issues.
<i>Corticon Studio: Extensions Guide</i>	Detailed technical information about the Corticon extension framework for extended operators and service call-outs. Describes several types of operator extensions, and how to create a custom extension plug-in.
<b>Corticon Server Documentation: Deploying Rules as Decision Services</b>	
<i>Corticon Server: Integration &amp; Deployment Guide</i>	An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the Deployment Console tool. Describes JSON request syntax and REST calls. Discusses relational database concepts and implementation of the Enterprise Data Connector. Goes deep into the server to discuss state, persistence, and invocations by version or effective date. Includes server monitoring techniques, performance diagnostics, and recommendations for performance tuning.

<i>Corticon Server: Deploying Web Services with Java</i>	Details installing the Corticon Server as a Web Services Server, and then deploying and exposing Decision Services as Web Services on the Pacific Application Server (PAS) and other Java-based servers. Includes samples of XML and JSON requests. Presents the features and functions of the browser-based Server Console. Provides administrative instructions for the Pacific Application Server.
<i>Corticon Server: Deploying Web Services with .NET</i>	Details installing the Corticon Server as a Web Services Server, and then deploying and exposing decisions as Web Services with .NET. Includes samples of XML and JSON requests. Provides installation and configuration information for the .NET Framework and Internet Information Services (IIS) on various supported Windows platforms.

## Overview of Progress Corticon

Progress® Corticon® is the Business Rules Management System with the patented "no-coding" rules engine that automates sophisticated decision processes.

### Progress Corticon products

Progress Corticon distinguishes its development toolsets from its server deployment environments.

- **Corticon Studio** is the Windows-based development environment for creating and testing business rules:
  - When installed as a standalone application, Corticon Studio provides the complete Eclipse development environment for Corticon as the **Corticon Designer** perspective. You can use this fresh Eclipse installation as the basis for adding other Eclipse toolsets.
  - When installed into an existing Eclipse such as the **Progress Developer Studio (PDS)**, our industry-standard Eclipse and Java development environment, the PDS enables development of Corticon applications in the **Corticon Designer** perspective that integrate with other products, such as Progress OpenEdge.

---

**Note:** Corticon installers are available for 64-bit and 32-bit platforms. Typically, you use the 64-bit installer on a 64-bit machine, where that installer is not valid on a 32-bit machine. When adding Corticon to an existing Eclipse, the target Eclipse must be an installation of the same bit width. Refer to the *Corticon Studio: Installation Guide* to access, prepare, and install Corticon Studio.

---

**Studio Licensing** - Corticon embeds a time-delimited evaluation license that enables development of both rule modeling and Enterprise Data Connector (EDC) projects, as well as testing of the projects in an embedded Axis test server. You must obtain Studio development licenses from your Progress representative.

- **Corticon Servers** implement web services for business rules defined in Corticon Studios:
  - **Corticon Server for deploying web services with Java** is supported on various application servers, 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 as Corticon Decision Services. The guide *Corticon Server: Deploying web services with Java* provides details on the full set of platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.

- **Corticon Server for deploying web services with .NET** facilitates deployment of Corticon Decision Services on Windows .NET Framework and Microsoft Internet Information Services (IIS). The guide *Corticon Server: Deploying web services with .NET* provides details on the platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.

**Server Licensing** - Corticon embeds a time-delimited evaluation license that enables evaluation and testing of rule modeling projects on supported platform configurations. You must obtain server deployment licenses and server licenses that enable the Enterprise Data Connector (EDC) from your Progress representative.

---

## Conceptual overview

---

This guide is a walkthrough of fundamental concepts and functions of Corticon Server for Java. The examples focus on the default Corticon Server, the Pacific Application Server from Progress Software. The Pacific Application Server (PAS) 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 with 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 with Java?

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.

## 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.

## 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*.

This Tutorial consists of four main sections:

### **Using the Corticon Server for Java installer**

Describes downloading and installing Corticon Server for Java on your designated server machine.

### **Installing the Corticon Server for Java as a web services server**

Describes deploying the Corticon Server to the Pacific Application Server, which is installed by default during the installation process described in the first section.

### **Deploying a Ruleflow to the Corticon Server**

Describes deploying Ruleflows to the Server and exposing them as web services, which we call *Decision Services*. Once a Ruleflow becomes a Decision Service, it may be consumed by any external application or process capable of interacting with standard document-style web services.

### **Consuming a Decision Service**

Describes integrating and testing your deployed Decision Service by creating and sending request messages to the Server, and viewing the response messages. Two methods of integration and testing are discussed: one method assumes you have access only to the tools contained in the default Server installation, while the other method assumes you have a commercially available SOAP client, application or tool to perform these tasks.





## Using the Corticon Server for Java installer

---

For details, see the following topics:

- [Downloading the Corticon Server for Java package](#)
- [System requirements](#)
- [Corticon Server for Java setup wizard](#)
- [Registering your Corticon license for Java Server](#)
- [Limits of the default evaluation license](#)

## Downloading the Corticon Server for Java package

Corticon Server for Java and its Service Packs are packaged in executable installer applications:

- The 5.4 installer, `PROGRESS_CORTICON_5.4_SERVER_WIN_64.exe`, to perform a new installation.
- The 5.4.1 Service Pack installer, `PROGRESS_CORTICON_5.4.1_SERVER_WIN_64.exe`, to update a 5.4.0 installation.

To download the installers:

1. Get credentials to access and download packages on the Progress Software Electronic Software Download (ESD) site.
2. Connect to the ESD, and then navigate to the Corticon 5.4 pages.
3. Locate, download, and save the installers to a temporary location accessible by the target machine.

## System requirements

Corticon Server for Java requirements include:

- 8 GB System RAM (minimum of 2 GB available RAM)
- 500 MB disk space
- Java version 6 or 7

Progress Corticon Server for Java is supported on several Microsoft Windows platforms. Servers can be deployed onto a variety of UNIX/Linux platforms and Application Servers. See the Progress Software web page [Progress Corticon 5.4 - Supported Platforms Matrix](#) for more information.

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.

## Corticon Server for Java setup wizard

You can install Corticon Server 5.4.1 for Java on a machine that does not already have it installed. If you are upgrading from 5.3, you can install 5.4 as a separate installed software on the same machine. You must manage any port overloads that might result from running both versions.

---

**Note:** An installed version of Corticon Server for Java 5.2 or earlier on the target machine must be [uninstalled](#), then Corticon Server for Java 5.4 installed. Consult with Progress Corticon Support or your Progress representative to consider your migration strategies for existing assets before you take action.

---

### Installation type

There are three ways to create Corticon Server for Java installations:

- **New 5.4.1 installation** - Perform a [new installation](#), then update it with the [5.4.1 Service Pack](#).
- **Update a 5.4.0 installation** - Apply the [5.4.1 Service Pack](#).
- **Upgrade a 5.2 or earlier installation to the latest 5.4 Service Pack** - An installed version of Corticon Server for Java 5.2 or earlier installed on the target machine must be [uninstalled](#), then [Corticon 5.4.0 Server installed](#), followed by applying the [5.4.1 Service Pack](#).

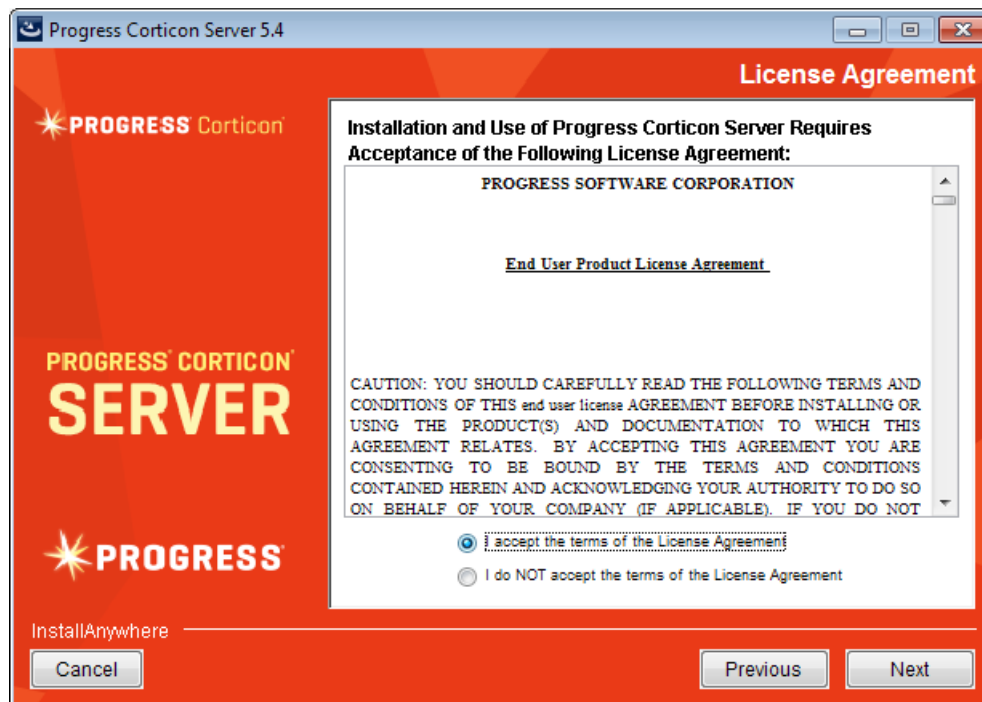
Consult with Progress Corticon Support or your Progress representative to consider your migration strategies for existing assets before you take action.

## Installation procedure

To create an installation of Corticon Server 5.4 for Java:

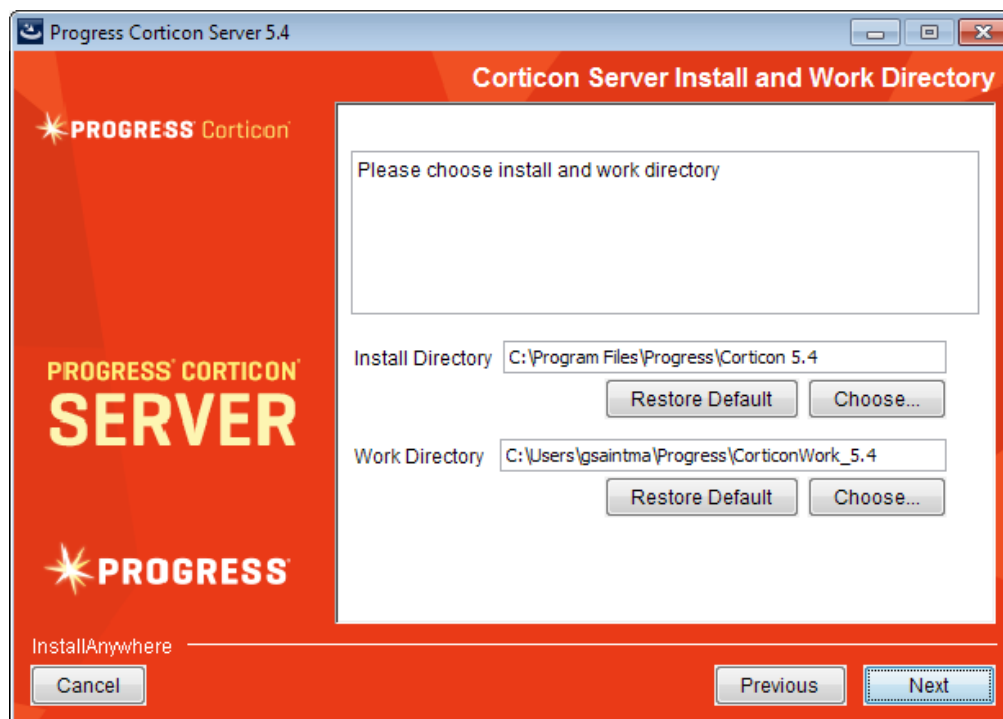
1. Double click on `PROGRESS_CORTICON_5.4_SERVER_WIN_64.exe` to open the Progress Corticon Server Setup Wizard.
2. Click **Next** to continue.

The **License Agreement** panel opens.



3. After you have read, understood, and agreed to the terms of End User License Agreement, choose **I accept the terms of the license agreement**, and then click **Next**.

The **Choose Install Folder** panel opens.

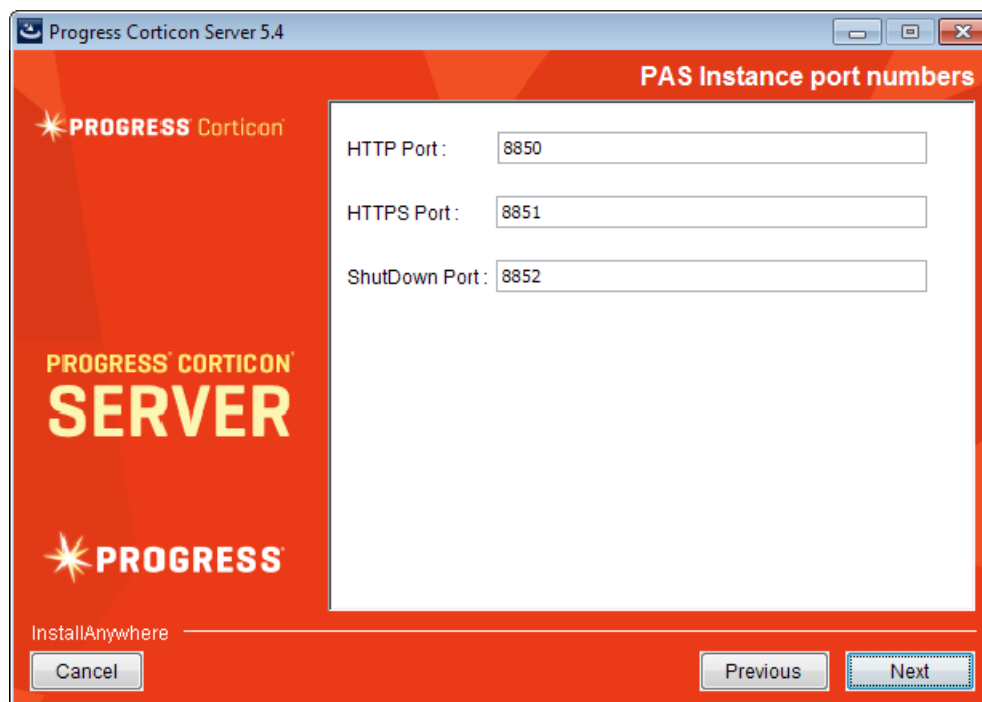


4. The default installation directories are shown.

To specify preferred directories on each line, either enter the explicit path, or click **Choose** to browse to each preferred directory.

Click **Next**.

The **PAS Instance port numbers** panel opens.



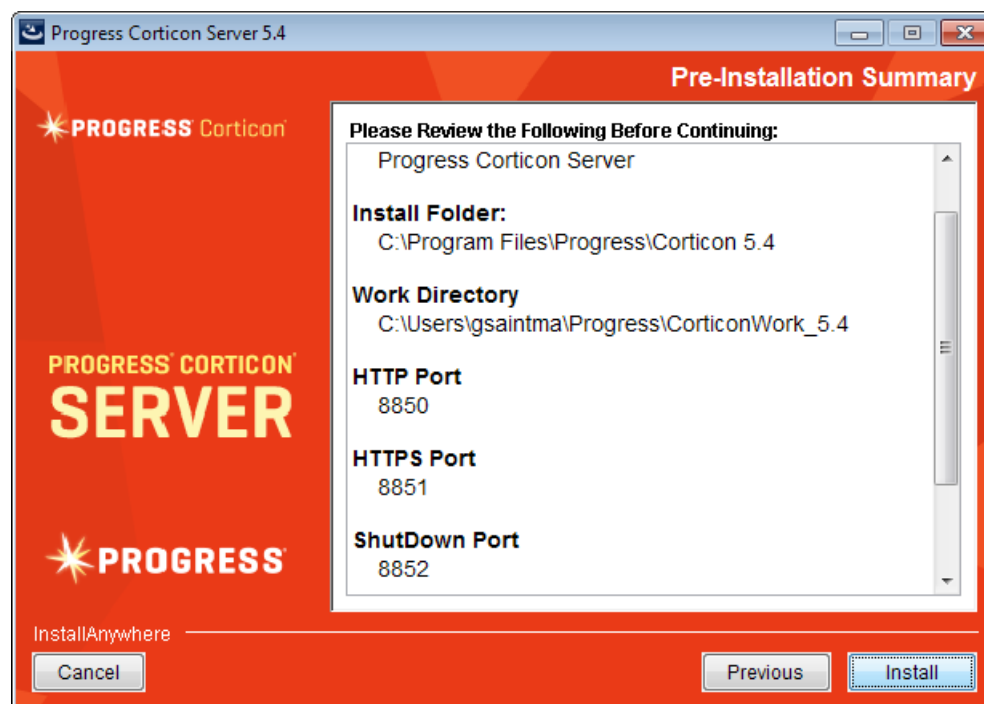
- 5.

**Note:** If any of the preferred ports conflict with existing applications, you can change them at this time to an integer value from 1024 to 49151. Be aware that the documentation references these ports as though the default values, as shown, were accepted. Consult with your system administrator to identify alternate ports you might use, or to reset the ports on the conflicting applications.

**Note:** If you change the HTTP port value, you must also change the deployment property `com.corticon.deployment.soapbindingurl_1=http://localhost:8850/axis` to have your preferred HTTP port value. To apply this override, add your line to the `brms.properties` file located at the server installation's `[CORTICON_WORK_DIR]` root.

Enter your preferred port numbers, and then click **Next**.

6. The **Pre-Installation Summary** panel opens.

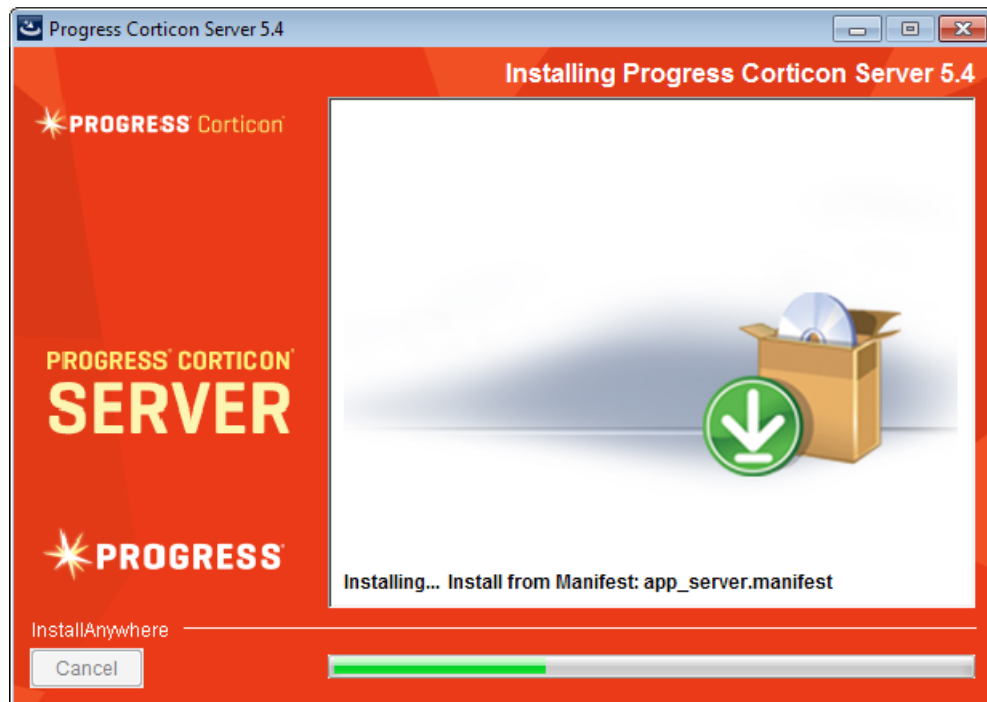


Verify your selections in the **Pre-Installation Summary** panel.

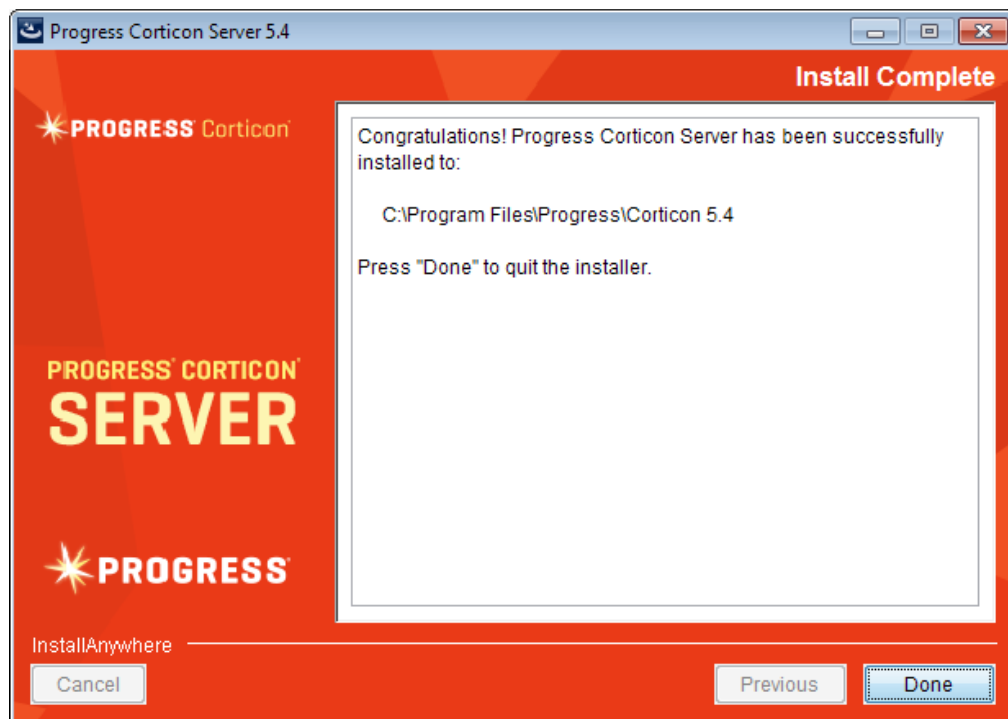
Confirm that your installation location has adequate disk space for the Corticon Studio components as well as 100 MB of workspace.

7. Click **Install** to continue.

The installation status window opens.



This panel displays a status bar during the installation process. When done, the **Install Complete** panel opens.

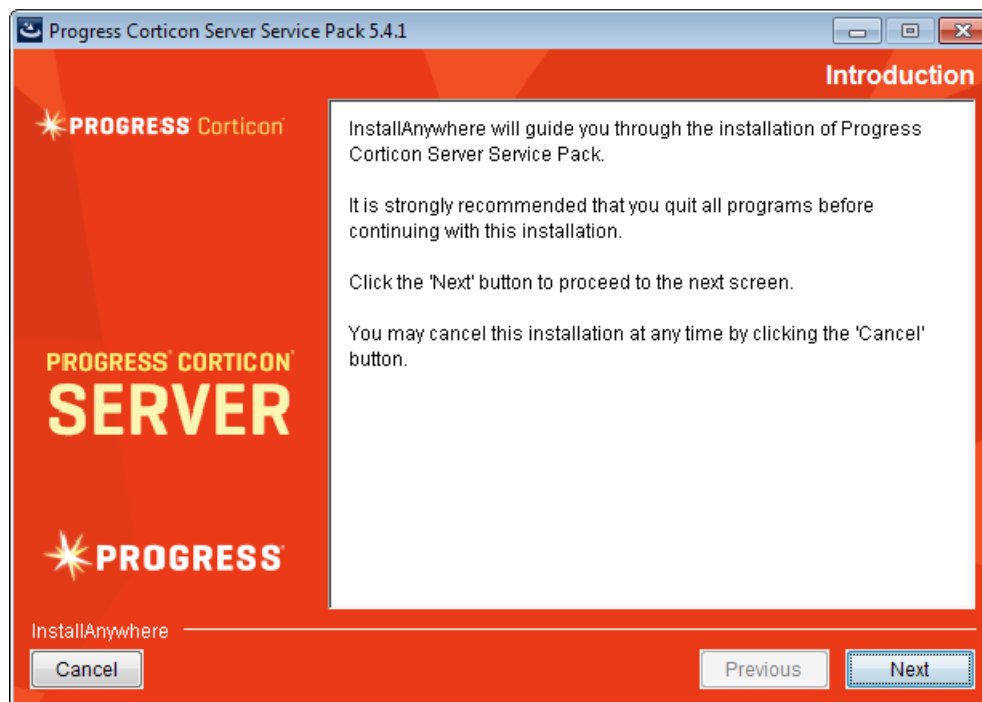


8. Choose **Done** to complete the Corticon Server for Java installation and close the installer.

### Service Pack updater

To update a 5.4.0 installation of Corticon Server for Java to a Service Pack:

1. If the installed Corticon Server 5.4.0 you want to update is running, stop it now.
2. Double click on `PROGRESS_CORTICON_5.4.1_SERVER_WIN_64.exe` to open the Progress Corticon Server Service Pack Setup Wizard:



3. Click **Next** through the panels, and then click **Install**.

When the 5.4 installation is detected, and it is determined that this Service Pack has not already been applied, the updater proceeds to complete the update of the installation.

## Uninstalling Corticon Server

To remove a version of the server to prepare for a new installation (or if you want to fully remove it), use this procedure.

---

**Note:** Uninstall removes a complete major.minor version. You cannot uninstall just a Service Pack.

---

### To uninstall Corticon Server for Java:

1. Stop the server.
2. Backup any files you want to retain.
3. Navigate to `[CORTICON_HOME]\Uninstall_Progress Corticon Server 5.4.`
4. Run `Uninstall Progress Corticon Server 5.4.exe`.

The installed files are removed. Note that files you created are NOT removed or replaced during this process.

If the Uninstaller program is unable to fully remove components (usually because they are open), it will display messages, and might require a reboot to complete the process.

## Registering your Corticon license for Java Server

Progress Corticon embeds an evaluation license in its products to help you get started.

- Corticon Studio evaluation licenses let you use database access (Enterprise Data Connector or "EDC"), and are timed to expire on a preset date.
- Corticon Server evaluation licenses do not enable use of Enterprise Data Connector, and limit the number of decision services, rules, and pools in use. They too are timed to expire on a preset date.

When you obtain a license file, it applies to Studios as well as Servers. You must perform configuration tasks to record it for each Corticon Studio, each Corticon Server, and each Deployment Console. If you intend to use EDC on your Corticon Servers, your Corticon license must allow it. Contact Progress Corticon technical support if you need to acquire a license.

The Corticon Server license is placed at two locations in the installation to enable use and -- when specified in the license -- enable EDC functions for:

- Corticon Server
- Corticon Server Console
- Corticon Deployment Console

Once you have run Corticon Server, you can update its license locations with your license.

### To configure Corticon Java Server to access your license file:

1. Stop Corticon Server.
2. Copy your license JAR by its default name, `CcLicense.jar`.
3. Navigate to the installation's `[CORTICON_HOME]\Server\lib` directory to paste the file and overwrite the existing file in that location.
4. Navigate to the installation's `[CORTICON_HOME]\Server\pas\corticon\webapps\axis\WEB-INF\lib` directory to paste the file and overwrite the existing file in that location.

When you restart Corticon Server, your license file will be referenced.

When you launch the Corticon Deployment Console, your license with its registration information is registered for the Corticon Deployment Console. If your license enables EDC, the **Database Access** fields and functions are enabled.



**Note:**

You can choose to locate the license by another JAR name at a preferred location, and then expressly identify it to the server.

To custom configure Corticon Java Server's license location:

1. Navigate in the file system to the installation's `[CORTICON_HOME]\Server\bin` subdirectory.
  2. Double-click on `testServerAxis.bat`, then do the following:
    - a. Type `416` and then press **Enter**.
    - b. Enter (or copy/paste) the complete path to the location of the license JAR file, as in this example, `C:\licenses\myCorticon540EDC_CcLicense.jar`. The command echoes back `Transaction completed`.
    - c. To confirm the setting, type `415` and then press **Enter**. The path is echoed back (you might need to scroll up to the command line.)
  3. For the in-process instance, double-click on `testServer.bat` to perform the same `416` and `415` tasks as in Step 2 above
- 

## Limits of the default evaluation license

The license included in the default Corticon Server installation has pre-set limits on certain Corticon Server and Decision Service parameters. These limits are:

- **Number of Decision Services** – Up to 20 Decision Services may be deployed at any given time. This means the sum total of all Decision Services loaded via `.cdd` files, Web Console, or APIs cannot exceed 20.
- **Pool Size** – No Decision Service may have a maximum pool size setting of greater than 1. Pool size is measured on a Decision Service-by-Decision Service basis, so you may have 20 Decision Services deployed (compliant with the Decision Service limitation above), each with 1 Reactor in its pool, without violating the default license restrictions.
- **Number of Rules** – All rules in all deployed Ruleflows (that is, all deployed Decision Services) must not exceed 500. A rule generally consists of a single Condition/Action Column or a single Action row in Column 0. Filter expressions do not count because they only modify other rules.

The Corticon Server log captures errors and exceptions caused by expired or "under-strength" licenses. Such log messages are detailed in the *Troubleshooting* section of the *Rule Modeling Guide*.

If you are Progress Corticon customer, you should have access to an unlimited license that will lift these restrictions. If you are an evaluator, and discover that these limitations are preventing or inhibiting your evaluation, contact your Progress Software representative to get a license with expanded capabilities.



---

## Starting Corticon Server for Java

---

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

```
[CORTICON_HOME]\Server\pas\corticon\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 Pacific Application Server, refer to the *Integration & Deployment Guide* for details on using the standard `axis.war` package in your preferred web server.

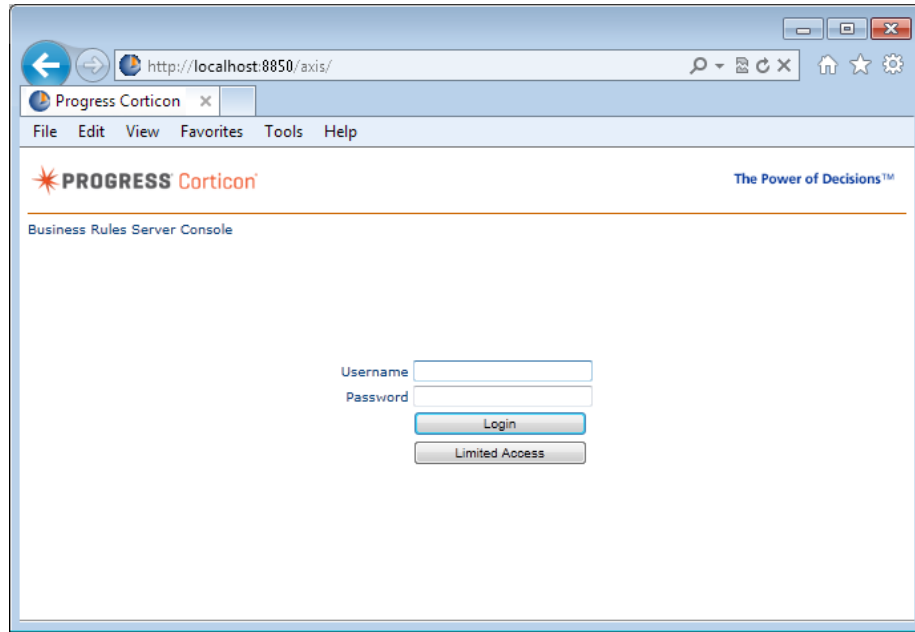
---

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

1. Start Corticon Server from the Windows **Start** menu by choosing **All Programs > Progress > Corticon 5.4 > Start Corticon Server**
2. In an Internet browser, enter the URL <http://localhost:8850/axis>.

The Business Rules Server Console opens to its login page, as shown:

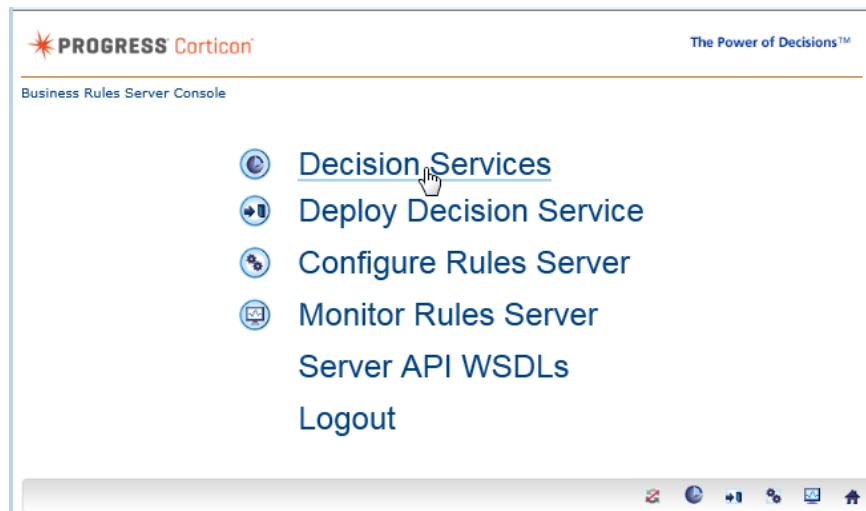
Figure 1: Server Console login page



3. Enter the Username `admin` and the Password `admin`. Click **Login**.

The Business Rules Server console lets you choose a functional path, as shown:

Figure 2: Server console functions



4. Click **Decision Services** to view the list of deployed Web services, which might look like this:

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	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	WSDL
Candidates	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	WSDL
ProcessOrder	1.10	<input checked="" type="checkbox"/>		Yes	No	0	0	Clear	WSDL





---

## 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\_HOME]\pas\corticon\webapps\axis\WEB-INF\web.xml 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.



---

## Corticon Java Server files and API tools

---

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

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

- 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 5.4 - 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.

For details, see the following topics:

- [Basic server classes](#)
- [Setting up Corticon Server use cases](#)
- [The Corticon home and work directories](#)
- [The Corticon Server Sandbox](#)
- [Testing the installed Corticon Server](#)

## Basic server classes

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

- `CcServer.jar` – The main Corticon Server JAR, containing the core engine logic.
- `CcConfig.jar` – Contains a set of text `.property` files that list and set all the configuration properties needed by Corticon Server. These properties pages are not intended for user access. Instead, the file `brms.properties`, installed by every product at the root of `[CORTICON_WORK_DIR]`, enables you to add your override settings to be applied after the default settings have been loaded.
- `CcLicense.jar` – An encrypted file containing the licensing information required to activate Corticon Server.
- `CcThirdPartyJars.jar` – Contains third-party software such as XML parsers and JDOM. Corticon warrants Corticon Server operation ONLY with the specific set of classes inside this jar.
- `CcI18nBundles.jar` – Contains the English and other language templates used by Corticon Server.
- `ant-launcher.jar` – Supports Corticon Server's deployment-time Decision Service compilation capability
- `CcExtensions.jar` – **Optional**. – Necessary only if you have added new rule language operators as "Extended Operators." (See the *Rule Language Guide* for details.)

## Setting up Corticon Server use cases

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

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

## 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 Pacific 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 Pacific Application Server. Your web server documentation will include instructions for installing or loading a `.war` file.

This `.war` file, named `axis.war`, is located in your Corticon working directory in the `[CORTICON_WORK_DIR]\Samples\Server\Containers\WAR` directory.

---

**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 Pacific Application Server (built on Apache Tomcat), and the Apache Axis SOAP messaging infrastructure. The Corticon Server Installer sets up and configures Pacific 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.

This `.ear` file, `CcServer.ear`, is located in the Corticon working directory `[CORTICON_HOME]\Server\Containers\EAR` directory.

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 [Using the Corticon Server for Java installer](#) on page 17 in this 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 *Rule Language Guide* for details.)]
2. Configure the Java classpath to include the JAR files listed above.
  3. Change the `logPath` property in `brms.properties` to an explicit path to a directory for the Corticon Server Log file.
  4. 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 37)
    - 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. As you could have elected to specify preferred locations for the installation directory and for the work directory, the installer sets `CORTICON_HOME` to the Server **Install Directory**, (referred to within the installer as `%CORTICON_INSTALL_DIR%`), and `CORTICON_WORK_DIR` to the Server **Work Directory** you specified. These variables are used throughout the product to determine the relative location of other files.

## Configurations that use global environment settings

### CORTICON\_HOME

The batch file `startServer.bat`, located in the `[CORTICON_HOME]\Server\bin` directory calls on the environment setting file `..\..\bin\corticon_env.bat` to set `CORTICON_HOME`.

The call to the startup script is `../pas/corticon/bin/startup.bat`. That launches the file `C:\Program Files\Progress\Corticon 5.4\Server\pas\corticon\bin\startup.bat`.

### CORTICON\_WORK\_DIR

The logs are set to be stored in the Corticon Server work directory by default:

```
logpath=%CORTICON_WORK_DIR%/logs
```

where `%CORTICON_WORK_DIR%` in a default installation is  
`C:\Users\{username}\Progress\Corticon 5.4.`

You can specify a preferred target subdirectory by setting the `logpath` property in your `brms.properties` file.

### 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.

## 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_5.4\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.

## 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 Pacific 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 Pacific 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.

When executed, `testServerAxis` script opens a Windows console. Once all classes are loaded, the Corticon Server starts up in the JRE required for this simple SOAP client class, as shown:

**Figure 3: Top Portion of the `testServerAxis.bat` Windows console**

```

C:\Windows\system32\cmd.exe

C:\Program Files\Progress\Corticon 5.4\Server\bin>echo off

Defining Corticon 5.4 runtime environment.

C:\Program Files\Progress\Corticon 5.4\Server\bin>CALL "%JRE%\bin\java.exe" -cp ..\lib\CcConfig.jar;..\lib\CcServer.jar;..\lib\CcLibBundles.jar;..\lib\CcThirdPartyJars.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\activation-1.2.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\axis.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\commons-discovery.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\commons-logging-1.1.3.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\jaxrpc.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\mail-1.2.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\saa.jar;..\pas\corticon\webapps\axis\WEB-INF\lib\soap-2.3.1.jar -DCORTICON_HOME="C:\Program Files\Progress\Corticon 5.4" -DCORTICON_WORK_DIR="C:\Users\gsaintma\Progress\CorticonWork_5.4" com.corticon.eclipse.server.core.CcServerApiTest re
note 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:\Users\gsaintma\Progress\CorticonWork_5.4\output

-----

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

```

The header information notes where it expects to submit requests. After the header information, the first set API menu commands are listed.

Figure 4: testServerAxis.bat 100 commands

```

C:\Windows\system32\cmd.exe

--- 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

---

**Note:** After you enter a transaction, the result is displayed followed by a restating of the current command set. You might need to scroll back a bit to see your results.

---

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 5: testServerAxis Response to command 121: License information**

```

C:\Windows\system32\cmd.exe
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="12/1/2014" />
    <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_USAGE_ENFORCED" value="false" />
    <LicenseProperty name="LICENSE_USAGE_NAME" value="Corticon" />
    <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>
</CcServerInfo>

```

**Figure 6: testServerAxis Response to command 121: Info on one of the deployed Decision Services**

```

C:\Windows\system32\cmd.exe
Enter transaction number:
121
<CDDs>
  <CDD path="C:/Users/g saintma/Progress/CorticonWork_5.4/cdd/OrderProcessing.cdd" timestamp="1398334220000">
    <DecisionService name="ProcessOrder">
      <EDS path="C:/Users/g saintma/Progress/CorticonWork_5.4/SER/CcServerSandbox/DoNotDelete/Decis
1405544017213.190572/ProcessOrder_v1_10.eds" />
      <ERF path="C:/Users/g saintma/Progress/CorticonWork_5.4/Samples/Rule Projects/OrderProcessing
estamp="1398334220000" />
      <ERSFiles>
        <ERS path="C:/Users/g saintma/Progress/CorticonWork_5.4/Samples/Rule Projects/OrderProcessi
imestamp="1398334220000" activityname="_Users_g saintma_Progress_CorticonWork_5_4_Samples_Rule_Projec
ing_Order_ers" activitynodename="ProcessOrder" step="1" />
      </ERSFiles>
      <ECORE path="C:/Users/g saintma/Progress/CorticonWork_5.4/Samples/Rule Projects/OrderProcessi
timestamp="1398334220000" />
      <LogPerThread name="false" />
      <DeploymentType value="RULEFLOW" />
      <DeployedAsTestDecisionService value="false" />
      <AutoReload value="false" />
      <MinPoolSize value="1" />
      <MaxPoolSize value="1" />
      <MessageStructure />
      <DatabaseAccess />
      <DatabaseReturnEntitiesMode value="ALL" />
      <DatabaseProperties />
      <AvailableReactors value="1" />
      <TotalReactors value="1" />
    </DecisionService>
  </CDD>
  <CDD path="C:/Users/g saintma/Progress/CorticonWork_5.4/cdd/TradeAllocation.cdd" timestamp="1398334220000">
    <DecisionService name="AllocateTrade">

```

**Figure 7: testServerAxis Response to command 121: Additional Server information**

```

Select C:\Windows\system32\cmd.exe

</CDD>
</CDDs>
<NonCDDs />
<RegisteredTrackingAttributes />
<ServiceStartups>
  <ServiceStartup ServiceName="UpdateService" ServiceValue="false" />
  <ServiceStartup ServiceName="PerformanceMonitoring" 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 8: testServer.bat**

```

testServer.bat - Notepad
File Edit Format View Help

echo off

set CORTICON_JARS=..\lib
set JAVA_HOME=..\JRE

call ..\..\bin\corticon_env.bat

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

echo on

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

pause

```

When executed, the `testServer.bat` opens a Windows console, loads all the classes, and then starts in the JVM, displaying the 100 series section of the API commands, as shown below:

**Figure 9: testServer.bat Windows console**

```

C:\Windows\system32\cmd.exe
Defining Corticon 5.4 runtime environment.

C:\Program Files\Progress\Corticon 5.4\Server\bin>CALL "%JRE%\bin\java.exe" -cp ..\lib\CcConfig.jar;..\lib\CcServer.jar;..\lib\CcLicense.jar;..\lib\CcExtensions.jar;..\lib\Cc118nBundles.jar;..\lib\CcThirdPartyJars.jar;..\lib\ant-launcher.jar -DCORTICON_SETTING=INP -DCORTICON_HOME="C:\Program Files\Progress\Corticon 5.4" -DCORTICON_WORK_DIR="C:\Users\gsaintna\Progress\CorticonWork_5.4" -Xms200m -Xmx400m com.corticon.eclipse.server.core.CcServerApiTest inprocess

Starting Progress Corticon Server : 5.4.0.0 -b6259
Progress Corticon Server log level : VIOLATION
Progress Corticon Server log path : C:\Users\gsaintna\Progress\CorticonWork_5.4\logs
Progress Corticon Server sandbox location : C:\Users\gsaintna\Progress\CorticonWork_5.4\INP\CcServerSandbox

Transactions:
-1 - Exit Server Api Test

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

Enter transaction number:

```

Because this is the in-process server, you can see that the first action is Starting Progress Corticon Server.

**Note:** The server started by `testServer.bat` does not load any Decision Services.



---

## 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 four ways:

- **Publish Wizard** - The easiest method, introduced in Corticon 5.3. It is discussed in detail in the *Publish and Download Wizards* section of the *Deploying Corticon Ruleflows* chapter of the *Integration & 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 the Monitoring Server chapter of the *Integration & Deployment Guide*.
- **Java APIs** - This method requires more knowledge of the Java programming language, and is not discussed in this Tutorial.

All four 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 the *Corticon Studio Tutorial: Advanced Rule Modeling*, 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 `[CORTICON_WORK_DIR]\Samples\Rule Projects\Tutorial\Tutorial-Done`.

To create a new Ruleflow, see the topics *"Ruleflows" 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 Deployment Console tool's Decision Services on the Java Server

To start the Corticon Deployment Console, choose the Windows **Start** menu command **All Programs > Progress > Corticon 5.4 > 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 10: Left Portion of Deployment Console, with Deployment Descriptor File Settings Numbered

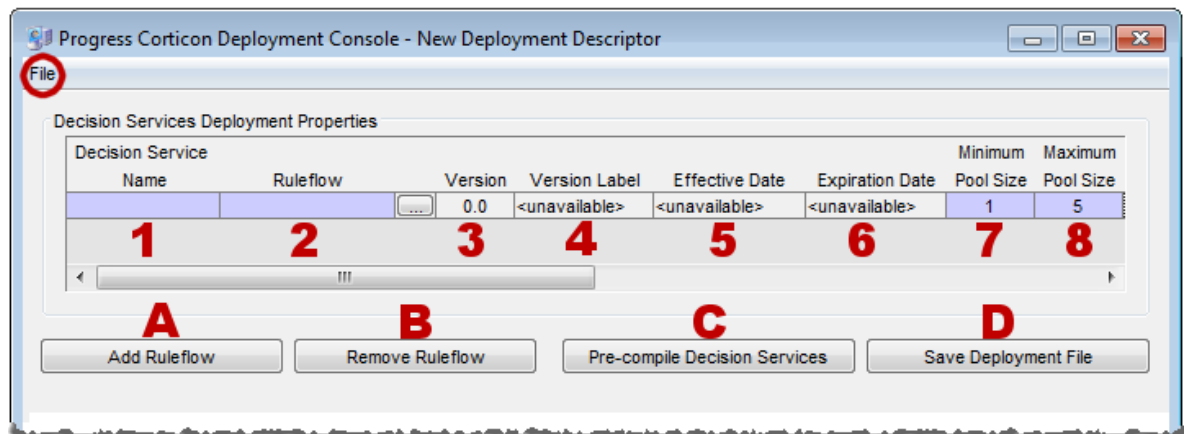
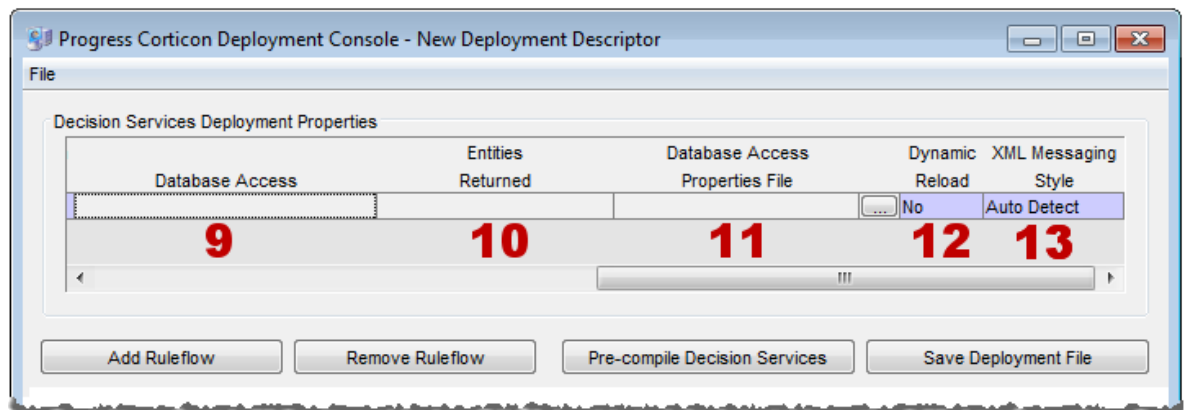


Figure 11: Right Portion of Deployment Console, with Deployment Descriptor File Settings Numbered




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

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

- To save the current file, choose (**File > Save**).
- To open an existing .cdd, choose (**File > Open**).
- To save a .cdd under a different name, choose (**File > Save As**).

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

1. **Decision Service Name** - A unique identifier or label for the Decision Service. It is used when invoking the Decision Service, either via an API call or a SOAP request message. See [Invoking Corticon Server](#) 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. **Minimum Pool Size** - The minimum number of instances or 'copies' created for a Decision Service when it is loaded by Corticon Server. Instances of a Decision Service are known as **Reactors** - These Reactors are placed in a pool, where they wait for assignment by Corticon Server to an incoming request, or they expire due to inactivity. The larger the pool size, the greater the concurrency (but greater the memory usage). The default value is **1**, which means that even under no load (no incoming requests) Corticon Server will always maintain one Reactor in the pool for this Decision Service.
8. **Maximum Pool Size** - The maximum number of Reactors Corticon Server can put into the pool for this Decision Service. Therefore, the number of Reactors that can execute concurrently is determined by the max pool size. If additional requests for the Decision Service arrive when all Reactors are busy, the new requests queue until Corticon Server can allocate a Reactor to the new transaction (usually right after a Reactor is finished with its current transaction). The more Reactors in the pool, the greater the concurrency (and the greater the memory usage). See [Performance and Tuning](#) chapter for more guidance on Pool configuration. The default value is **5**.

---

**Note:** Functions 9, 10, and 11 are active only if your Corticon license enables EDC, and you have registered its location in tool.


---

---

**Note:** If you are evaluating Corticon, your license requires that you set the parameter to 1.

---



9. **Database Access** - Controls whether the deployed Rule Set has direct access to a database, and if so, whether it will be read-only or read-write access.
10. **Entities Returned** - Determines whether the Corticon Server response message should include all data used by the rules including data retrieved from a database (**All Instances**), or only data provided in the request and created by the rules themselves (**Incoming/New Instances**).
11. **Database Access Properties File** - The path and filename of the database access properties file (that was typically created in Corticon Studio) to be used by Corticon Server during runtime database access. Use the adjacent  button to navigate to a database access properties file.
12. **Dynamic Reload** - If **Yes**, then Corticon Server will periodically look to see if a Deployment Descriptor file, or any of the Decision Service entries in that file, has changed since the `.cdd` was last loaded. If so, it will be automatically reloaded. The time interval between checks is defined by property `com.corticon.ccserver.serviceIntervals` in [Server properties](#). Even if **No**, Corticon Server will still use the most recent Ruleflow when it adds new Reactors into the pool.
13. **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 Pacific Application Server, the included startup scripts ensure the following sequence occurs upon launching Pacific 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 Pacific Application Server starts up.
2. Corticon Server starts up as a web service in Pacific 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 Tutorial, 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 Tutorial.

Of course, this location is fully configurable. See the Deploying Corticon Ruleflows chapter of the *Server Integration & Deployment Guide* for more details.

Now, when the startup sequence reaches step 3 above, 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.

---

## Consuming a decision service

---

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 Pacific 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 it returns.

For details, see the following topics:

- [Integrating and testing a Decision Service](#)
- [Path 1- Using Corticon Studio as a SOAP client to consume a Decision Service](#)
- [Path 2 - Using bundled sample code to consume a Decision Service](#)
- [Path 3 - Using SOAP client to consume a Decision Service](#)
- [Path 4 - Using JSON/RESTful client to consume a Decision Service on Java Server](#)
- [Troubleshooting](#)

## Integrating and testing a Decision Service

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 Tutorial 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 Studio: 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 *Studio Installation Guide* and the *Basic Tutorial for Corticon Studio – Modeling Rules* 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 new Java server test in Corticon Studio

Return to Corticon Studio, or reopen it if closed. Open `Cargo.ecore` and then, *without opening any Ruleflows*, open a new Test by selecting **File>New>Ruletest** from the Corticon Studio menubar.

For the Ruletest creation process outlined below, see also [Requesting List of Remote Decision Services](#):

1. Specify a filename for your Ruletest, and then click **Next**.
2. You will be asked to **Select Test Subject**. Be sure to select <http://localhost:8850/axis> in the **Remote Servers** box.
3. Select **Update List**. Corticon Studio will attempt to contact a Corticon Server instance at the location specified above. If a Corticon Server instance is running, it will respond with a list of available Decision Services, and display that list in the **Remote Decision Services** window.
4. Choose the Decision Service to invoke. In this Tutorial, we want `tutorial_example`.
5. Click **Next**.
6. Select the Vocabulary to use, as per normal Ruletest creation procedure.

---

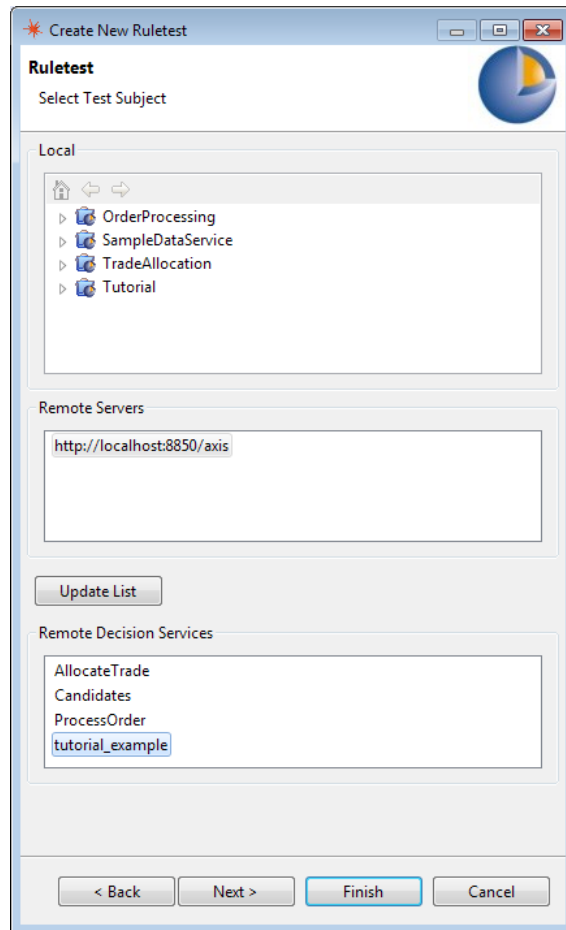
**Important:** Remember, even though we are using Corticon Studio to test, we are using its *remote* testing feature, which executes a Decision Service running on Corticon Server ("remotely"), not a Ruleflow open in Corticon Studio ("locally").

---

To keep this distinction clear, we are **not** going to open `tutorial_example.erf` in Corticon Studio – it is not necessary since we're really testing the Decision Service running on Corticon Server.

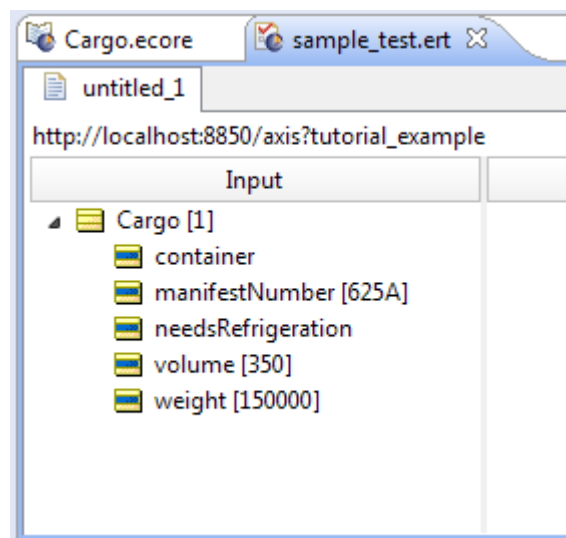
In step 1, you selected the default URL: Corticon Server running on `localhost`. If you want to change the URL to another address, see "Designer properties & settings" in *Server Integration & Deployment Guide* for more information about configuring Corticon Studio properties.

Figure 12: Requesting List of Remote Decision Services



Now, drag a `Cargo` entity from the Vocabulary to the Input pane. Enter sample data as shown:

Figure 13: Sample Data in a Studio Remote Ruletest

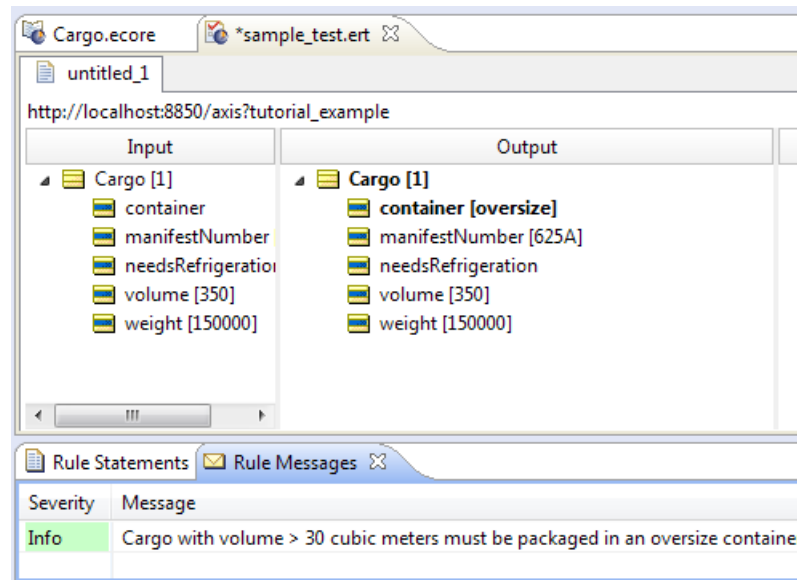


## Executing the remote test

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 14: Response from Remote Decision Service**



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

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

### Creating a Request Message for a Decision Service

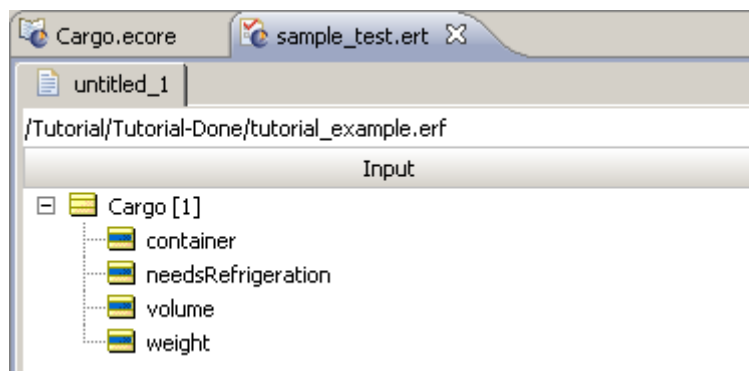
In this path, we will use a feature of Corticon Studio to generate a request message directly. The steps to accomplish this are:

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

1. Open Corticon Studio.
2. Open the Ruleflow you have deployed as a Decision Service. 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`.

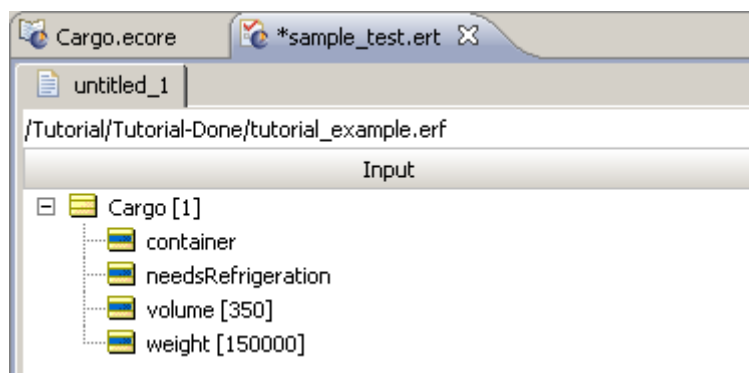
4. Create an Input test tree manually as in Option 1 above, or use menubar option **Ruletest>Testsheet>Data>Input>Generate Data Tree**, which produces the structure of a request message in the Input pane. One `Cargo` entity should appear in the Input pane, as shown below:

Figure 15: A New Test



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

Figure 16: Test with Data



6. Use Corticon Studio's menubar option **Ruletest>Testsheet>Data>Input>Export Request XML** to export this Input pane as an XML document. We will use this exported XML document as the body or "payload" of our request message. Give the export file a name and remember where you save it. We will assign a filename of `sample.xml` for purposes of this Tutorial.
7. Open `sample.xml` in any text editor. It should look very similar to the following figure:



Figure 17: Sample XML File Exported from a Studio Test

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <CorticonRequest xmlns="urn:Corticon" xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance" decisionServiceName=
  "InsertDecisionServiceName">
3   <WorkDocuments>
4     <Cargo id="Cargo_id_1">
5       <container xsi:nil="true" />
6       <needsRefrigeration xsi:nil="true" />
7       <volume>350</volume>
8       <weight>150000</weight>
9     </Cargo>
10  </WorkDocuments>
11 </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 18: Sample XML File with Changes Made

```

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

```

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

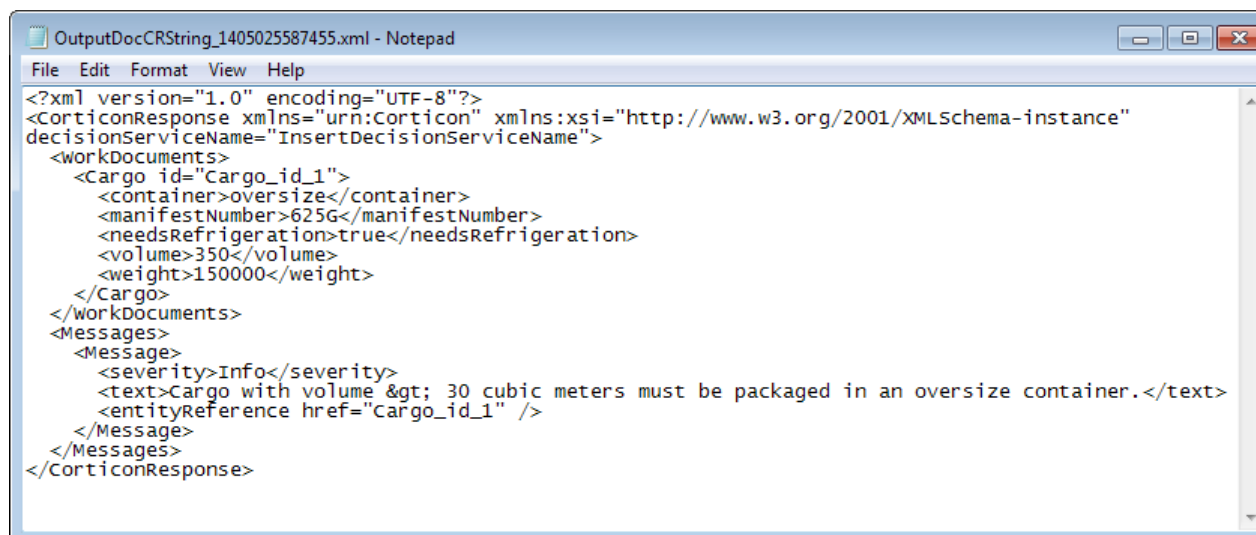
## Sending a request message to Corticon Server

The Corticon Server test scripts, described in [Testing the installed Corticon Server](#) on page 38, provides commands that run sample requests.

1. Start Corticon Server.
2. Launch the script `testServerAxis.bat` file, found 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_HOME]\Server\SER\output`, open the most recent `OutputDoc` file. The response from the Decision Service looks like this:

Figure 19: Corticon Response



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

- The `container` attribute has been assigned a value of `Oversize`.
- The input data `volume` and `weight` have been returned in the response message unchanged. If your rules do not change input data, it will be returned exactly as sent.
- The 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.

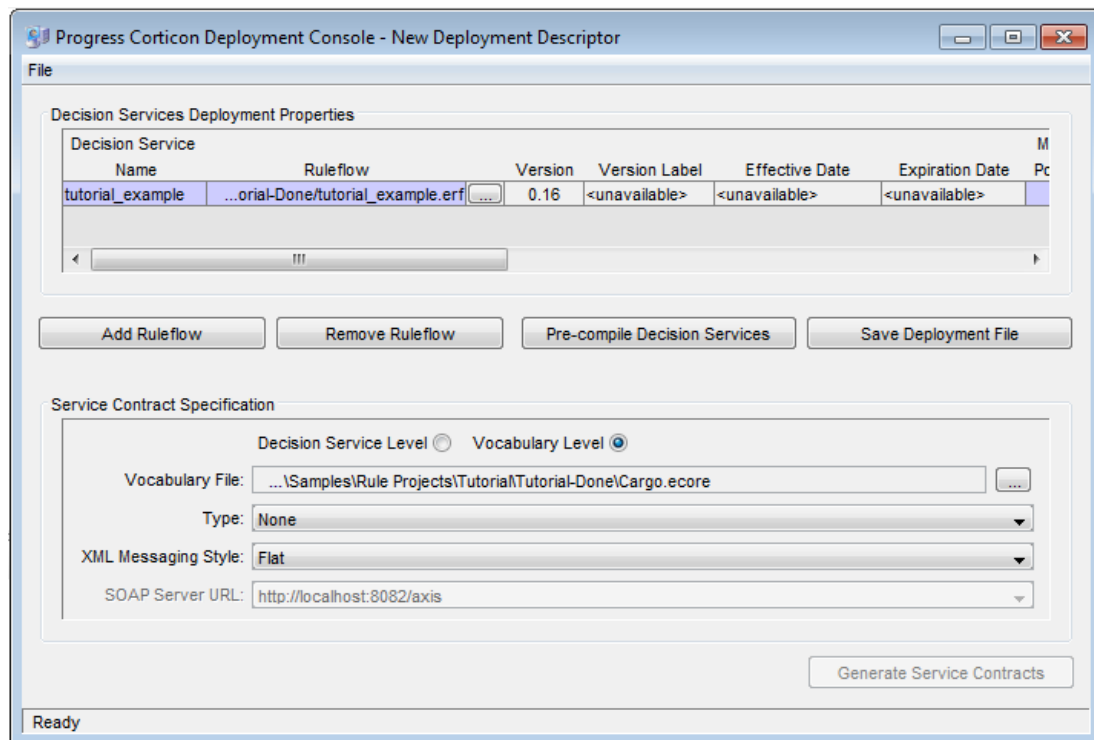
---

**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 service contract using the Deployment Console

Figure 20: Deployment Console's Service Contract Specification Window



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
-----

132 - Execute JSON REST request
Enter transaction number
```

4. Enter 132.
5. When prompted for **Input JSON File Path**, enter (or copy) the path to the sample:

```
C:\Users\{user}\Progress\CorticonWork_5.4\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

The *Rule Modeling Guide* contains an extensive chapter on *Troubleshooting*, including tips for troubleshooting rules in Corticon Studio, as well as problems encountered during Decision Service consumption. Refer to the *Rule Modeling Guide* for more details.

When the default port settings, conflict with the default port setting of Corticon's Pacific 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*.



---

# Modeling and managing rules using Server Console

---

The Progress Corticon Server Console enabled rule modelers to make some changes to Rulesheets. The 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 Server Console](#) on page 71 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:

- [Lifecycle management Through the Server Console](#)
- [Creating a new decision service version](#)
- [Opening the new version](#)
- [Modifying the new version](#)
- [Promoting the new version to live](#)
- [Removing the version from live](#)
- [Telling the server where to find Deployment Descriptor files](#)

## Lifecycle management Through the 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.

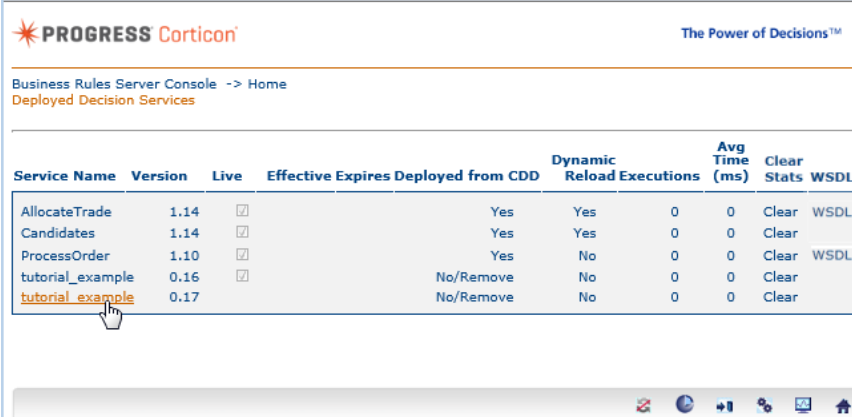
## Creating a new decision service version

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 21: Server Console with a new version of `tutorial_example` shown**



Service Name	Version	Live	Effective Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats	WSDL
<a href="#">AllocateTrade</a>	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	WSDL
<a href="#">Candidates</a>	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	
<a href="#">ProcessOrder</a>	1.10	<input checked="" type="checkbox"/>		Yes	No	0	0	Clear	WSDL
<a href="#">tutorial_example</a>	0.16	<input checked="" type="checkbox"/>		No/Remove	No	0	0	Clear	
<a href="#">tutorial_example</a>	0.17	<input type="checkbox"/>		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 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 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.

## Using the Corticon Server Console

---

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

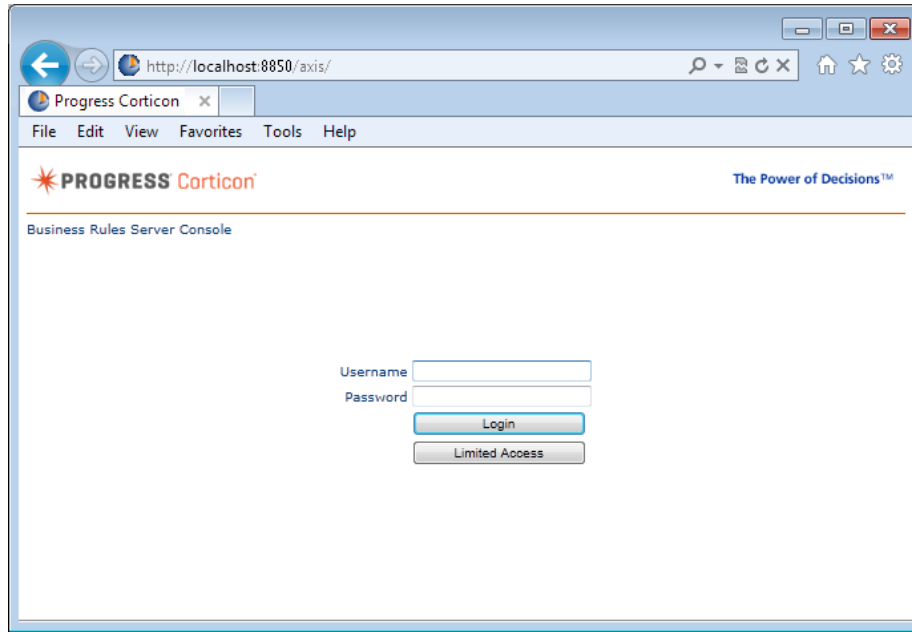
For details, see the following topics:

- [Launching and logging in to Corticon Server Console](#)
- [Using LDAP authentication in Server Console](#)
- [Console main menu page](#)
- [Decision Services page](#)
- [Deploy Decision Service page](#)
- [Configure Rules Server](#)
- [Monitor Rules Server page](#)
- [WSDLs](#)

## 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 Pacific Application Server, and the default URL is <http://localhost:8850/axis>. The Corticon Server Console login page opens, as shown:

Figure 22: 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 23: Server Console User Rights

	guest	tester	modeler_limited	modeler_full	admin
<b>Decision Services List</b>					
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
<b>Deploy Decision Service</b>					
Allow user to deploy a new Decision Service	N	N	N	N	Y
<b>Configure Rules Server</b>					
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
<b>Monitor Rules Server</b>					
View contents of Rules Server	N	N	N	N	Y
<b>WSDL</b>					
View WSDL	N	N	N	N	Y
<b>Decision Service Details</b>					
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
<b>Ruleflow Editor</b>					
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
<b>Rulesheet Editor</b>					
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 24: CcUsernamePassword.xml

```

CcUsernamePassword.xml x
1 <?xml version='1.0' encoding='utf-8'?>
2 <serverconsole-users>
3   <user username="" password="" rights="guest"/>
4   <user username="tester" password="tester" rights="tester"/>
5   <user username="modeler1" password="modeler1" rights="modeler_limited"/>
6   <user username="modeler2" password="modeler2" rights="modeler_full"/>
7   <user username="admin" password="admin" rights="admin"/>
8   <user username="administrator" password="changeme" rights="admin"/>
9 </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
<code>ldap.base.provider.url</code>	URL for connection to a directory service.
<code>ldap.base.dn</code>	Parent Domain for the specified directory.
<code>ldap.security.principal</code>	Qualified username or e-mail address of admin user.
<code>ldap.security.credentials</code>	Password of the user
<code>ldap.user.base</code>	Location to search for users.
<code>ldap.group.base</code>	Location to search for groups.
<code>ldap.user.search</code>	Search field for user.
<code>ldap.group.search</code>	Search field for group.
<code>ldap.group.mappings</code>	Mapping of Corticon Server Console Roles with Active Directory groups.

Property	Description
<code>ldap.check.all.directories</code>	Property to authenticate user against all specified directories in <code>ldap.server.ids</code>
<code>ldap.server.ids</code>	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}/
ldap.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, authentication 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.

---

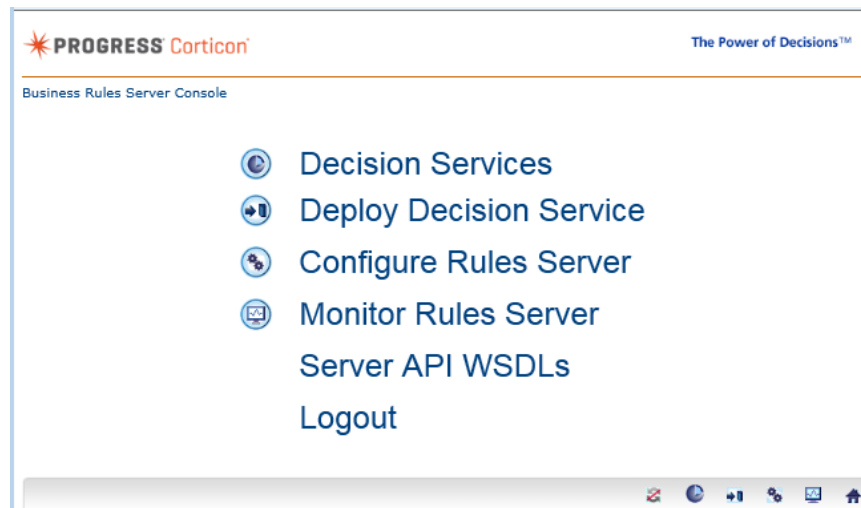
**Note:** You can revert to the non-LDAP authentication mechanisms by removing (or relocating) the `ldap.properties` file. Then, when you restart the Corticon Server to implement the changes, user connections that were dropped will need to use the local credentials (in `CcUsernamePassword.xml`) to connect 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 25: 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 <a href="#">Decision Services</a> menu.
	Show the <a href="#">Deploy Decision Service</a> menu.
	Show the <a href="#">Configure Rules Server</a> menu.
	Show the <a href="#">Monitor Rules Server</a> menu.
	Show the <b>Home</b> 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 26: the Decision Services Page**




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	WSDL
AllocateTrade	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	WSDL
Candidates	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear	
ProcessOrder	1.10	<input checked="" type="checkbox"/>		Yes	No	0	0	Clear	WSDL
tutorial_example	0.16	<input checked="" type="checkbox"/>		No/Remove	No	0	0	Clear	
<a href="#">tutorial_example</a>	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 91.
- **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 91.
- **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.

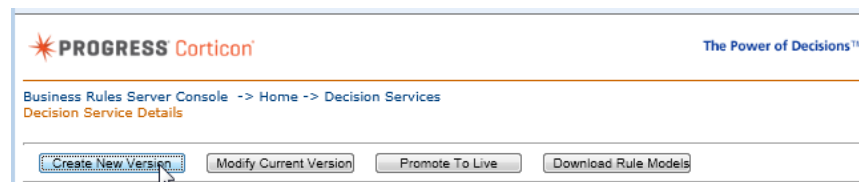
If the Decision Service was deployed by a `.cdd` file, then the contents of the `.cdd` are displayed on this page. If the Decision Service was deployed by the **Deploy Decision Service** page, then the information entered on that page (which is persisted in `serverstate.xml`) is displayed. The information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

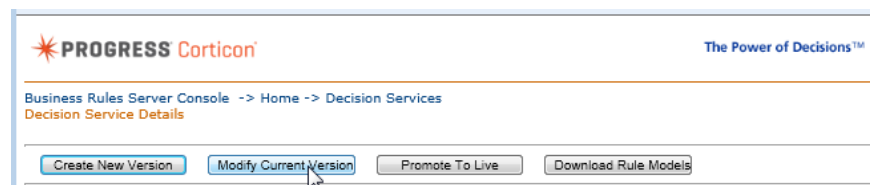
The four buttons at the top of a Decision Service information page let you perform the following actions:

- **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.

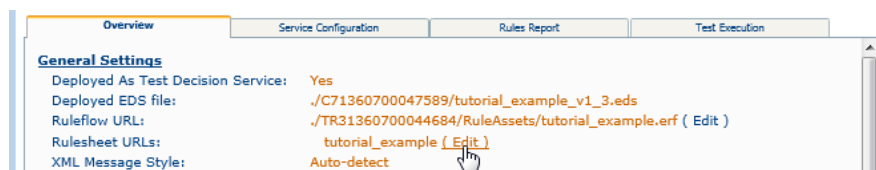
- **Modify Current Version** - Click the button, as shown:



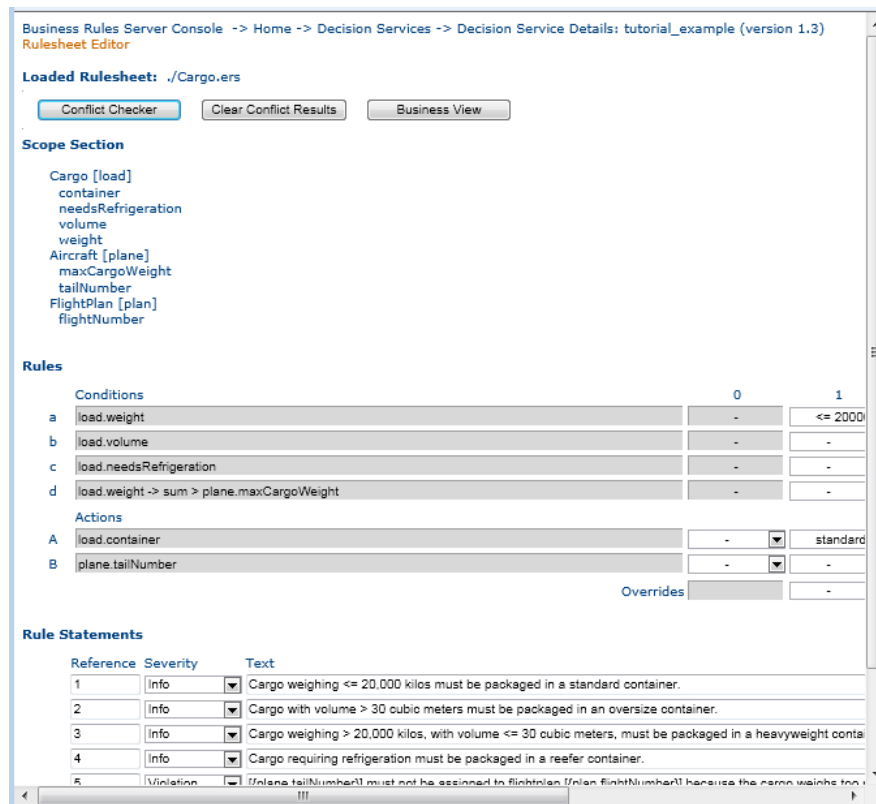
Opens a panel that defines a new minor version of the Decision Service which you can edit.

Click on an editable item to open its associated asset.

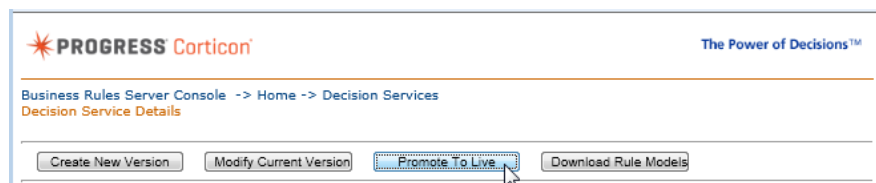




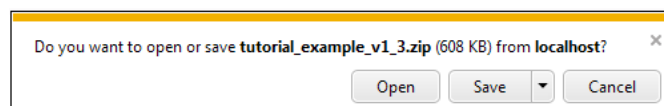
On the selected asset -- a Rulesheet in this example -- adjust selected parameters, click to check for conflicts, and see the alternative Business View.



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



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



## TABS

### Overview tab

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

**Figure 27: Overview Tab**

The screenshot displays the Progress Corticon Business Rules Server Console interface. At the top, the Progress Corticon logo and tagline "The Power of Decisions™" are visible. Below the breadcrumb navigation "Business Rules Server Console -> Home -> Decision Services", the "Decision Service Details" page is shown. A notification states "Decision Service has been updated:" with details: "Restrict Warning RuleMessages: No" and "Restrict Violation RuleMessages: Yes". Action buttons include "Create New Version", "Modify Current Version", "Promote To Live", and "Download Rule Models". The "Overview" tab is selected, showing "General Settings", "Pool Settings", "Database Access Settings", and "Monitored Attributes".

**General Settings**

Deployed As Test Decision Service:	Yes
Deployed EDS file:	./C7_1391723118140.437168/ProcessOrder_v2_0.eds
Ruleflow URL:	./TR0_1391723114601.366515/RuleAssets/Order.erf ( Edit )
Rulesheet URLs:	ProcessOrder ( Edit )
XML Message Style:	Auto-detect
Execution Log Level:	RULETRACE
Execution Log Path:	C:/Corticon/logs/ProcessOrder_v2_0
Execution Log Per Thread:	Yes
Restrict Info RuleMessages:	(System Default)
Restrict Warning RuleMessages:	No
Restrict Violation RuleMessages:	Yes
Dynamic Reload:	Yes
Loaded from CDD file:	No
Deployment Timestamp:	Feb 24, 2014 1:08:33 PM
Ruleflow Timestamp:	Feb 6, 2014 4:45:14 PM
EDS Timestamp:	Feb 6, 2014 4:45:18 PM
Total Number of Rules Deployed:	6
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 / 1

**Database Access Settings**

Database Access Mode:	None
-----------------------	------

**Monitored Attributes**

No Monitored Attributes Registered

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 28: Service Configuration Tab**

**PROGRESS Corticon** The Power of Decisions™

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

Create New Version Modify Current Version Promote To Live Download Rule Models

Overview **Service Configuration** Rules Report Test Execution

Current Rule Asset URL: ./C7\_1391723118140.437168/ProcessOrder\_v2\_0.eds

Rule Asset URL:  Browse...

Minimum Pool Size:

Maximum Pool Size:

XML Message Style: Auto-detect ▼

Execution Log Level: (System Default) ▼

Execution Log Path: C:/Corticon/logs/ProcessOrder\_v2\_0

Execution Log Per Thread: Yes ▼

Restrict Info RuleMessages: (System Default) ▼

Restrict Warning RuleMessages: No ▼

Restrict Violation RuleMessages: Yes ▼

Database Access Mode: None ▼

Database Access Entities Returned Mode: All Entities ▼

Current Properties File Location:

Database Access Properties File:  Browse...

Update

No Monitored Attributes Registered

Monitored Attribute	Analysis Bucket
<input type="text"/>	<input type="text"/>

Add

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 [Deploying Corticon Ruleflows](#) chapter 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.

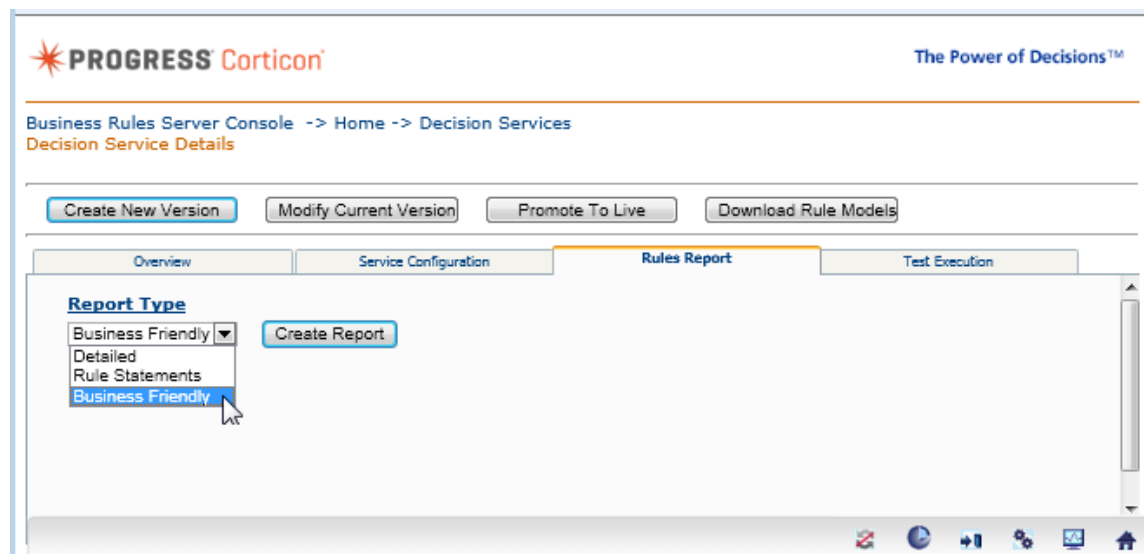
For example, the Ruleflow created in the *Basic Rule Modeling Guide* assigns values such as `oversize` and `reefer` to attribute `Cargo.container`. To monitor this attribute, enter the fully qualified attribute name (including the entity) in the **Monitored Attribute** box and enter the exact value or 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 29: 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:

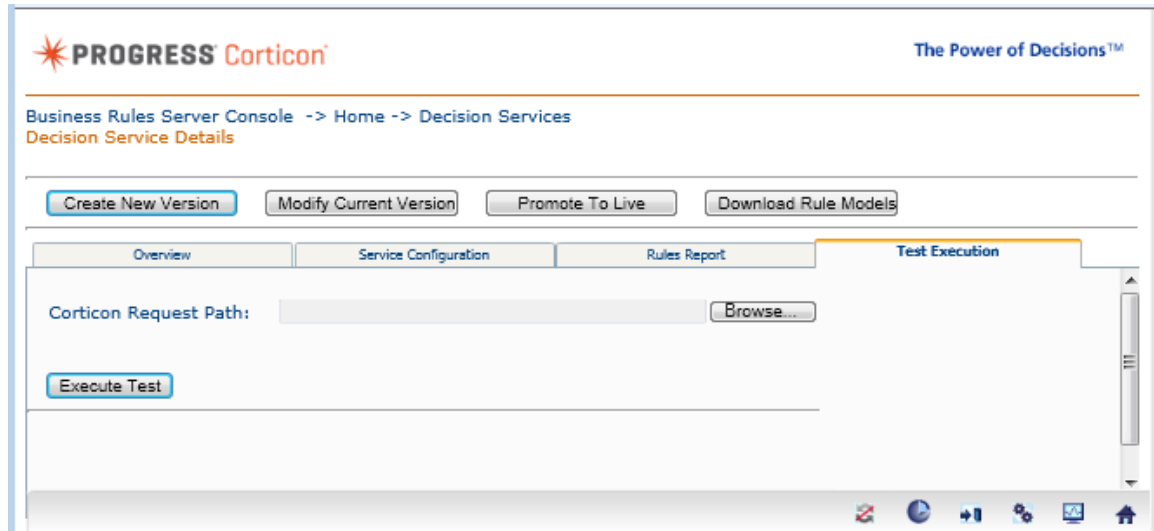
**Figure 30: Business Friendly Report**



## Test Execution Tab

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

Figure 31: Test Execution Tab



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

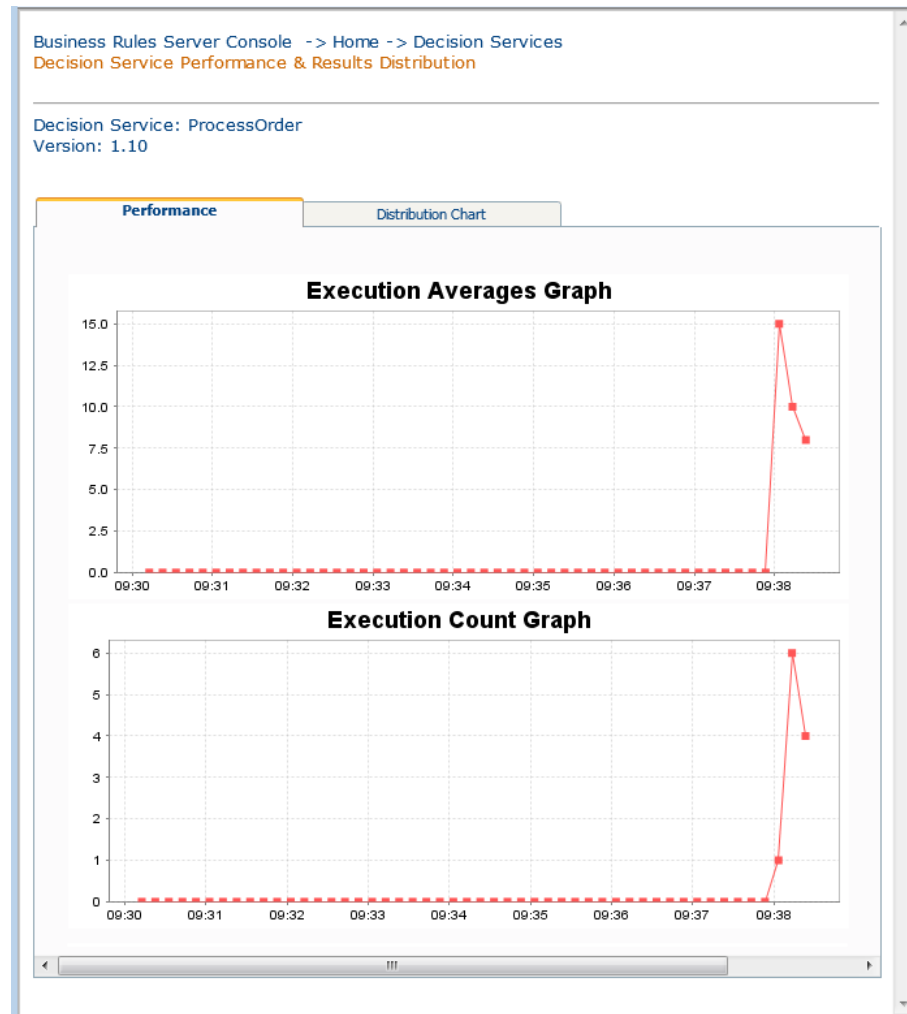
To access the visualizations of execution and distribution data, click on a decision service line's Execution or Avg Time value, as shown:

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 32: Performance Tab: Execution Average Time &amp; Count Graphs



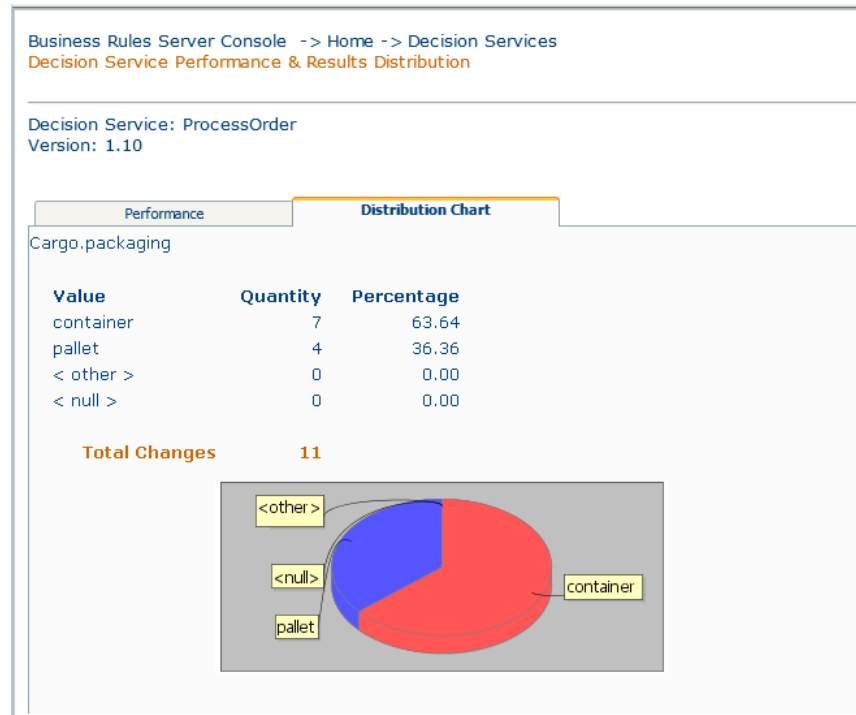
The [Decision Service Performance Monitoring Service](#) and [Decision Service Time Interval Performance Analysis Service](#) must be **On** for this feature to work. To turn these services **On**, go to the **Configure Decision Service** page, *Decision Service Options* tab.

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 overridden 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 33: Distribution Chart Tab: Results of Monitored Attributes



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 Server Console, instead of using the *Deployment Console* to create a `.cdd` file. Server Console uses standard Server APIs to perform the deployment.

To deploy Decision Services from the Server Console, you must use the pre-compile option in the *Deployment Console* to create an `.eds` file. Only an `.eds` file can be deployed using Server Console.

---

**Important:** Only an `.eds` file can be deployed using Server Console.

---

## Server state

In Corticon Server, the deployment state of all Decision Services, regardless of how deployed, is persisted in a file named `ServerState.xml`, located in `{CORTICON_WORK_DIR}\SER\CcServerSandbox\DoNotDelete`.



The **Deploy Decision Service** page captures the same deployment information as a `.cdd`, and stores it in `ServerState.xml` file. These fields are described in the [Deploying Corticon Ruleflows](#) chapter above. One exception: to update a Decision Service, we assume you will 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.

When Corticon Server restarts, it reads the `ServerState.xml` file and redeploys any Decision Services included in it.

**Figure 34: Deploy Decision Service Page**

To use this page, you must pre-compile your `.erf` file into an `.eds` file, and use the `.eds` file in the **Rule Set (path)** field above. You cannot directly deploy uncompiled `.erf` files via Server Console. Enter the required deployment information in the provided boxes, and click **Deploy**.

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 35: 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" />
      </DecisionService>
    </CDD>
  </CDDs>
  <NonCDDs>
  </NonCDDs>
</CcServerInfo>
</ServerState>
```

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

**Note:**

In Corticon Server 4.2 and earlier, Decision Services deployed via APIs remained deployed only for the duration of Corticon Server's session. If Corticon Server (or its host container) was stopped and restarted, Decision Services deployed previously by APIs would NOT redeploy automatically until those APIs were called again. Decision Services deployed via .cdd files, however, would redeploy automatically.

## Configure Rules Server

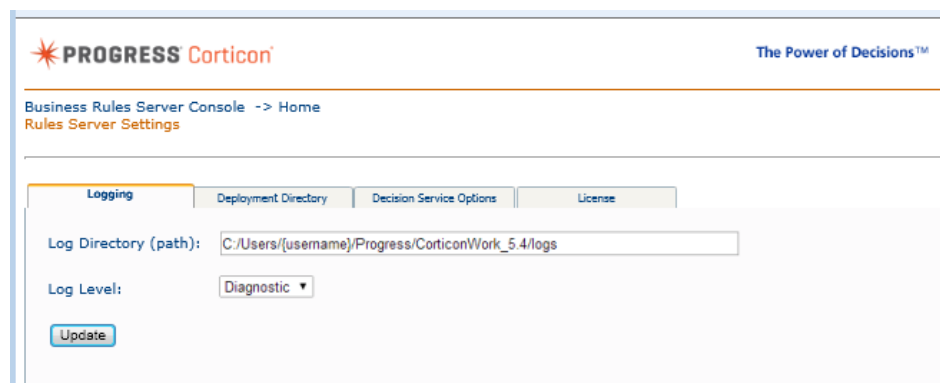
Click **Configure Rules Server** on the Home page (or  from the icon bar) to open the configuration pages.

**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:

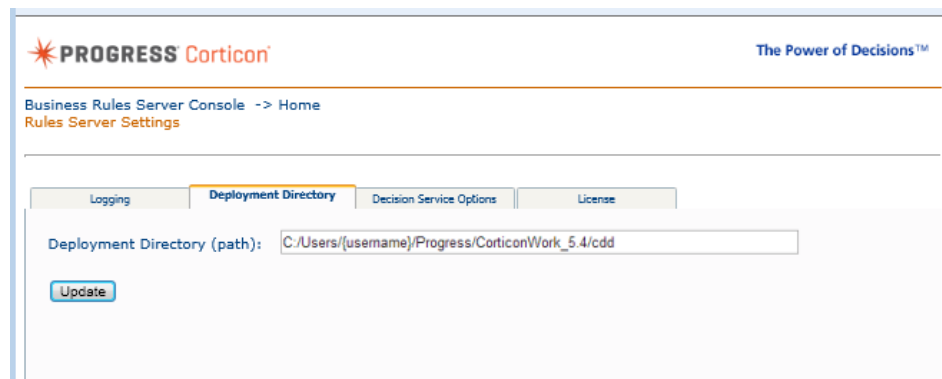
Figure 36:



## Deployment Directory tab

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

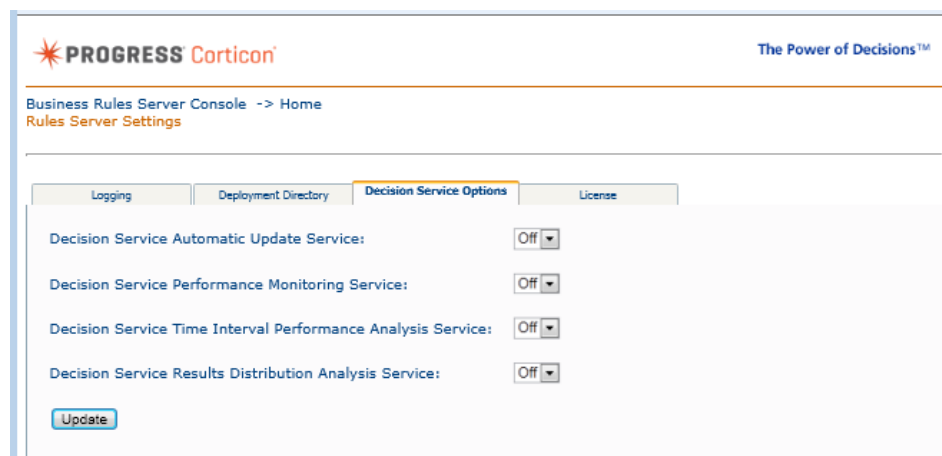
Figure 37: Deployment Directory tab



## Decision Service Options tab

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

Figure 38: Decision Service Options tab



By default, all 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`.

### ***Decision Service Performance Monitoring 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 Time Interval Performance Analysis Service***

This service is responsible for tracking and recording the time intervals across which other services operate. If this service is off, then the Performance Monitoring Service will not be able to display the data it collects because much of it is shown as averages over time. Generally, you will want to activate or deactivate both this and the Performance Monitoring service. The option chosen is recorded in `ServerState.xml`.

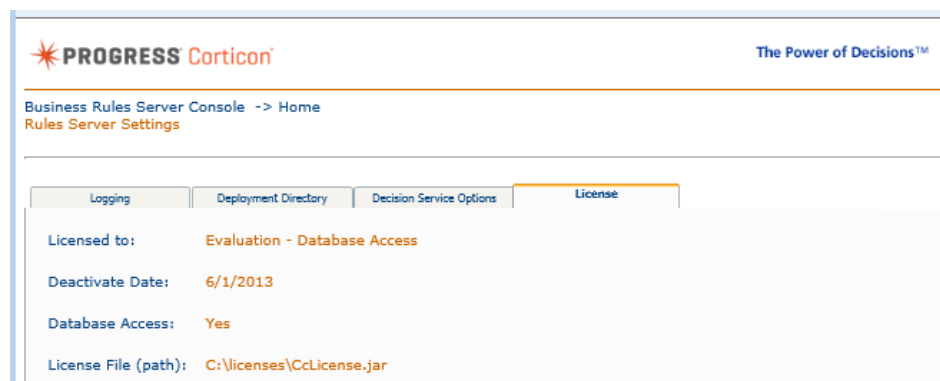
### ***Decision Service Results Distribution Analysis Service***

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`.

## 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 39: License tab**



This page is informational only.

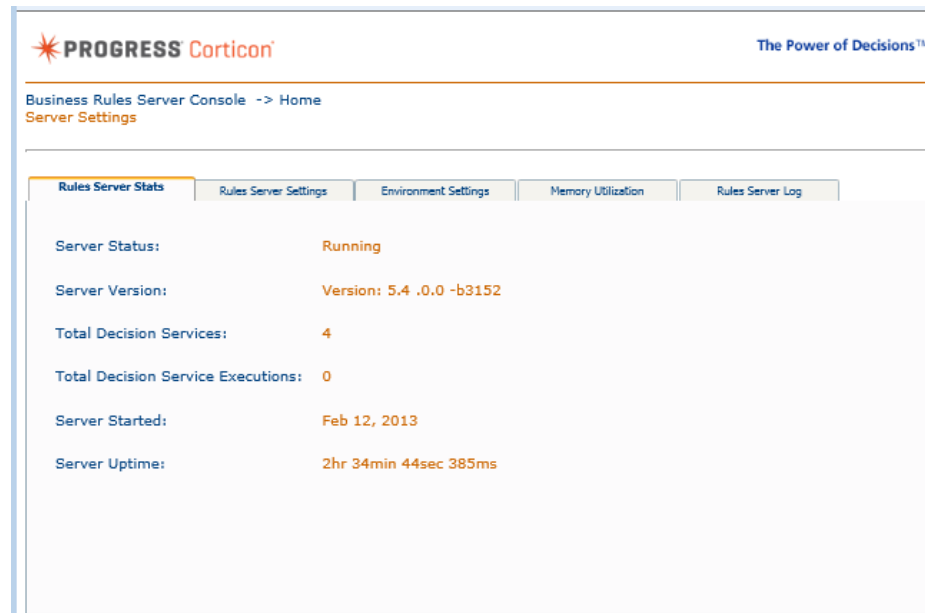
## Monitor Rules Server page

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:

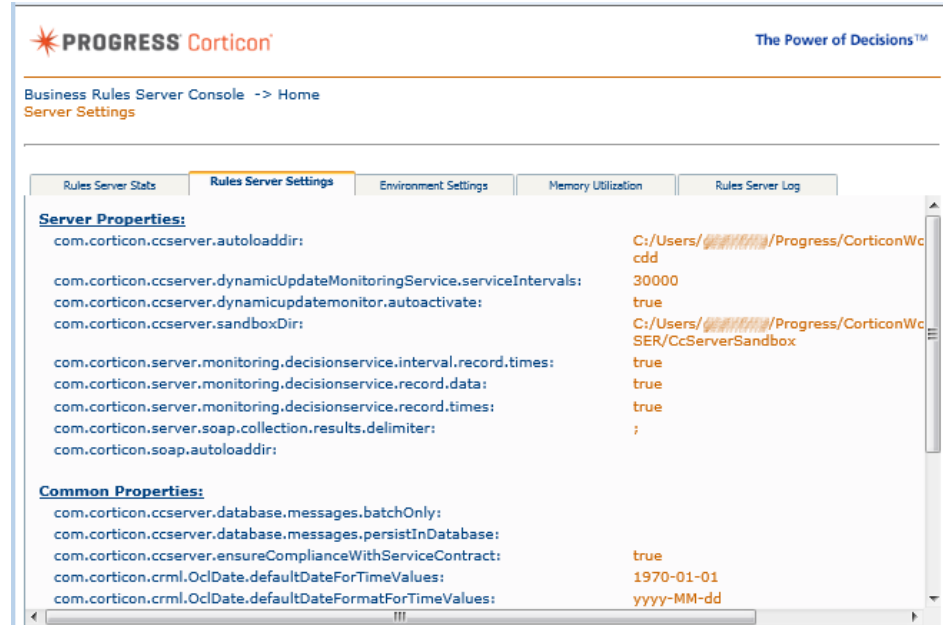
Figure 40: Rules Server Stats tab



## Rules Server Settings tab

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

Figure 41: Rules Server Settings tab



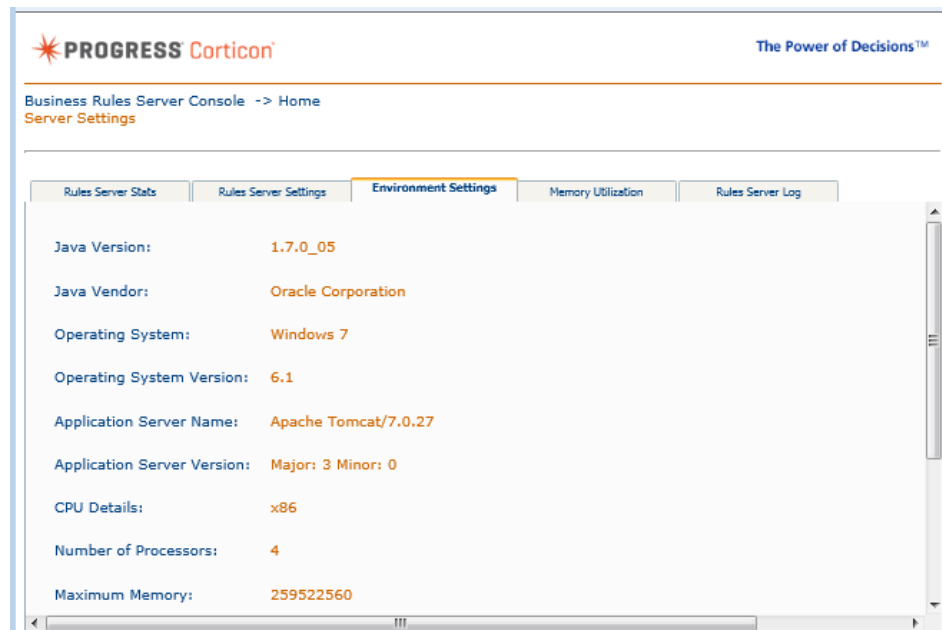
These settings are maintained in `CcConfig.jar` and user overrides are loaded last through the `brms.properties` file. (See [Configuring Corticon properties and settings](#) for more information about this file) . If a property setting (such as `logLevel`) has been overridden by a setting in Server Console (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 42: Environment Settings tab**

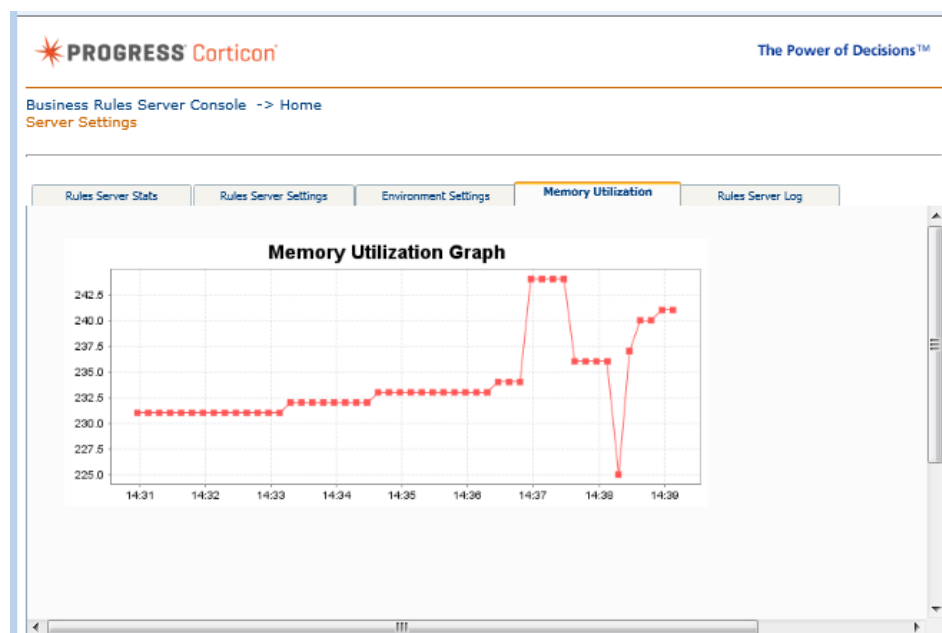


This information may be useful to Progress technical support in the event you need to contact us.

## Memory Utilization tab

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

**Figure 43: Memory Utilization tab**

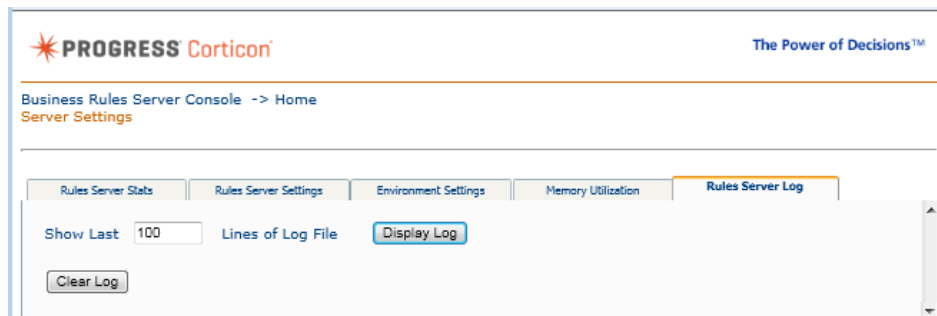


This graph will help you understand if Corticon Server's host environment is under-resourced, and may also be useful to Progress technical support in the event you need to contact us about memory problems.

## Rules Server Log

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

**Figure 44: Rules Server Log**

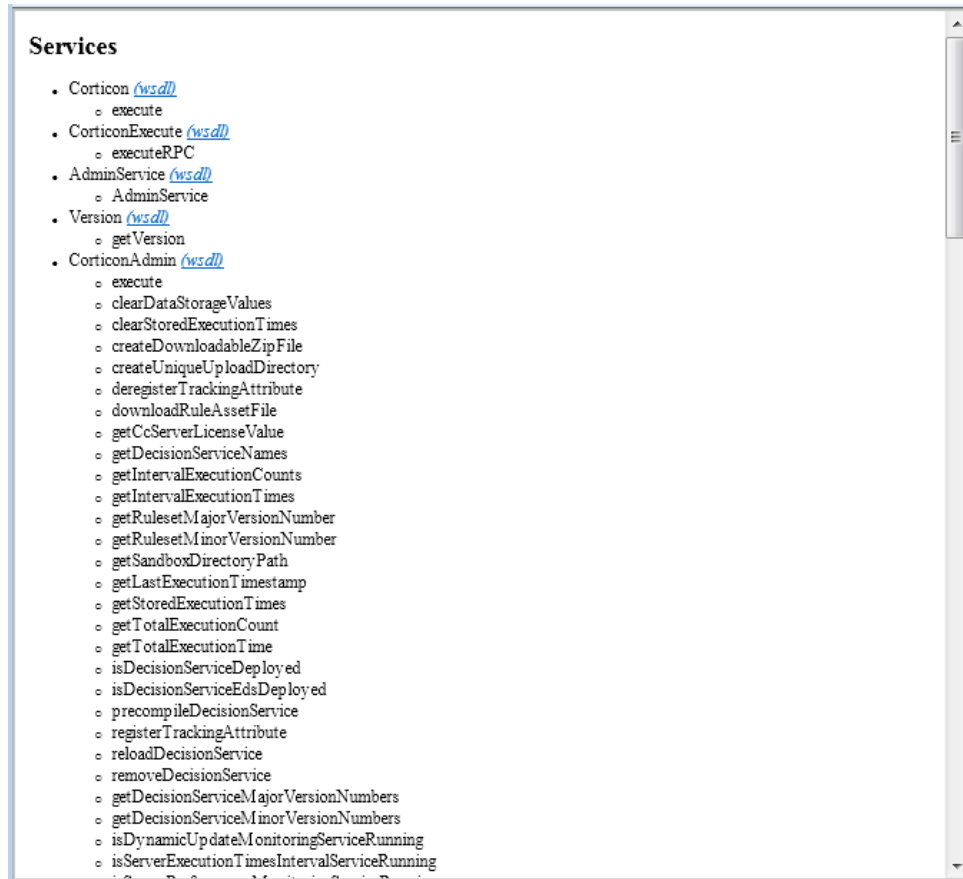


This tab allows you to read recent log entries without having direct access to the log file itself. You can also clear the log.

## WSDLs

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

Figure 45:



This URL is the home address for Server Console. You also can access this page directly (external of the Console) by using this address:

`http://<yourServer>:<yourPort>/axis/servlet/AxisServlet`



---

## 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	<p>Provides the source code used by <code>testServer.bat</code> to create a command interface for SOAP on the Server API.</p> <hr/> <p><b>Note:</b> Uses Corticon Server in the same JVM as the Java test code and cannot be executed against Corticon Server inside the Pacific Application Server (you can, of course, write your own Java client using these APIs to operate against any deployment).</p> <hr/>

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 <code>testDeployConsole.bat</code> to create a Windows console interface for Server API.
REST	CcServerRestTest.java	<p>Provides the source code used by <code>testServerRest.bat</code> to create a command interface for JSON/RESTful services on the Server API.</p> <hr/> <p><b>Note:</b> Uses Corticon Server in the same JVM as the Java test code and cannot be executed against Corticon Server inside the Pacific Application Server (you can, of course, write your own Java client using these APIs to operate against any deployment).</p> <hr/>

## Pacific Application Server Administration

---

Pacific Application Server administration involves the use of standard Tomcat utilities along with Progress customizations and new features.

For details, see the following topics:

- [Administrative scripts](#)
- [Administrative utilities](#)
- [Startup and shutdown sequences when using TCMAN](#)
- [Working with Instances](#)
- [Tomcat logging](#)
- [Updating the core Tomcat server in Pacific Application Server](#)
- [TCMAN Reference](#)

### Administrative scripts

This topic is a brief overview of some of the administrative utilities for the Pacific Application Server, which are implemented as scripts in the Tomcat server's `/bin` directory. Each script has a UNIX (`.sh` extension) and a Windows (`.bat` extension) version.

Notice that the Pacific Application Server implements the TCMAN command line utility as an interface to administrative functionality provided by core Tomcat scripts. You will find extended administrative functionality that is easier to find and to use if you run TCMAN rather than running the Tomcat scripts directly.

The following table lists the standard Tomcat utilities in `$CATALINA_HOME/bin` that were tailored by Progress to set the value of `CATALINA_HOME` and `CATALINA_BASE` to values appropriate for the Pacific Application Server and its instances.

**Table 3: Standard Tomcat utilities modified for Pacific Application Server**

<code>startup.sh(.bat)</code>	<p>Initiates the start up of the Tomcat server and its web applications.</p> <p>Functionality of this utility is also available through <code>tcman.sh start</code></p>
<code>shutdown.sh(.bat)</code>	<p>Initiates the Tomcat server's shutdown process, beginning with a graceful termination of any running web applications.</p> <p>Functionality of this utility is also available through <code>tcman.sh stop</code></p>
<code>version.sh(.bat)</code>	<p>A standard Tomcat script that displays version and system information.</p> <p>Functionality of this utility is also available through <code>tcman.sh info</code></p>
<code>configtest.sh(.bat)</code>	<p>Displays the Tomcat server's configuration and environment information.</p> <p>Functionality of this utility is also available through <code>tcman.sh test</code></p>

In addition, there are a number of supporting scripts that are called by Tomcat administrative utilities. Some of these supporting scripts are described in the following table.

**Table 4: Supporting administrative scripts for Pacific Application Server**

<code>catalina.sh(.bat)</code>	<p>Called by administrative utilities on startup or shutdown of the Tomcat server. Calls <code>setenv.sh(.bat)</code> (if it exists) and <code>setclasspath.sh(.bat)</code> to set environment.</p>
--------------------------------	---

<code>setclasspath.sh(.bat)</code>	Called by <code>catalina.sh(.bat)</code> to set <code>JAVA_HOME</code> or <code>JRE_HOME</code> if not already set. If they are set, it validates to ensure that the values are consistent with startup options.
<code>setenv.sh(.bat)</code>	<p>An optional file called by <code>catalina.sh(.bat)</code> on startup. Deploying this file is the standard way to customize the server environment for each running instance.</p> <p><code>setenv.sh</code> performs the following operations:</p> <ol style="list-style-type: none"> <li>1. Set JVM memory size and the CATALINA environment variables common to all Progress product Web applications.</li> <li>2. Load Java system properties from <code>conf/server.xml</code>.</li> <li>3. Find and run any <code>progress_product_setenv.sh(.bat)</code> files.</li> </ol>

In addition to the standard Tomcat administrative utilities and supporting scripts described above, Pacific Application Server The following table lists the utility scripts in the Pacific Application Server `/bin` directory that were added by Progress to the standard Tomcat server.

**Table 5: Pacific Application Server scripts**

<code>tcman.sh(.bat)</code>	Runs the TCMAN administrative utilities necessary to manage the Pacific Application Server. <code>tcman.sh</code> is a command-line wrapper for tools that are implemented in <code>tcmanager.sh</code> and it also manages the instance variables, <code>\$CATALINA_HOME</code> and in <code>\$CATALINA_BASE</code> .
<code>tcmanager.sh(.bat)</code>	Called by <code>tcman.sh</code> to implement the TCMAN administrative utilities necessary for managing the Pacific Application Server. This script is a wrapper for the built-in Tomcat utilities, including <code>startup.sh</code> , <code>shutdown.sh</code> , <code>version.sh</code> , and <code>configtest.sh</code> . In addition, TCMAN adds new administrative functionality specifically for the Pacific Application Server.
<code>progress_product_setenv.sh(.bat)</code>	<p>Called by the Tomcat <code>setenv</code> script to set Progress product-specific environment variables and make them available to Java processes.</p> <p>If this file exists in <code>\$CATALINA_HOME/bin</code> and in <code>\$CATALINA_BASE/bin</code>, the file in <code>\$CATALINA_BASE</code> takes precedence.</p>

## Administrative utilities

The Pacific Application Server (PAS) includes the following administrative utilities:

- **TCMAN**

The TCMAN command line utility is an interface to administrative functionality provided by core Tomcat scripts. Extended administrative functionality is easier to find and to use if you run TCMAN rather than running the Tomcat scripts directly.

- **JMX and JConsole**

PAS supports the use of The Java Management Extensions (JMX) technology, which facilitates dynamic access to applications and other resources. This access is accomplished through the use of Java objects called *Managed Beans*, or *MBeans*. One or more MBeans instrument a resource through the use of the MBean's attributes, actions (defined as methods), and notifications. For more detailed information on JMX and MBeans, see the Java documentation at <http://docs.oracle.com/javase/8/docs/technotes/guides/jmx/index.html>.

JConsole, which is an application included with Java, can be used to monitor and manage MBeans. JConsole can be used for development and debugging, but you should not use JConsole on a local production server because JConsole itself uses significant resources. Using JConsole with a remote connection avoids this problem, although this usage requires more attention to security.

## TCMAN

the Pacific Application Server implements the TCMAN command line utility as an interface to administrative functionality provided by core Tomcat scripts. You will find extended administrative functionality that is easier to find and to use if you run TCMAN rather than running the Tomcat scripts directly.

## JMX and JConsole

The Pacific Application Server supports the use of The Java Management Extensions (JMX) technology, which facilitates dynamic access to applications and other resources. This access is accomplished through the use of Java objects called *Managed Beans*, or *MBeans*. One or more MBeans instrument a resource through the use of the MBean's attributes, actions (defined as methods), and notifications. For more detailed information on JMX and MBeans, see the Java documentation at <http://docs.oracle.com/javase/8/docs/technotes/guides/jmx/index.html>.

JConsole, which is an application included with Java, can be used to monitor and manage MBeans. JConsole can be used for development and debugging, but you should not use JConsole on a local production server because JConsole itself uses significant resources. Using JConsole with a remote connection avoids this problem, although this usage requires more attention to security.

## Startup and shutdown sequences when using TCMAN

The startup and shutdown scripts included with Tomcat have been enhanced for the Pacific Application Server (PAS). In particular, the PAS startup and shutdown customizations add shell environment variables, like `CATALINA_HOME`, `CATALINA_BASE`, and `CATALINA_TMPDIR` to support the use of multiple instances.

You can run Tomcat startup scripts directly. However, invoking them indirectly through the Progress TCMAN utility is preferable because TCMAN combines and extends Tomcat utilities to provide PAS-specific administrative support, particularly with regard to correctly setting variables when you are running multiple instances.

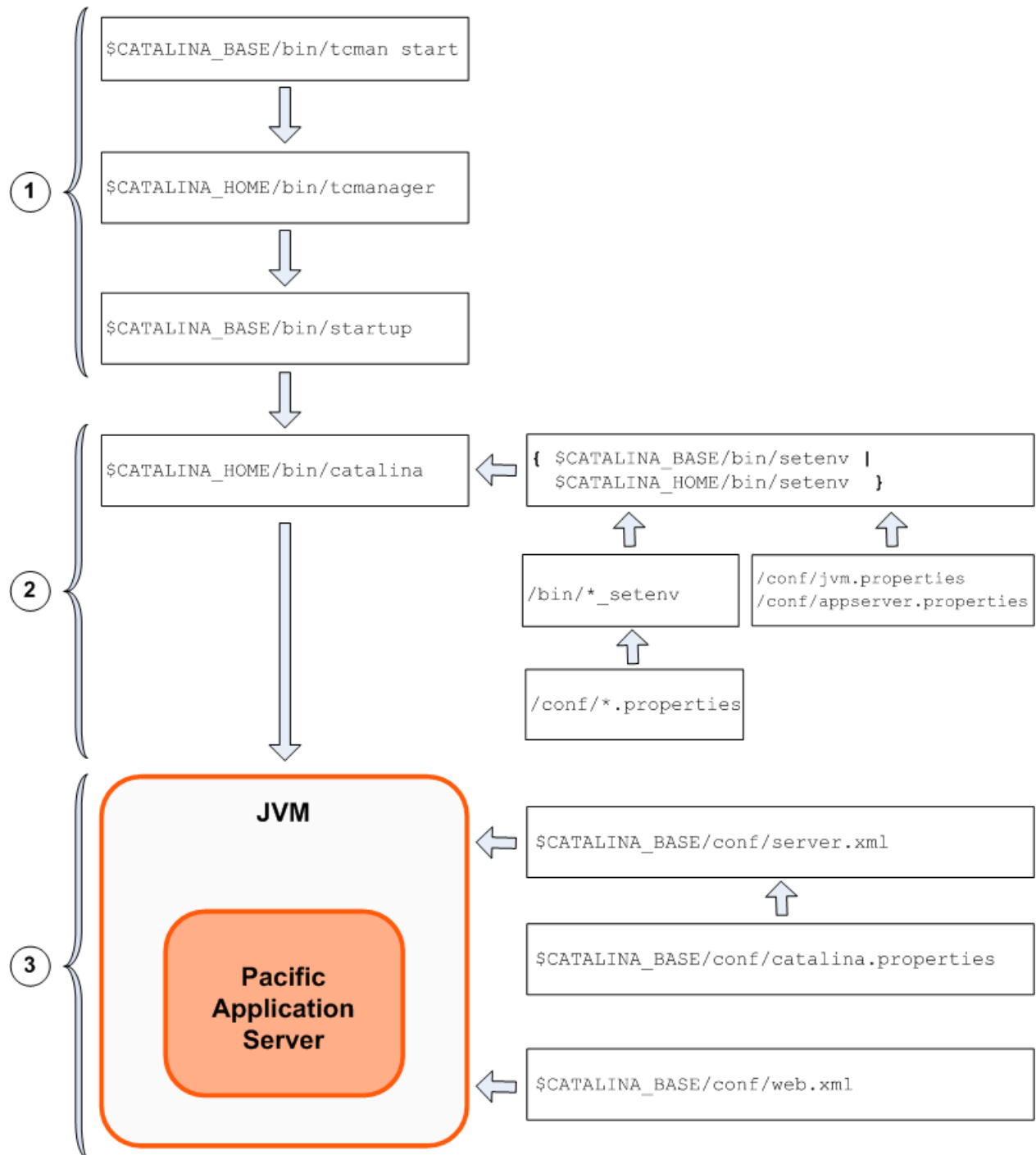
The following figure is an overview of the sequence that is initiated when you use the TCMAN utility to start a PAS server:

---

**Note:** Scripts in the `/bin` directories have a Windows version and a UNIX version. File names for the Windows version have a `.bat` extension. The UNIX scripts have a `.sh` extension. The extensions were omitted from the file names in the following figure for the sake of legibility.

---

Figure 46: PAS startup sequence



1. From a command prompt, you run the TCMAN start action.

You can run the `start` action from the `/bin` directory of the server or instance that you want to start. In addition, you can run the action with the `-I instance_name` option in order to explicitly specify which instance to start.

The `tcman` script sets the `CATALINA_HOME` and `CATALINA_BASE` variables. `$CATALINA_HOME` is the path of the parent directory of the core PAS server. `$CATALINA_BASE` is the path of the instance that you are starting. Startup utilities reference files in both locations.



After the variables are set, the `tcmn` script launches the `tcmanager` script, which, in turn, launches the Tomcat `startup` script that was specifically tailored for PAS.

2. After doing some PAS-specific tailoring, `startup` launches the `catalina` script. The initial function of the `catalina` script is to setup the runtime environment by running `setenv` scripts.

The figure above shows that `catalina` runs `setenv` from `$CATALINA_BASE/bin`. If that file does not exist, it runs the `setenv` in `$CATALINA_HOME/bin`.

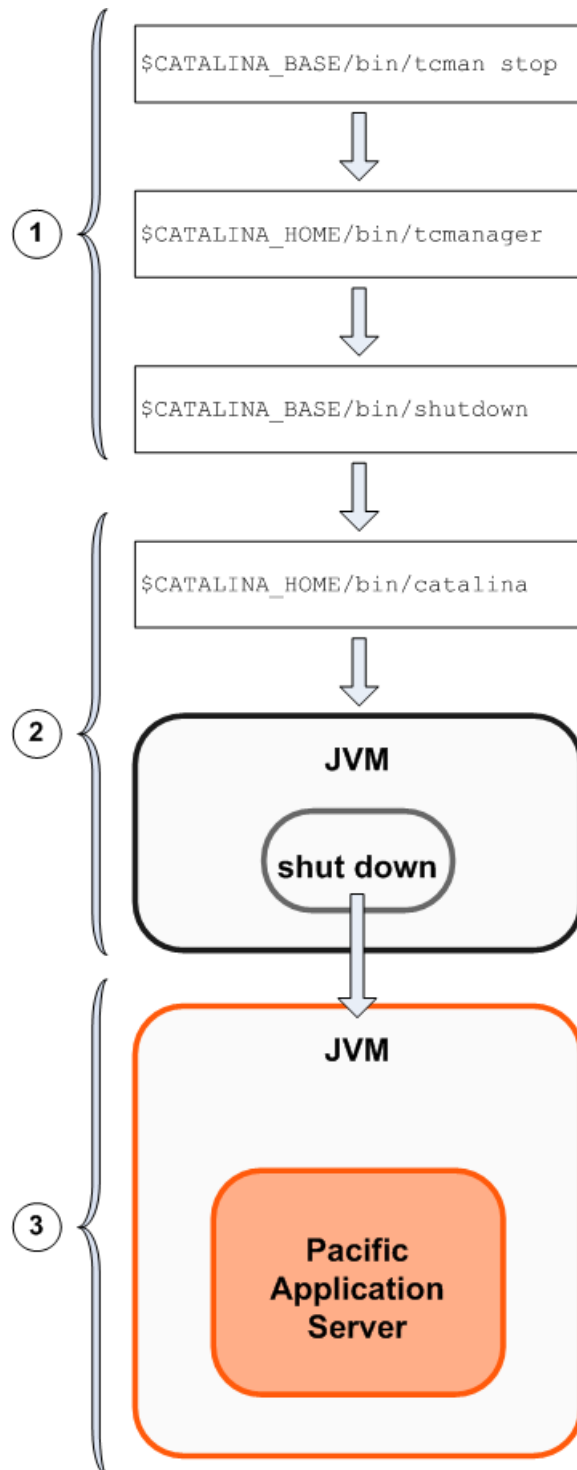
The `setenv` runs any environment setup scripts (`/bin/*_setenv`) that support specific Progress products. These scripts reference product properties files (`/conf/*.properties`). Next, `setenv` loads properties from the `/conf/jvm.properties` and the `/conf/appserver.properties` files.

After the `setenv` scripts run, `catalina` launches a Pacific Application Server hosted by a JVM.

3. The Pacific Application Server starts in a JVM after server configuration (from `$CATALINA_BASE/conf/server.xml`) and Web application information (from `$CATALINA_BASE/conf/web.xml`) are loaded.

The following figure is an overview of the sequence that is initiated when you use the TCMAN utility to stop a PAS server:

Figure 47: PAS shutdown



1. From a command prompt, you run the TCMAN `stop` action.

You can run the `stop` action from the `/bin` directory of the server or instance that you want to stop. In addition, you can run the action with the `-I instance_name` option in order to explicitly specify which instance to stop.

The `tcman` script launches the `tcmanager` script, which, in turn, launches the Tomcat shutdown script that was specifically tailored for PAS.

---

**Note:** If necessary, you can initiate a forced shutdown and kill the PAS process by using the `-F` option.

---

2. The Tomcat shutdown script launches the catalina script which starts a new JVM. The new JVM sends a shut down request via a Java server socket to the JVM that is running the PAS server.

---

**Note:** When you create an instance with the `TCMAN create` action, it is possible to set up a TCP port that can be used to stop an instance. However, the use of a TCP port to stop a server is a security risk and should only be used in an internal development environment.

---

3. The following processes are stopped:
  - a. All running Web applications
  - b. The PAS server process
  - c. The JVM that was hosting the PAS server
  - d. The JVM that issued the shut down request

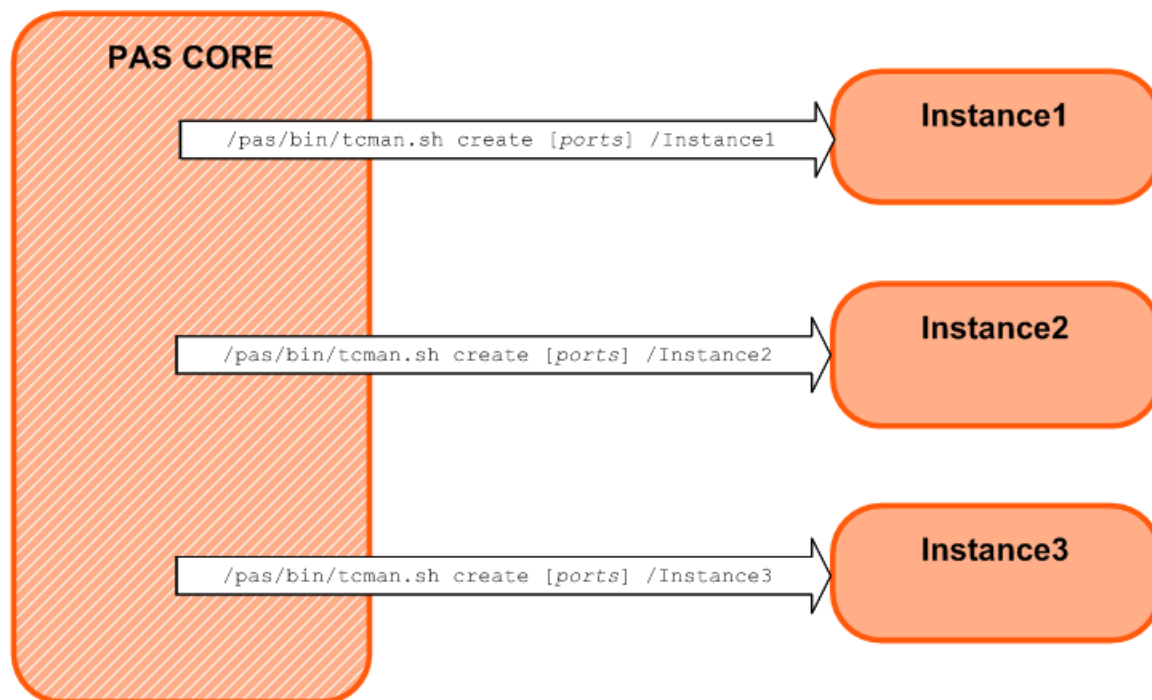
## Working with Instances

After you install the core Pacific Application Server, you can create an instance.

*Instances* are a standard Apache Tomcat feature. They allow you to create individual deployment and/or development servers that are based on the core Pacific Application Server that you installed.

The following figure illustrates the creation of multiple instances using the `TCMAN` command-line utility (with syntax simplified).

Figure 48: Generating PAS instances



Instances are independently running copies of the core Pacific Application Server. Each instance runs on its own JVM, has its own configuration with unique ports, and hosts its own web applications. However, each instance runs a Tomcat server that uses a number of common files from the same `$CATALINA_HOME` directory.

There are a number of advantages when you deploy your web applications to an instance of the Pacific Application Server, rather than deploying to the Pacific Application Server that you installed. This practice prevents accidental corruption of the core executables, configuration settings, and libraries. It also prevents accidental deletion of web applications if the core Pacific Application Server is removed when you uninstall a Progress PAS product.

Some additional advantages of instances are:

- Updates to the core Apache Tomcat server libraries and executables do not affect your web applications. You avoid the necessity of updating the applications and/or re-configuring them.
- You can establish different security policies for each of the instances.
- You can tailor the JVM for individual applications, since each instance runs in its own JVM with its own configuration.
- Instances provide you with quick way to create a test server for experimenting with new configurations and applications without the danger of permanently corrupting an existing server.
- You can package an instance as a Web application and deploy it to other PAS core servers.

You use `$CATALINA_HOME/bin/tcman.sh create` command to create a new instance.

When you create an instance, the root directory of the instance is assigned to the `CATALINA_BASE` environment variable within the scripts in its `/bin` directory. The root directory of the installed (core) Pacific Application Server is assigned to the `CATALINA_HOME` environment variable in the scripts in the instance's `/bin` directory. (Notice that the scope of these environment variables is limited to the context of an individual instance's `/bin` scripts.)

All instances of a core Pacific Application Server execute a set of common JAR files, scripts, and libraries from the following directories on the parent server:

- `$CATALINA_HOME/lib`
- `$CATALINA_HOME/common/lib`
- `$CATALINA_HOME/bin`

However, each instance is created with :

- A `$CATALINA_BASE/bin/` directory with its own copy of some of the scripts from the core PAS. These include scripts for start up, shut down, deployment, running TCMAN actions, and so on.
- A `$CATALINA_BASE/conf/` directory with its own copy of properties and configuration files.
- A `$CATALINA_BASE/webapps/` which initially only contains the `ROOT` Web application.
- A number of directories that are initially empty. These include `/logs`, `/temp`, `/work`, and `/common/lib`.

## Creating instances with TCMAN

Before you can create an instance of the Pacific Application Server (PAS) using the TCMAN command line utility, you must:

- Install the core Pacific Application Server
- Install JDK or JRE version 1.7 or later
- Set the `JAVA_HOME` or `JRE_HOME` environment variable to the JDK/JRE install directory .

---

**Note:** TCMAN is a Progress extension of the basic Tomcat administrative utilities. TCMAN simplifies instance creation and management.

---

An instance runs the Tomcat executable of a core PAS, but it runs in a separate JVM, is configured with its own unique ports, and other properties. (You should not attempt to use the installed, core PAS as a development or production server.) Instances allow you to run a variety of server configurations without corrupting the files in the core server. They also allow you to update the core server without re-deploying or re-configuring your Web applications.

To create an instance using the TCMAN utility:

1. Open a command shell and navigate to `$CATALINA_HOME/bin`.

`$CATALINA_HOME` is the directory where you installed the core Pacific Application Server.

2. Run `tcman.sh create basepath` (or `tcman.bat` on Windows systems) .

The `base_path` parameter specifies the path name where you will create the instance. It is the only required parameter for the `create` action. If you are creating multiple running instances, you should override the default port assignments by specifying the following parameters:

Options	Description
<code>-p port_num</code>	Specify the TCP port that listens for HTTP messages. The default is 8080.
<code>-P port_num</code>	Specify the TCP port that listens for HTTPS messages. The default is 8443.

You can also activate these ports:

Options	Description
<b>-s port_num</b>	Specify the TCP port to use to stop an instance. (Required on Windows systems, optional on UNIX )
<b>-j port_num</b>	Specify the TCP port that listens for AJP13 messages, an Apache protocol for handling requests from a web server to an application server. (Optional on both Windows and UNIX systems)

See [Create an instance \(create\)](#) on page 126 for information about other parameters.

3. (Optional) Deploy remote management applications from `$CATALINA_HOME/extras` to the instance.

Remote management applications are not pre-installed, and installing them is a security decision. For example, you might want to eliminate access to the configuration and control of instances by not deploying management applications to production servers, while deploying management applications to development servers.

To deploy a management application:

- a) Open a command shell and navigate to `$CATALINA_BASE/bin`.
- b) Run `tcman.sh deploy '$CATALINA_HOME/extras/admin_webapp.war'`.

The `admin_webapp.war` can be one of the following:

Options	Description
<b>host-manager.war</b>	A Tomcat administration application used to get server information and provide other functionality. It should not be necessary to deploy <code>host-manager.war</code> if you are using the TCMAN utilities.
<b>manager.war</b>	A Tomcat administration application which you must deploy in order to run some TCMAN actions. See the <a href="#">TCMAN Reference</a> on page 114 for information on which TCMAN actions require deployment of <code>manager.war</code> .
<b>Progress applications</b>	Progress products can have web applications that enable the use of their own administrative tools.

For example the following command line creates an instance of `/psc/pashome` in `/psc/acme1` and specifies its ports:

```
$: /psc/pashome/bin/tcman.sh create -p 8501 -P 8601 -s 8701 /psc/acme1
Server instance acme1 created at /psc/acme1
```

## Instance management with TCMAN

TCMAN includes actions for configuring, starting, stopping, monitoring, and deleting instances.

The following table is a brief description of the instance management actions that you can perform with TCMAN. Entries link to the reference topics that provide more details, syntax, and examples.

Action	Purpose
<code>create</code>	Create an instance of the Pacific Application Server.
<code>delete</code>	Remove the directory tree and all of the files in an instance.
<code>start</code>	Start an instance of a Pacific Application Server.
<code>stop</code>	Stop a running instance.
<code>config</code>	View, add, update, or delete the property values specified in <code>../conf/appserver.properties</code> .
<code>test</code>	Displays information on the configuration and environment of an instance. It also displays information about error conditions.
<code>instances</code>	Display all the instances created from the Pacific Application Server installed in <code>\$CATALINA_HOME</code> .
<code>unregister</code>	Stop tracking an instance by removing the instance's entry from the <code>\$CATALINA_HOME/conf/instances.[unix windows]</code> file.
<code>register</code>	Register an instance for tracking purposes. (Note that instances are registered for tracking by default when they are created. The register action is only necessary if you explicitly unregistered an instance.)
<code>clean</code>	Truncate, move, or delete the log files located in the <code>/logs</code> directory of either the core server or an instance.
<code>version</code>	Show the Apache Tomcat runtime version and OS information for an instance.

## Installing and running an instance as a Windows service

To install a Pacific Application Server (PAS) instance as a Windows service, you must have administrator privileges. On systems with User Account Control (UAC), you must disable UAC as well.

In addition, the instance must be registered with a core PAS server, which you can accomplish with the `tcman.bat register` action.

A service (called a daemon process on UNIX systems) is an application without a user interface that runs in the background and provides core operating system functionality. Web servers like PAS and Tomcat typically run as Windows services or UNIX daemons.

**Note:** If you run a PAS instance with `tcman.bat start`, the instance runs in the context of the command shell process. It is not available as a system service that can handle external client requests. The instance must be installed as a Windows service before you can run it as a functioning Web server.

This is a summary of how to install and run a PAS instance as a Windows service:

1. Open a command prompt (`cmd.exe`) window.
2. Navigate to the core PAS `/bin` directory (`$CATALINA_HOME/bin`).

3. Run the `service.bat` script using the following syntax:

```
service.bat install instance_name
```

`instance_name` is the name of an existing instance of the core PAS.

---

**Note:** See the *Windows service HOW-TO* help topic in the Apache Tomcat Documentation (<http://tomcat.apache.org>) for more information about installing instances as Windows services.

---

4. Use the Services Microsoft Management Console (MMC) or the `sc config` command to start the service.

---

**Note:** Refer to Windows help for more information on starting, stopping, and other administrative tasks with regard to Windows Services.

---

## Installing and running an instance as a UNIX daemon

A daemon process (called a service on Windows systems) is an application without a user interface that runs in the background and responds to requests. Web servers like PAS and Tomcat typically run as Windows services or UNIX daemons.

---

**Note:** If you run a PAS instance with `tcman.sh start`, the instance runs in the context of the command shell process. It is not available as a system service that can handle external client requests. The instance must be installed as a daemon process before you can run it as a functioning Web server.

---

The file `$CATALINA_HOME/bin/daemon.sh` can be used as a template for starting Tomcat automatically at boot time as a child of the `init` process. For more information, see:

[https://tomcat.apache.org/tomcat-7.0-doc/setup.html#Unix\\_daemon](https://tomcat.apache.org/tomcat-7.0-doc/setup.html#Unix_daemon)

However, you will need to consult with a system administrator before you can configure and run PAS as a daemon process due to differences among UNIX systems and because you need administrative privileges for access to the system.

## Tomcat logging

The core Pacific Application Server (PAS) uses the standard logging technology employed by the Apache Tomcat Web server. For system logging, Tomcat uses an implementation of the Apache Commons Logging library based on the `java.util.logging(JULI)` framework. JULI is implemented in `$CATALINA_HOME/bin/tomcat-juli.jar`.

---

**Note:** This topic only applies to the standard JULI-based logging implemented for the core Tomcat Web server. Web application logging may be based on a different framework (Log4j for example).

---



Log files are written to the `/logs` directory of a running instance (`$CATALINA_BASE/logs`). The default log files for PAS are:

- `catalina.log`, a log with entries that describe server activity
- `catalina.out`, a log for system output and standard error messages
- `localhost.log`, a log file for tracking Web application activity
- `localhost_access.log`, a log for tracking requests processed by the server
- `host_manager.log`, a log for Tomcat's `host-manager.war` Web application
- `manager.log`, a log for Tomcat's `manager.war` Web application

Log files, with the exception of `catalina.out`, are saved daily with the date appended to the filename (`MM-DD-YYYY`). `catalina.out` persists while the server is running. You can delete or archive all of the log files with the TCMAN `clean` action.

The default log files and logging levels are defined in `/conf/logging.properties`. For example, the following entry from `logging.properties` instantiates the `catalina.log` file:

```
1catalina.org.apache.juli.FileHandler.level = INFO
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.
```

Notice that the logging level is set to `INFO` which is the default for all log files in PAS. The following is a list of the JULI logging levels that apply to Tomcat.

**Table 6: JULI logging levels for Tomcat**

Level	Log content
SEVERE	Serious failures
WARNING	Potential problems
INFO	Informational messages
CONFIG	Static configuration messages
FINE	Trace messages
FINER	Detailed trace messages
FINEST	Highly detailed trace messages

The list is arranged in order by level of detail from the least (`SEVERE`) to the greatest (`FINEST`). Note that a level includes all the content of previous levels. For example, `INFO` specifies that the log should include informational messages, potential problems (`WARNING`), and serious failures (`SEVERE`).

## Updating the core Tomcat server in Pacific Application Server

To update the core Tomcat server in Pacific Application Server, install a newer release or service pack of a Progress PAS product. Although a Progress update may not contain the very latest version of Tomcat, you will install the latest version of Tomcat that has been tested for backward compatibility and support of Progress Web applications.

Also, if you attempt to replace the core Tomcat server by downloading from the Apache Web site, will loose the Progress-specific tailoring and customizations that are in the core PAS server.

---

**Note:** Updating the core Tomcat server is easier from an administrative standpoint, when you deploy Web applications to instances of the server rather than to the core Tomcat server itself. Using instances eliminates the need to redeploy Web applications whenever there is a server update. Also, it is easier to back out of a problematic core server update when you deploy your Web applications to instances.

---

## TCMAN Reference

TCMAN is a command-line utility for managing and administering the Pacific Application Server. TCMAN extends the basic Tomcat scripts for starting, stopping, and managing server instances.

This *TCMAN Reference* contains usage information for the `tcmman` command as well as syntax information on all of the TCMAN actions.

## The tcmman command

### Purpose

TCMAN is a command-line utility for managing and administering the Pacific Application Server. On UNIX systems, you run the `tcmman.sh` script followed by appropriate TCMAN actions and options. On Windows systems, you run the `tcmman.bat` batch file, which is identical syntactically and functionally with `tcmman.sh`.

---

**Note:** For the sake of brevity, all the syntax statements and examples in this reference show the `tcmman.sh` script.

---

### Syntax

```
{ $CATALINA_HOME | $CATALINA_BASE } /bin/tcmman.sh action [general_options]
[action_options]
```

## Parameters

`$CATALINA_HOME` | `$CATALINA_BASE`

Specify whether to run TCMAN from the root directory of the installed Pacific Application Server (`$CATALINA_HOME`) or from the root directory of an instance (`$CATALINA_BASE`). The context of where you run TCMAN (whether from the `/bin` directory of the parent, or the `/bin` directory of an instance) affects which server the utility acts on.

---

**Note:** TCMAN automatically determines the value of `CATALINA_BASE` from the directory where you start it. When you run it from the `/bin` directory of an instance, the value of `CATALINA_BASE` is the root directory of the instance. If you run it from the `/bin` directory of the installed Pacific Application Server, the value of `CATALINA_BASE` is the root directory of the installed server (which is the same value as `CATALINA_HOME`).

---

*action*

Specify which TCMAN action to invoke.

*general\_options*

Specify one or more of the TCMAN common options that can apply to most actions. Note that one or more of the general options may be required by a specific action. For example, the `list` action requires `-u` in order to pass a user name and password.

The output of `tcman.sh help action` includes a list of general options that are applicable to a particular action.

The following table is a list of the common options:

**Table 7: TCMAN general options**

Common options	Function
<code>-u user_name:password</code>	Pass a valid user name and a password for HTTP Basic access authentication.
<code>-v</code>	Display verbose output.
<code>-M URL</code>	Override the default manager that manages Web applications by specifying the URL of an alternative manager. <i>URL</i> is expressed in the following format:  <code>{http https}://host:port/manager_application</code>
<code>-B</code>	Override default  <code>CATALINA_BASE</code>  environment settings.

Common options	Function
-n	Debug the TCMAN action but do not execute changes.
-I <i>instance_name</i>	Run TCMAN from the <code>/bin</code> directory of the specified instance.

#### *action\_options*

Specify an option that applies to the selected *action*. These options are explained in the topics that describe each action.

### Example

Run the `help` action from the core server (`/psc/pashome`) to display a list of available TCMAN actions:

```
/psc/pashome/bin/tcman.sh help
usage:  tcman action [options...]
  manager actions:
    list      list deployed applications
    info      list server info
    deploy    deploy application
    undeploy  undeploy application
    reload    reload application
    status    show server status
    leaks     show server memory leaks
    enable    start web application running
    disable   stop running web application
    resources list server global resources
    sessions  list a web application's sessions
  server actions:
    create    create a new server instance
    delete    delete server instance
    config    dump CATALINA_BASE configuration
    clean     clean/archive log files
    instances list tracked server instances
    register  manually register an instance
    unregister manually unregister an instance
    start     start this server
    stop      stop this server
    version   show the server version information
    test      test the server's configuration
  general actions:
    env       show tcman execution environment
    help      show this information
```

## Manager actions

This section details the actions available for deploying, running, and monitoring web applications on a server instance.

## List deployed applications (list)

### Purpose

Display all the web applications that are deployed on an instance.

---

**Note:** This command may be used whether the instance is online or offline. However, the output differs. When used offline, TCMAN simply shows a list of deployed application directories in the instance's web applications directory. When used online, it provides additional run-time details about the deployed web applications.

---

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance if the instance is online. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh list [general_options] [-u user_id:password]
```

### Parameters

*general\_options*

Specify one or more of the options that can be used with any TCMAN action.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

---

**Note:** This option is required if the server is online. It is not required if the server is offline.

---

### Example

Show the Web applications deployed to `acme1` when the instance is online:

```
/psc/acme1/bin/tcman.sh list -u tomcat:tomcat
OK - Listed applications for virtual host localhost
/:running:0:ROOT
/manager:running:4:manager
/oemanager:running:0:oemanager
/oeadapters:running:0:oeabl
```

Show the Web applications deployed to `acme1` when the instance is offline:

```
/psc/acme1/bin/tcman.sh list
OK - Listing directories for /psc/acme1/webapps
/manager:stopped:0:manager
/oeadapters:stopped:0:oeabl
/oemanager:stopped:0:oemanager
/:stopped:0:ROOT
```

## Display OS and server information (info)

### Purpose

Display server and OS information for a running instance.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Use the `test` action to show configuration information about a server that is not running.

### Syntax

```
tcman.sh info [general_options] -u user_name:password
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help info` to see which general options are appropriate.

`-u user_name:password`

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

### Example

Display the OS and server information for the running instance named `acme1`:

```
$: /psc/pashome/tcman.sh info -I acme1 -u tomcat:tomcat
OK - Server info
Tomcat Version: Apache Tomcat/7.0.42
OS Name: Linux
OS Version: 2.6.18-164.el5
OS Architecture: amd64
JVM Version: 1.7.0_02-b13
JVM Vendor: Oracle Corporation
```

## Deploy a Web application (deploy)

### Purpose

Deploy a Web application (`.war` file) to a Pacific Application Server instance whether the server is running (online) or is not running (offline). TCMAN copies the web application to the server's web application directory. If the server is online, you must stop and restart it in order to complete the deployment.

## Syntax

```
tcman.sh deploy [general_options] [-u user_id:password] [-a app_name]  
war_file_path
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help deploy` to see which general options are appropriate.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication.

---

**Note:** This option is required if the server is online. It is not required if the server is offline.

---

`-a app_name`

Specify a name for the web application. If you do not use this option, the application name will be the same as the `.war` file name.

*war\_file\_path*

Specify the location of the web application `.war` file that you want to deploy.

## Example

Deploy and rename `oeabl.war` (a web application that implements OpenEdge adapters) to the `acme1` instance of the core `pashome` server:

```
/psc/acme1/bin/tcman.sh deploy -a oeadapters /psc/pashome/extras/oeabl.war  
OK - deployed /psc/pashome/extras/oeabl.war to local directory  
/psc/acme1/webapps
```

---

**Note:** The `$CATALINA_HOME/extras` directory (`/psc/pashome/extras` in the example above) also contains number of instance management applications, including `host-manager.war`, `manager.war`, and `oemanager.war`.

---

## Undeploy a Web application (undeploy)

### Purpose

Remove a Web application from running (online) or stopped (offline) instances. If the instance's autodeploy option is off, you must stop and restart a running server to complete removal. Note that the autodeploy option is set in the `.../conf/appserver.properties` file and is off by default.

## Syntax

```
tcman.sh undeploy [general_options] [-u user_id:password] app_name
```

## Parameters

*general\_options*

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help undeploy` to see which general options are appropriate.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.) This option is required if you are accessing an online instance.

*app\_name*

Specify the name of the web application to remove.

## Example

Remove the oemanager application from the `acme1` instance:

```
/psc/acme1/bin/tcman.sh undeploy -u tomcat:tomcat oemanager
OK - Undeployed application at context path /oemanager
```

## Reload a Web application (reload)

### Purpose

Restart a deployed, running Web application so that the application can pick up changes to its classes or libraries.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

---

**Note:** The reload action does not reload the web application's `web.xml` file. To begin using changes to `web.xml`, you must stop and restart the web application.

---

## Syntax

```
tcman.sh reload [general_options] -u user_id:password app_name
```



## Parameters

*general\_options*

Specify one or more of the options that can be used with any TCMAN action. Run `tcmn.sh help reload` to see which general options are appropriate.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

---

**Note:** This option is required if the server is online. It is not required if the server is offline.

---

*app\_name*

Specify the name of the web application to restart.

## Example

Reload the `oemanager` web application running on the `acme1` instance:

```
/psc/acme1/bin tcmn.sh reload -u tomcat:tomcat oemanager
OK - Reloaded application at context path /oemanager
```

## Display detailed server status (status)

### Purpose

List information from the core server's memory, including web application statistics. Information includes memory pool usage, connector thread status, and connector status. Output is in XML format. (Note that redirecting the output to an XML viewer makes it more readable.)

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcmn.sh status [general_options] -u user_name:password [-f]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcmn.sh help status` to see which general options are appropriate.

`-u user_name:password`

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

-f

Return full status information.

## Example

Display core server's memory and web application statistics and use `xmllint` to format for readability:

```
$: tcman.sh status -u tomcat:tomcat | xmllint --format -
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="/manager/xform.xsl" ?>
<status>
  <jvm>
    <memory free="453196832" total="520028160" max="1051394048"/>
    <memorypool name="PS Eden Space" type="Heap memory" usageInit="50331648"
usageCommitted="48758784" usageMax="55967744" usageUsed="1525560"/>
    <memorypool name="PS Old Gen" type="Heap memory" usageInit="469762048"
usageCommitted="469762048" usageMax="1006632960" usageUsed="63861584"/>
    <memorypool name="PS Survivor Space" type="Heap memory" usageInit="8388608"
usageCommitted="1507328" usageMax="1507328" usageUsed="1444184"/>
    <memorypool name="Code Cache" type="Non-heap memory" usageInit="2555904"
usageCommitted="3407872" usageMax="50331648" usageUsed="3303104"/>
    <memorypool name="PS Perm Gen" type="Non-heap memory" usageInit="67108864"
usageCommitted="67108864" usageMax="134217728" usageUsed="47406400"/>
  </jvm>
  <connector name="&quot;http-bio-8601&quot;">
    <threadInfo maxThreads="150" currentThreadCount="0"
currentThreadsBusy="0"/>
    <requestInfo maxTime="0" processingTime="0" requestCount="0" errorCount="0"
bytesReceived="0" bytesSent="0"/>
    <workers/>
  </connector>
  <connector name="&quot;http-bio-8501&quot;">
    <threadInfo maxThreads="300" currentThreadCount="10"
currentThreadsBusy="1"/>
    <requestInfo maxTime="2008" processingTime="2116" requestCount="10"
errorCount="0" bytesReceived="0" bytesSent="5838"/>
    <workers>
      <worker stage="S" requestProcessingTime="2" requestBytesSent="0"
requestBytesReceived="0" remoteAddr="127.0.0.1" virtualHost="localhost"
method="GET" currentUri="/manager/status" currentQueryString="XML=true"
protocol="HTTP/1.1"/>
    </workers>
  </connector>
</status>
```

## Display memory leaks (leaks)

### Purpose

List Web applications with potential memory leaks.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh leaks [general_options] -u user_name:password
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help leaks` to see which general options are appropriate.

`-u user_name:password`

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

## Example

Display memory leaks for web applications deployed on the `acme1` server instance:

```
/psc/acme1/bin/tcman.sh leaks -u tomcat:tomcat
OK - Found potential memory leaks in the following applications:
/warehouse
```

## Start a Web application (enable)

### Purpose

Start a web application that is deployed but not running.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh enable [general_options] -u user_id:password app_name
```

## Parameters

*general\_options*

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help start` to see which general options are appropriate.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

*app\_name*

Specify the name of the web application to start.

---

**Note:** To start the ROOT web application, you can specify `/` or `ROOT`.

---

## Example

Start the `oeabl` application deployed on the `acme1` instance:

```
tcman.sh enable -u tomcat:tomcat oeabl
OK - Started application at context path /oeabl
```

## Stop a Web application (disable)

### Purpose

Stop a running Web application.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh disable [general_options] [-u user_id:password] app_name
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help disable` to see which general options are appropriate.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

*app\_name*

Specify the name of the web application to disable.

---

**Note:** To disable the ROOT web application, you can specify `/` or `ROOT`.

---

### Example title

Disable the `oeabl` application running on the `acme1` instance:

```
/psc/acme1/bin/tcman.sh disable -u tomcat:tomcat oeabl
OK - Stopped application at context path /oeabl
```

## Display global server resources (resources)

### Purpose

List the global resources used by the core server.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh resources [general_options] -u user_name:password
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help resources` to see which general options are appropriate.

`-u user_name:password`

Pass a valid user name and a password for HTTP Basic access authentication.  
(The default is `-u tomcat:tomcat`.)

### Example

Display global resources for the running instance, `acme1`:

```
$: /psc/acme1/bin/tcman.sh resources -u tomcat:tomcat
OK - Listed global resources of all types
ServiceRegistry/ServiceRegistryFactory:com.progress.appserv.services.naming.ServiceRegistry
UserDatabase:org.apache.catalina.users.MemoryUserDatabase
```

## Display Web application HTTP sessions (sessions)

### Purpose

Display how many sessions are active for the specified Web application, categorized by their duration.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

### Syntax

```
tcman.sh sessions [general_options] -u user_id:password app_name
```

## Parameters

*general\_options*

Specify one or more of the options that can be used with any TCMAN action.

`-u user_id:password`

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

*app\_name*

Specify the name of the web application to analyze for session information.

## Example

Show the active sessions for the `manager` application deployed on the `acme1` instance:

```
/psc/acme1/bin/tcman.sh sessions -u tomcat:tomcat manager
OK - Session information for application at context path /manager
Default maximum session inactive interval 30 minutes
<1 minutes: 1 sessions
8 - <9 minutes: 2 sessions
9 - <10 minutes: 1 sessions
```

# Server actions

This section details the actions available for creating and monitoring server instances.

## Create an instance (create)

### Purpose

Create a new instance of the core Pacific Application Server server by running this action from `/bin` directory of the core server ( `$CATALINA_HOME/bin/tcman.sh create`).

### Syntax

```
tcman.sh create [general_options] [-f] [-p port_num] [-P port_num]
               [-s port_num] [-j port_num] [-W pathname] [-N instance_name]
               [-U user_id] [-G group_id] base_path
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help create` to see which general options are appropriate.

`-f`

Copy all deployed web application archives (`.war` files) from `$CATALINA_HOME` to the new instance.

`-p port_num`

Specify the TCP port that listens for HTTP messages. The default is 8080.

`-P port_num`

Specify the TCP port that listens for HTTPS messages. The default is 8443.

`-s port_num`

Specify the TCP port to use to stop an instance. On Windows systems, you must specify a shutdown port. On UNIX, shutdown ports are optional.

`-j port_num`

Specify the TCP port that listens for AJP13 messages (an Apache protocol for handling requests from a web server to an application server). The default is 8009.

`-W pathname`

Specify the directory where web applications will be deployed. The default is `$CATALINA_BASE/webapps`.

`-N instance_name`

Specify a name for the instance. The default is the name of the directory where the instance is created.

All instances are automatically registered for tracking when they are created. If you intend to track an instance, the instance name cannot contain spaces or any of the following characters: "[ . # | ] \$ ? + = { / , }"

`-U user_id`

Specify the user-id of the owner of all the files and directories of the instance. The default is the user-id of the current process. `-G` must be specified if you use this option.

`-G group_id`

Specify the group-id of the owner of all the files and directories of the instance. The default is the group-id of the current process. `-U` must be specified if you use this option.

`base_path`

Specify the pathname where you will create the instance.

## Example

Create an instance of `/psc/pashome` in `/psc/acme1`:

```
$: /psc/pashome/bin/tcman.sh create -p 8501 -P 8601 -s 8701 /psc/acme1
Server instance acme1 created at /psc/acme1
```

## Delete an instance (delete)

### Purpose

Remove the directory tree and all of the files in an instance. Alias tracking is disabled for servers that are removed.

To execute this action, the instance cannot be running.

---

**Note:** You cannot recover any files or directories removed by the `delete` action. Backup anything you want to save before launching this action.

Also note that you cannot use `delete` to remove the installed, root server ( `$CATALINA_HOME` ).

---

### Syntax

```
tcman.sh delete [general_options] [-y] [base_path | alias_name]
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help delete` to see which general options are appropriate.

`-y`

Delete everything without prompting for confirmation.

*base\_path*

Specify the pathname of the instance that you intend to delete.

*alias\_name*

Refer to the instance that you intend to delete by its alias rather than its pathname.



## Example

Delete the instance of pashome that was created in /psc/acme3:

```
$: /psc/pashome/bin/tcman.sh delete /psc/acme3
The following directory tree will be removed permanently:
( WARNING all deployed web applications will be DELETED!! )
/PAS/wrkdir/acme3
/PAS/wrkdir/acme3/conf
/PAS/wrkdir/acme3/temp
/PAS/wrkdir/acme3/common
/PAS/wrkdir/acme3/common/lib
/PAS/wrkdir/acme3/logs
/PAS/wrkdir/acme3/webapps
/PAS/wrkdir/acme3/webapps/ROOT
/PAS/wrkdir/acme3/webapps/ROOT/static
/PAS/wrkdir/acme3/webapps/ROOT/static/error
/PAS/wrkdir/acme3/webapps/ROOT/static/auth
/PAS/wrkdir/acme3/webapps/ROOT/META-INF
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters/rest/PingService
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters/soap
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com/progress
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com/progress/appserv
/PAS/wrkdir/acme3/work
/PAS/wrkdir/acme3/bin
Type 'yes' to continue
yes
Delete operation complete
server removed at /PAS/wrkdir/acme3
```

## Display and manage an instance's configuration (config)

### Purpose

View, add, update, or delete the property values specified in `../conf/appserver.properties`.

When you run `tcman.sh config` with no parameters, it displays the core Tomcat server's configuration, and all the properties in both `../conf/appserver.properties` and `../conf/jvm.properties`. Note, however, that you can only view `jvm.properties`. You cannot modify its contents with the `config` action.

### Syntax

```
tcman.sh config [general_options]
[prop_name | prop_name=value | +prop_name=value | ~prop_name]
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help config` to see which general options are appropriate.

*prop\_name*

Display the specified property and its value.

*prop\_name=value*

Set the value of a property that exists in `.../conf/appserver.properties`.

*+prop\_name=value*

Add a new property to `.../conf/appserver.properties` and set its value.

*~prop\_name*

Remove the specified property from `.../conf/appserver.properties`.

## Examples

Show the configuration and properties of `acme1`, an instance of the core server, `pashome`:

```
$: /psc/acme1/bin/tcman.sh config
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
Server version: Apache Tomcat/7.0.42
Server built:   Jul 2 2013 08:57:41
Server number:  7.0.42.0
OS Name:        Linux
OS Version:     2.6.18-164.el5
Architecture:   amd64
JVM Version:    1.7.0_02-b13
...
```

Display the value of a single property:

```
$: /psc/acme1/bin/tcman.sh config psc.as.http.port
psc.as.http.port=8501
```

Update the value of a property that exists in the `appserver.properties` file and then check the value:

```
$: /psc/acme1/bin/tcman.sh config psc.as.http.port=6543
$: tcman.sh config psc.as.http.port
psc.as.http.port=6543
```

Add a new property/value pair to the `appserver.properties` file and then check the value:

```
$: /psc/acme1/bin/tcman.sh config +my.home.dir=/home/jarhead
$: tcman.sh config my.home.dir
my.home.dir=/home/jarhead
```

Remove a property/value pair from the `appserver.properties` file and check if deletion was successful:

```
$: /psc/acmel/bin/tcman.sh config ~my.home.dir
$: tcman.sh config my.home.dir
Property does not exist - my.home.dir
```

---

**Caution:** There are no restrictions to property removal. You can render the server unable to start if you remove a property required by `conf/server.xml`.

---

## Notes

- All property names are case sensitive.
- You cannot enter multiple property names (*prop\_name*) on the command line to view, update, or add properties to the `appserver.properties` file.
- You cannot use the `config` action to update existing values or add new values to the `jvm.properties` file

## Display or modify the server features of an instance (feature)

### Purpose

View, enable, or disable the server features contained in the `/conf/server.xml` file of an instance.

When you run `tcman.sh feature` with no parameters, it displays a list of the features (and their current status) that you can enable or disable. You can also display the status of a single server feature. After viewing the status of a feature, you can use `tcman.sh feature` to change its setting.

### Syntax

```
tcman.sh feature [general_options] [feature_name [= {on | off}]]
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help feature` to see which general options are appropriate.

*feature\_name*

Specify one of the features defined in an instance's `conf/server.xml` file. Running `tcman.sh feature` without *feature\_name* displays a list of all the features.

on

Enables the named feature.

off

Disables the named feature.

## Example

Display the list of server feature settings for `acme1`, enable AJP13 (Apache JServ Protocol version 1.3), and verify that the feature is enabled:

```
$: /psc/acme1/bin/tcman.sh feature
SecurityListener=off
JMXLifecycle=off
PSCRegistry=on
HTTP=onHTTPS=on
AJP13=off
Cluster=off
UserDatabase=on
JAASRealm=off
LDAPRealm=off
PASInstrument=off
RemoteHostValve=on
RemoteAddrValve=onSingleSignOn=on
AccessLog=on
CrawlerSessionManager=on
StuckSessionValve=on

$: /psc/acme1/bin/tcman.sh feature AJP13=on

$: /psc/acme1/bin/tcman.sh feature AJP13
AJP13=on
```

## Notes

- Server features for instances are set in `$CATALINA_BASE/conf/server.xml`. You can change feature status by manually editing this file. However, it is safer to use `tcman.sh feature` to avoid corrupting the file with erroneous entries.
- Run `tcman.sh feature` when the instance is offline.

## Clean up or archive a server's log files (clean)

### Purpose

Truncate, move, or delete the log files located in the `/logs` directory of the core server or instance. If the server is running, `clean` truncates log files to zero length. If the server is not running, `clean` deletes the log files from the file system.

You have the option to save log files to a subdirectory of `/logs`.

## Syntax

```
tcman.sh clean [general_options] [-A]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help clean` to see which general options are appropriate.

-A

Archive log files to a subdirectory of `$CATALINA_BASE/logs`. The directory is automatically named with a month-day-year-second (*MM-DD-YYYY-ss*) time-stamp format. If the server is not running, the files in `$CATALINA_BASE/logs` are deleted.

## Example

Archive the log files of `acme1`, an instance of the core server `pashome`, and save to a file:

```
/psc/pashome/tcman.sh clean -I acme1 -A
```

# Display a server's instances (instances)

## Purpose

Display all the instances created from the Pacific Application Server installed in `$CATALINA_HOME`.

---

### Note:

The server does not keep track of its instances by default. Before you can track instances, you must create a `$CATALINA_HOME/conf/instances.unix` or a `$CATALINA_HOME/conf/instances.windows` text file. (Use the file name extension corresponding to the OS you are running.) The `instances` action displays the content of the file.

If the `instances` file exists, the TCMAN utility adds instance entries to the file when you execute a `create` action. TCMAN removes instance entries when you execute the `delete` action.

You can manually add or remove instance entries from the `instances` file by using the `register` or `unregister` actions.

---

## Syntax

```
tcman.sh instances [general_options]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help instances` to see which general options are appropriate.

## Example

Display the instances of the core server installed in `/psc/pashome`:

```
/psc/pashome/bin/tcman.sh instances
acme1 : /psc/acme1 : instance : ok
acme2 : /psc/acme2 : instance : ok
```

## Register an instance for tracking (register)

### Purpose

Register an instance for tracking purposes.

---

#### Note:

Instances are automatically registered for tracking when you execute a `create` action. You use the `register` action to restart tracking on instances after tracking was stopped.

A typical use for unregistering and then re-registering an instance is to make configuration changes when moving instances from one location (core server) to another. The `register` action enables tracking and also updates the value of `CATALINA_HOME` in all of the executable scripts in the instance's `/bin` directory to refer to the new core server.

---

### Syntax

```
tcman.sh register alias_name instance_path
```

### Parameters

*alias\_name*

Specify a meaningful name for the instance. The alias name must be unique in the instances file.

*instance\_path*

Specify the OS file system path to where the instance exists. This value will be expanded into a fully qualified OS directory path and will be verified to exist.

## Example

Track `test1`, which is an alias for the instance `/psc/acme1`:

```
/psc/pashome/bin/tcman.sh register test1 /psc/acme1
```

## Notes

When you register an instance for tracking or create a new instance with the `create` command, an entry is created in the core Pacific Application Server's

`$CATALINA_HOME/conf/instances.[unix|windows]` file.

The `instances.[unix|windows]` file is a simple text file, which can be manually edited (with care) in the event that it becomes out of date. The format for entries is:

```
instance_name = base_path
```

An `instances.unix` file uses Unix OS file path syntax (forward slashes), and an `instances.windows` file uses Windows OS file path syntax (backslashes) to specify *base\_path*.

Also note that in an `instances` file:

- Any line starting with a pound-sign ( `#` ) is a comment line.
- Blank lines are skipped.
- Instance names cannot contain spaces or any of the following characters: " [ . # | ] \$ ? + = { / , } "

## Stop tracking an instance (unregister)

### Purpose

Stop tracking an instance by removing the instance's entry from the `$CATALINA_HOME/conf/instances.[unix|windows]` file.

---

### Note:

You use the `register` action to restart tracking on instances after tracking was stopped with `unregister`.

A typical use for unregistering and then re-registering an instance, is to make configuration changes when moving instances from one location, or core server, to another. The `register` action not only enables tracking, it also updates the value of `CATALINA_HOME` in all of the executable scripts in the instance's `/bin` directory to refer to the new core server.

---

### Syntax

```
tcman.sh unregister alias_name
```

## Parameters

*alias\_name*

Specify the alias name of the instance that you want to stop tracking. The alias name must exist in an `instances.[unix|windows]` file.

## Example

Stop tracking `test1`, which is an instance of `/psc/pashome`:

```
/psc/pashome/bin/tcman.sh unregister test1
```

## Start an instance (start)

### Purpose

Start an instance of a Pacific Application Server.

### Syntax

```
tcman.sh start [general_options] [-D | -J]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help start` to see which general options are appropriate.

`-D`

Start the server in Tomcat debug mode. `-D` overrides the `-J` option.

`-J`

Start the server in debug mode using the JDPA (Java Platform Debugger Architecture) APIs for debugging. `-J` cannot be used if the `-D` option is specified.

Before you run a server with the `-J` option, you must define a port for the JDPA debugger by setting the `JDPA_ADDRESS` environment variable to a unique TCP network port number.



## Example

Start the server in `/psc/acme1`, which is an instance of the core server in `/psc/pashome`:

```
/psc/acme1/bin/tcman.sh start
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
```

## Stop an instance (stop)

### Purpose

Stop a running instance.

### Syntax

```
tcman.sh stop [general_options] [-F [-w seconds]]
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help stop` to see which general options are appropriate.

`-F`

Kill the sever process if it does not stop 5 seconds. Change the default 5 second interval by using the `-w` option.

`-w seconds`

Optionally specify the number of seconds to wait before killing a server process.

### Example

Stop the server in `/psc/acme1`, which is an instance of the core server in `/psc/pashome`:

```
/psc/acme1/bin/tcman.sh stop
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
```

## Display server, OS, and runtime version information (version)

### Purpose

Show the Apache Tomcat runtime version and OS information for an instance.

To execute this action, the instance cannot be running

### Syntax

```
tcman.sh version [general_options]
```

### Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help version` to see which general options are appropriate.

### Example

Display the server and runtime information for `acme1`, an instance of the core server installed in `/psc/pashome`:

```
$: /psc/pashome/bin/tcman.sh version -I acme1
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/users/doc/agarbacz/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
Server version: Apache Tomcat/7.0.42
Server built:   Jul 2 2013 08:57:41
Server number:  7.0.42.0
OS Name:        Linux
OS Version:     2.6.18-164.el5
Architecture:   amd64
JVM Version:    1.7.0_02-b13
JVM Vendor:     Oracle Corporation
```

## Test a server configuration (test)

### Purpose

Displays information on the configuration and environment of an instance. It also displays information about error conditions.

The `test` action starts a server (instance), loads all the configuration files, and then displays information. The instance is stopped, exiting gracefully even if there is an error condition.

To execute this action, the instance cannot be running

## Syntax

```
tcman.sh test [general_options]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help test` to see which general options are appropriate.

## Example

Run a test of the configuration of `acme1`, which is an instance of the core server installed at `/psc/pashome`:

```
$: /psc/pashome/bin/tcman.sh -I acme1 test
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
. . .
```

## Notes

The `test` action is particularly useful for testing to verify that a server will start and run properly after you make changes to configuration and properties files.

# General actions

This section details the actions available for displaying help and server runtime environment information.

## Display help (help)

### Purpose

Display summary or detailed help for all TCMAN actions, property names, and server features.

### Syntax

```
tcman.sh help [action | property | feature]
```

## Parameters

*action*

Show the syntax and options of the specified action. If no action is specified, show a list of all actions and the general options.

*property*

Show the settings for specified property.

*feature*

Show if the specified feature is enabled or disabled.

## Example

Display the usage help for the `create` action:

```
$: tcman.sh help create
usage: tcman create [options] -p <http-port> [instance-opts] <new-base-path>

instance-opts:
  [-s <shutdown-port>]
  [-P <https-port>]
  [-j <ajp13-port>]
  [-W <web-apps-dir>]
  [-N <inst-alias-name>]
  [-U <file-owner> -G <file-group>]

general options:
  -u uid:pwd  pass uid and pwd for HTTP BASIC authentication
  -v          print verbose output
  -M url      override the CATALINA_BASE manager's URL with
              <{http|https}://<host>:<port>/<mgr-app>
  -B          override CATALINA_BASE environment setting
  -n          debug run action but do not execute changes
```

## Display runtime environment information (env)

### Purpose

Show details about a server's state.

### Syntax

```
tcman.sh env [general_options] [keyword]
```

## Parameters

*general\_options*

Specify one or more of the general TCMAN options. Run `tcman.sh help env` to see which general options are appropriate.

*keyword*

Specify one or more keywords that represent the name of the state that you want to view. If no keyword is specified, then all of the state information is displayed.

Keywords include:

Keyword	Description
running	Indicate if a server is running ( 1 ) or not running ( 0 ).
mgrurl	Display the URL of the manager application.
type	Display the server type.
alias	Display the server's alias.
parent	Display the pathname of the parent of an instance.
tracking	Indicate if tracking is on (1) or off ( 0 ).
http	Display the server's http port number.
https	Display the server's https port number.
shut	Display the server's shutdown port number. A value of -1 indicates that there is no shutdown port.
pid	Display the server's process id. A hyphen ( - ) indicates that the server is not running.

**Example**

Display all of the state information for the instance created in `/psc/acme1`:

```
/psc/acme1/bin/tscman.sh env
catalina home:      /psc/pashome
catalina base:      /psc/acme1
java home:          /tools/linuxx86_64/java64/jdk1.7.0_02/
jre home:
manager http port:  8501
manager https port: 8601
manager shut port:  8701
manager URL:        http://localhost:8501/manager
config type:        instance
config alias:       acme1
config parent:      /psc/pashome
server running:     0
instance tracking:  1
instance file:      /psc/pashome/conf/instances.unix
server process-id:  -
```

