PROGRESS Corticon

Corticon Server: Deploying Web Services with .NET

PROGRESS

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 Easyl, DataDirect Spy, DataDirect SupportLink, Easyl, 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

Preface.....7

Progress Corticon documentation.....7

Overview of Progress Corticon.....9

Chapter 1: Conceptual overview.....11

What is a web service?.....11

What is a decision service?.....12

What is the Corticon Server for .NET?.....12

What is a web services consumer?.....12

Chapter 2: Getting started with Corticon Server for .NET.....13

Chapter 3: Installing Corticon Server for .NET.....15

Downloading the Corticon Server for .NET package.....15

System requirements.....16

Corticon Server for .NET setup wizard.....16

Chapter 4: Setting up Windows Internet Information Services.....21

Windows Server 2008 or Windows 7 (IIS 7.5).....22

Windows Server 2012, R1 and R2 (IIS 8.0 and 8.5).....37

Chapter 5: Testing the configuration.....43

Chapter 6: Updating your Corticon license JAR for .NET.....45

Chapter 7: Corticon .NET Server files and API tools.....47

Setting up Corticon .NET Server use cases.....48

Installing Corticon .NET Server as an in-process server.....48

The Corticon .NET Server home and work directories.....49

.NET Configurations that use global environment settings.....49

The Corticon .NET Server Sandbox.....50

Testing the installed Corticon .NET Server.....50

Testing as in-process.....50

Testing a remote server on IIS.....52

Chapter 8: Deploying a Ruleflow to the Corticon Server for .NET.....55
Creating a Ruleflow.....56
Creating and installing a Deployment Descriptor file.....56
Using the Deployment Console tool's Decision Services on the .NET Server.....56
Installing the Deployment Descriptor file.....59
Hot re-deploying deployment descriptor files and ruleflows.....60

Chapter 9: Consuming a Decision Service.....61
Integrating and testing a Decision Service.....62
Path 1- Using Corticon Studio as a SOAP client to consume a Decision Service.....62
Configuring Studio to send a SOAP Message to IIS.....63
Creating a new .NET server test in Corticon Studio 63
Executing the remote test.....64
Path 2 - Using bundled C# sample code to consume a Decision Service.....65
Creating the WSDL and proxy files.....65
Path 3 - Using SOAP client to consume a Decision Service.....67
Web services messaging styles.....67
Creating a service contract using the Deployment Console.....68
Creating a request message for a decision service.....68
Sending a request message to the server.....69
Limits of the default evaluation license.....69
Troubleshooting.....70

Chapter 10: Using .NET Business Objects as payload for Decision Services.....71

Chapter 11: Support for Windows Communication Framework (WCF)...83
Creating WSDL and proxy files.....83

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	
Corticon Studio Tutorial: Basic Rule Modeling	Introduces modeling, analyzing, and testing rules and decisions in Corticon Studio. Recommended for evaluators and users getting started. *See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's **Help** menu.*
Corticon Studio Tutorial: Advanced Rule Modeling	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. *See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's **Help** menu.*

Corticon Tutorial: Using Enterprise Data Connector (EDC)	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.
Corticon Studio Documentation: Defining and Modeling Business Rules	
Corticon Studio: Installation Guide	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.
Corticon Studio: Rule Modeling Guide	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 *Test Yourself* exercises.
Corticon Studio: Quick Reference Guide	Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.
Corticon Studio: Rule Language Guide	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.
Corticon Studio: Extensions Guide	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.
Corticon Server Documentation: Deploying Rules as Decision Services	
Corticon Server: Deploying Web Services with Java	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 SOAP and JSON/RESTful samples. Presents the features and functions of the browser-based Server Console. Provides administrative instructions for the Pacific Application Server.

Corticon Server: Deploying Web Services with .NET	Details installing the Corticon Server as a Web Services Server, and then deploying and exposing decisions as Web Services with .NET. Provides installation and configuration information for the .NET Framework and Internet Information Services (IIS) on various supported Windows platforms.
Corticon Server: Integration & Deployment Guide	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. 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.

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.

1

Conceptual overview

This guide is a walkthrough of fundamental concepts and functions of Corticon Server for .NET. The examples focus on Microsoft Internet Information services (IIS) as the web server to provide .NET Web Services.

For details, see the following topics:

- What is a web service?
- What is a decision service?
- What is the Corticon Server for .NET?
- 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.]

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 component or .NET library). 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 for .NET, it becomes a Decision Service.

What is the Corticon Server for .NET?

The Corticon Server for .NET 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 for .NET 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.

2

Getting started with Corticon Server for .NET

This tutorial steps through the procedure necessary for running the Corticon Server for .NET as a web services server, deploying Ruleflows to the Server, exposing the Ruleflows as Decision Services and 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 *Server Integration & Deployment Guide*.

3

Installing Corticon Server for .NET

For details, see the following topics:

- Downloading the Corticon Server for .NET package
- System requirements
- Corticon Server for .NET setup wizard

Downloading the Corticon Server for .NET package

Corticon Server for .NET is packaged in an executable installer application:

- `PROGRESS_CORTICON_5.4_SERVER.NET.exe`

To download the installer:

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 installer to a temporary location accessible by the target machine.

System requirements

Corticon Server for .NET requirements include:

- All Java-supported processors
- 512 MB RAM recommended
- 10 MB disk space (core libraries)
- 150 MB disk space (full install)

Progress Corticon Server for .NET is supported on several Microsoft Windows platforms. See the Progress Software web page Progress Corticon 5.4 - Supported Platforms Matrix for more information.

Corticon Server for .NET setup wizard

You can install Corticon Server .NET 5.4 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.

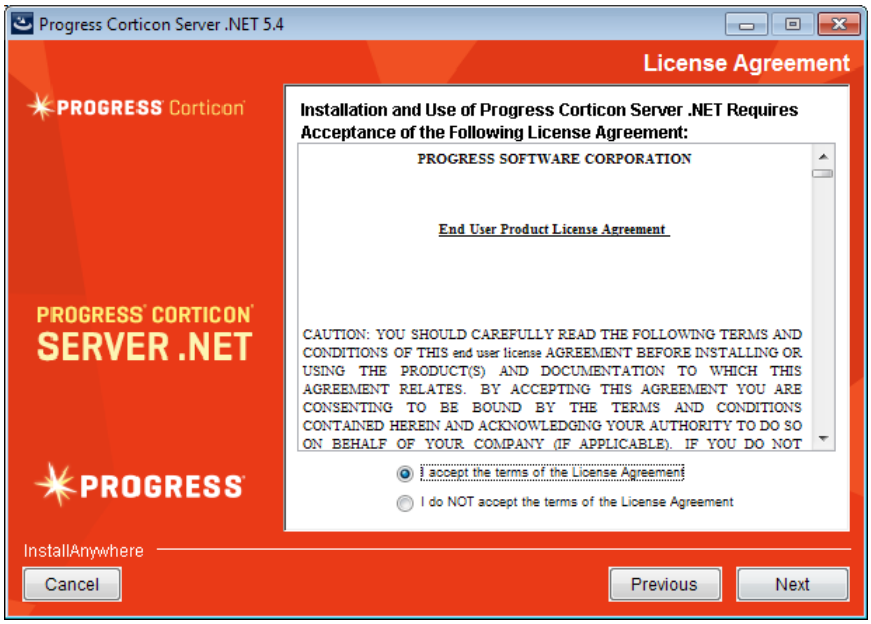
Note: An installed version of Corticon Server .NET 5.2 or earlier on the target machine must be uninstalled, then Corticon Server .NET 5.4 installed. Consult with Progress Corticon Support or your Progress representative to consider your migration strategies for existing assets before you take action.

Installation procedure

To create an installation of Corticon Server 5.4 for .NET:

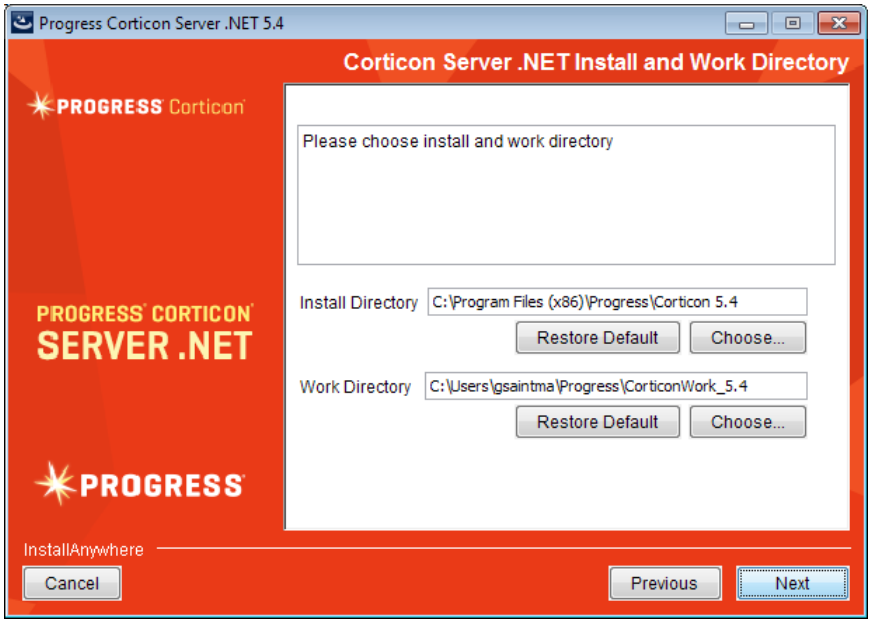
1. Double click on `PROGRESS_CORTICON_5.4_SERVER.NET.exe` to open the Progress Corticon Server .NET Setup Wizard.
2. Click **Next** to continue.

   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**.

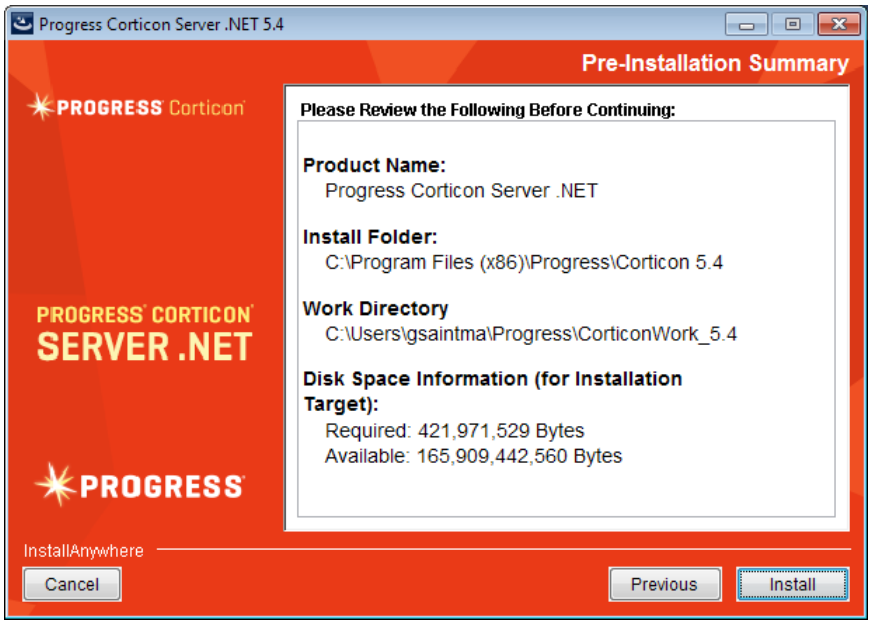
   The **Corticon Server .NET Install and Work Directory** panel opens.



4. Select the installation and work directories. To accept the default locations, click **Next**.

   To specify preferred locations, click **Choose**, navigate to each preferred directory, click **OK** to close the chooser dialog, and then click **Next**.

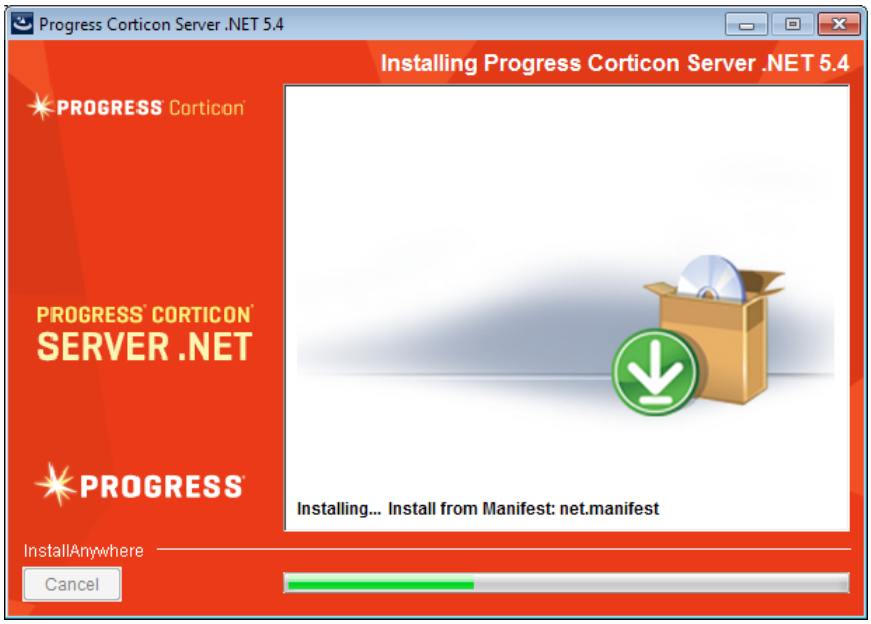
   The **Pre-Installation Summary** page opens.



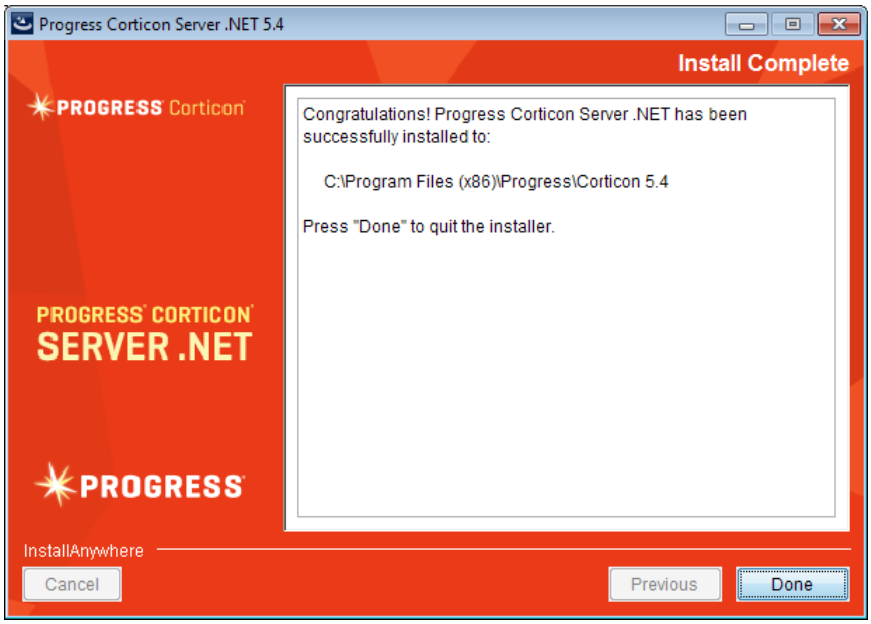
Verify your selections in the **Pre-Installation Summary** panel.

5. 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.



6. Choose **Done** to complete the Corticon Server for .NET installation and close the installer.

Service Pack updater

When a Corticon 5.4 service pack release is announced, you can download its installer and apply it to a 5.4 installation that does not already have that SP or an earlier one applied. The *What's New* guide for the service pack will describe the service pack's features as well as procedures for applying it, and then updating the IIS location.

Uninstalling Corticon Server .NET

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

To uninstall Corticon Server for .NET:

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

4

Setting up Windows Internet Information Services

The following procedures outline the general instructions for setting up a Microsoft Windows installation to use Internet Information Services (IIS). Consult your specific operating system guide and your system administrator for any special considerations or requirements in your deployment topology. Internet Information Services is usually pre-installed but not turned on. If IIS is not installed, you might need to access download sites or appropriate operating system media.

Note: Platforms and versions

The procedures in this section demonstrate set up of IIS for Corticon Server .NET on supported Windows platforms. Consult your specific operating system guide and your system administrator for any special considerations or requirements in your deployment topology. As IIS and Windows are third-party products, this information might change without notice.

You can refer to Microsoft information resources or interpolate from these procedures to set up IIS on other supported Windows OS version and IIS version combinations. Contact Progress Corticon support for assistance.

For details, see the following topics:

- Windows Server 2008 or Windows 7 (IIS 7.5)
- Windows Server 2012, R1 and R2 (IIS 8.0 and 8.5)

Windows Server 2008 or Windows 7 (IIS 7.5)

The procedures in this topic apply to Windows Server 2008 and Windows 7 (IIS 7.5).

Required .NET Framework

Microsoft .NET Framework 4.0 and 4.5 are the supported version for this release of Corticon. If you download Microsoft .NET Framework 4.5, it is applied as an update to the 4.0 location.

Check to see if the target machine already has a v4.0.X folder at `C:\Windows\Microsoft.NET\Framework64\` or `C:\Windows\Microsoft.NET\Framework\`.

Note: If you have the Framework Client Profile, check to see that the file `aspnet_regiis.exe` is listed. You might need to download and install the **Microsoft .NET Framework 4 Extended** upgrade to make all the required files available.

As needed, follow these steps to install it before you start setting up IIS:

1. Browse to `http://www.microsoft.com/downloads`.
2. Search for `.NET Framework 4 download`
3. Choose the appropriate installer for your operating system.
4. Download and save the installer to your hard drive. You might have to add the download URL to your list of trusted sites.
5. Double-click the installer executable and follow its instructions.

Once you have completed and verified the .NET Framework installation and configuration, proceed to set up IIS.

Setting up IIS on Windows 2008 or Windows 7

The following procedures outline the general instructions for setting up a Microsoft Windows Server 2008 or Windows 7 installation to use Internet Information Services (IIS).

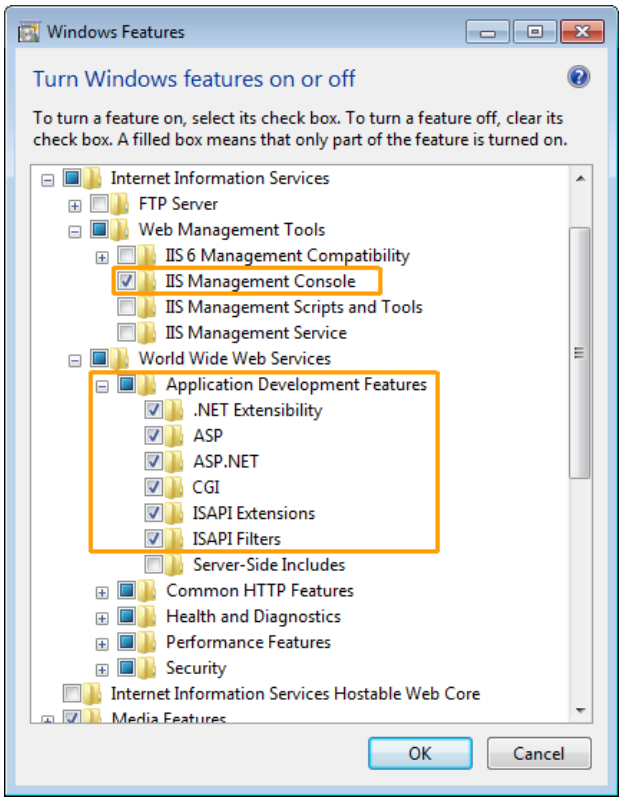
Turning on the Windows IIS feature in Windows 2008 or Windows 7

Windows 2008 and Windows 7 installations typically include the Internet Information Server (IIS) but require user interaction to enable it.

Note: If IIS is not installed, you might need to access download sites or appropriate operating system media.

To enable IIS on a host machine for Corticon Server for .NET:

1. Choose **Start > Control Panel**. Choose **Programs and Features**, and then select **Turn Windows feature on or off**, as shown:
2. Select the following features:



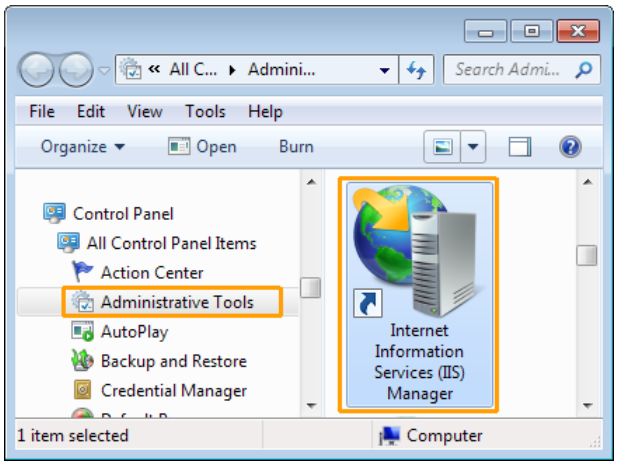
- Internet Information Services -> Web Management Tools -> IIS Management Console
- Internet Information Services -> World Wide Web Services -> Application Development Features:
  - .NET Extensibility
  - ASP
  - ASP .NET
  - CGI
  - ISAPI Extensions
  - ISAPI Filters

3. Click **OK**. Windows installs the selected components.

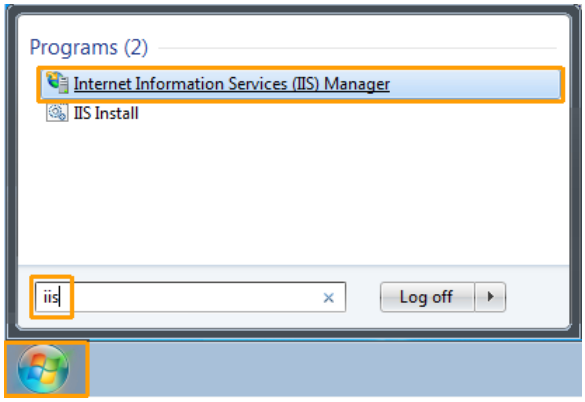
IIS is now enabled.

Launching the IIS Manager on Windows 2008 or Windows 7

1. To start the IIS Manager, either:
   - Choose **Start > Control Panel > All Control Panel Items > Administrative Tools > Internet Information Services (IIS) Manager**, as shown:



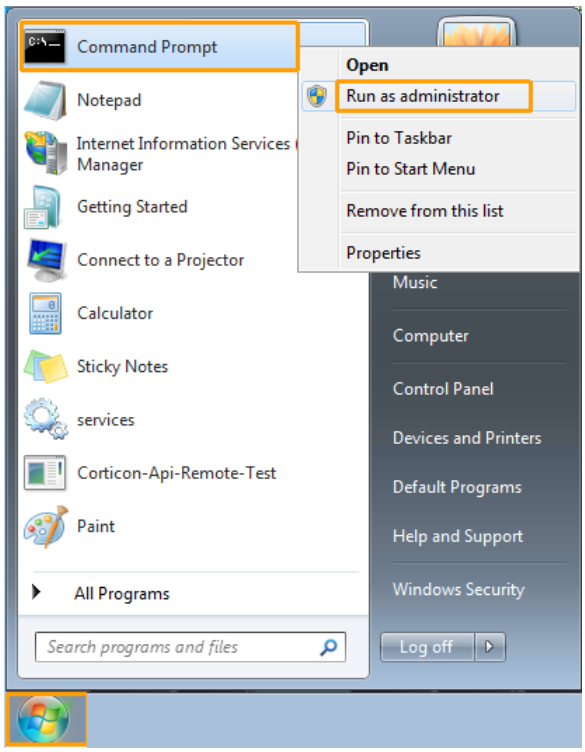
- Select **Start**, type `iis`, and then choose the IIS Manager, as shown:



Configuring IIS for .NET Framework on Windows 2008 or Windows 7

To configure IIS to use .NET Framework v4.0:

1. Open a **Command Prompt** window with Administrator privileges, as shown:

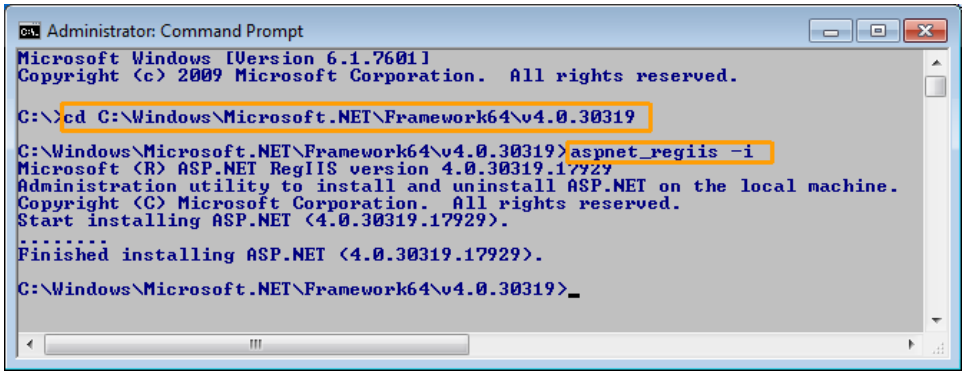


2. Enter `C:` to change to the install drive, and then change directory (`cd`) to the v4.0 Microsoft .NET Framework location, typically `C:\Windows\Microsoft.NET\Framework64\v4.0.30319`.

3. Enter the command

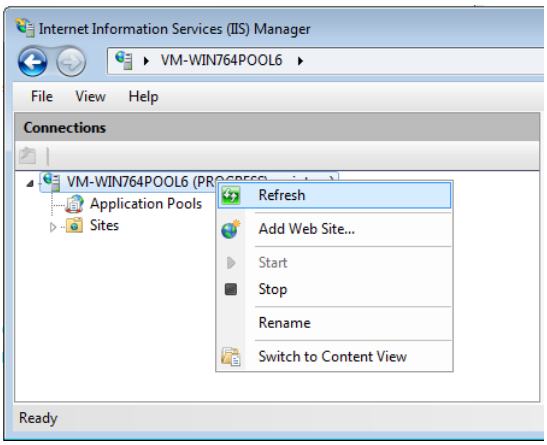
   ```
   aspnet_regiis -i
   ```

   as shown:

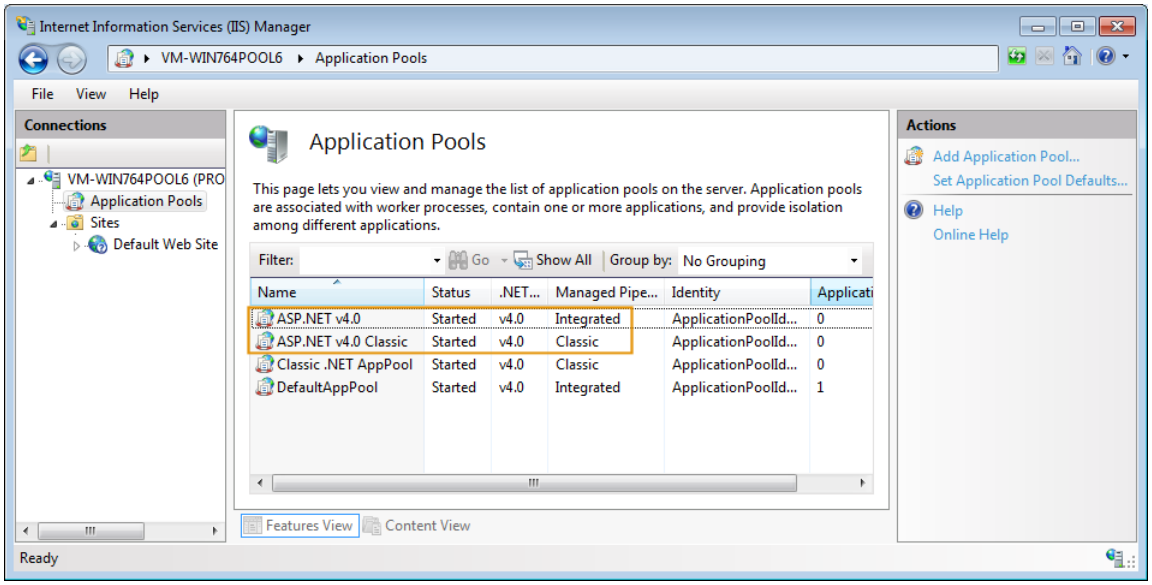


   The system registers the current directory's .NET Framework with IIS; in this example, `v4.0.30319`

4. In the IIS Manager, right-click on the root level, and then choose **Refresh**, as shown:

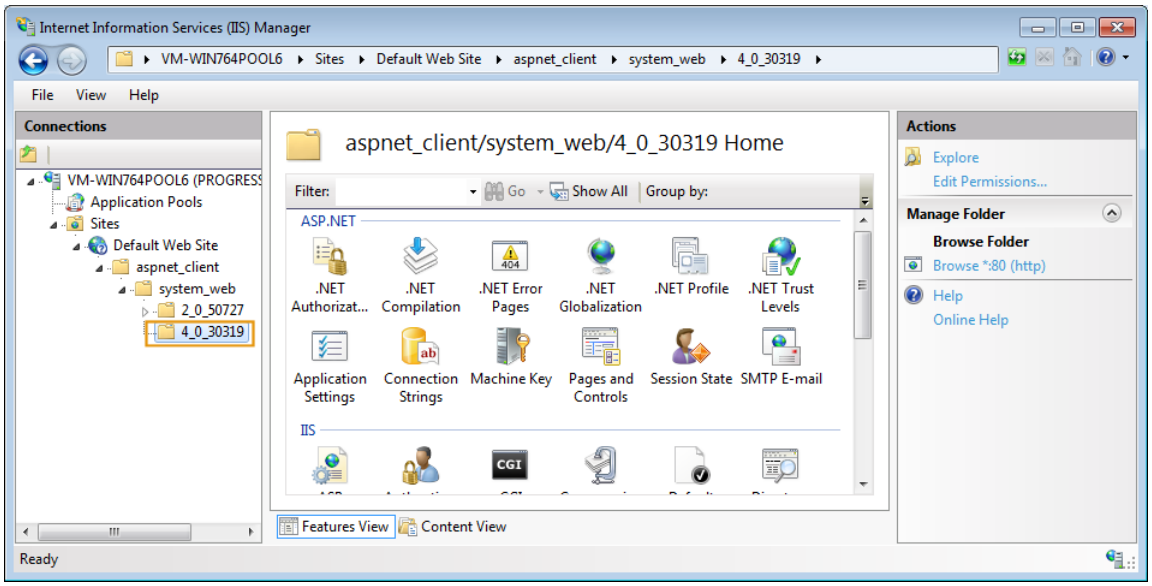


5. Click on **Application Pools** in the left panel. The Application Pools panel opens, as shown:



ASP .NET v4.0 and **ASP .NET v4.0 Classic** are listed if you are properly set up to this point.

6. In the left panel, expand **Sites > Default Web Site > aspnet_client > system_web**. Verify that .NET Framework 4.0 is listed, as shown:



(Note that .NET Framework 2.0 is also listed because IIS includes that version.)

IIS is now configured to use .NET Framework v4.0.

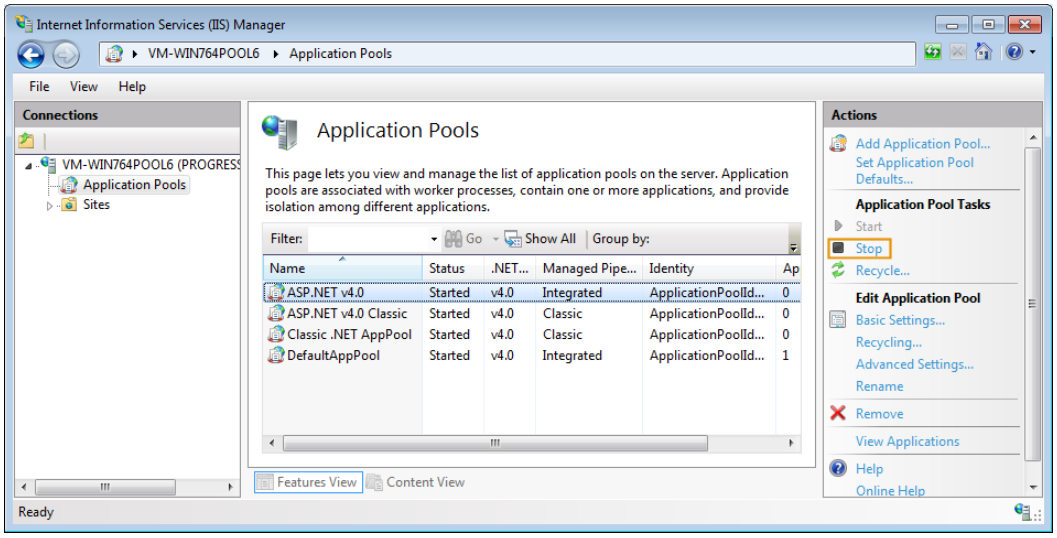
Optional steps in IIS setup on Windows 2008 or Windows 7

Some additional steps are not required yet might be important in your setup.

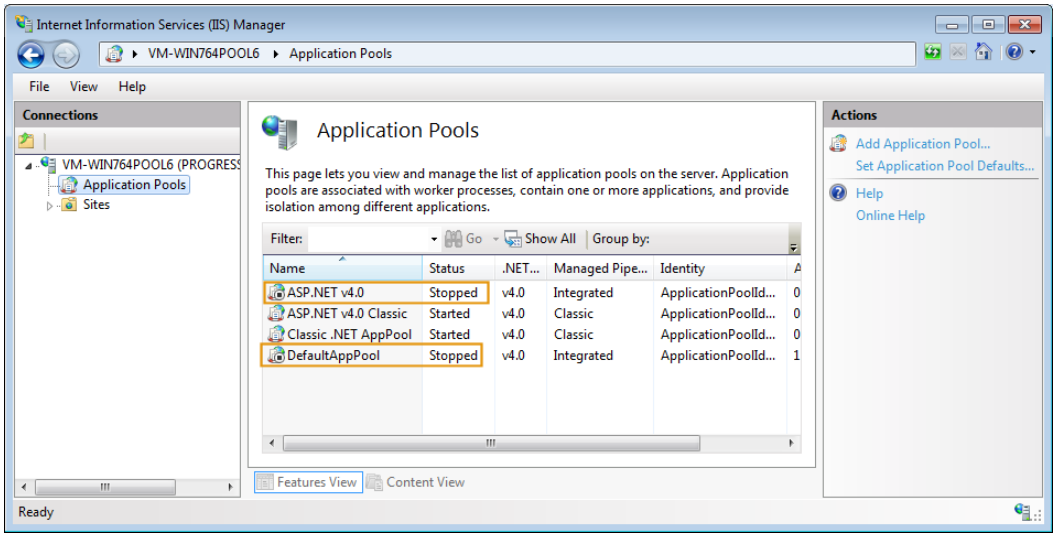
1. Click on **Application Pools** in the left panel.

   If there are **Managed Pipeline Modes** listed as *Integrated*, they must be stopped.

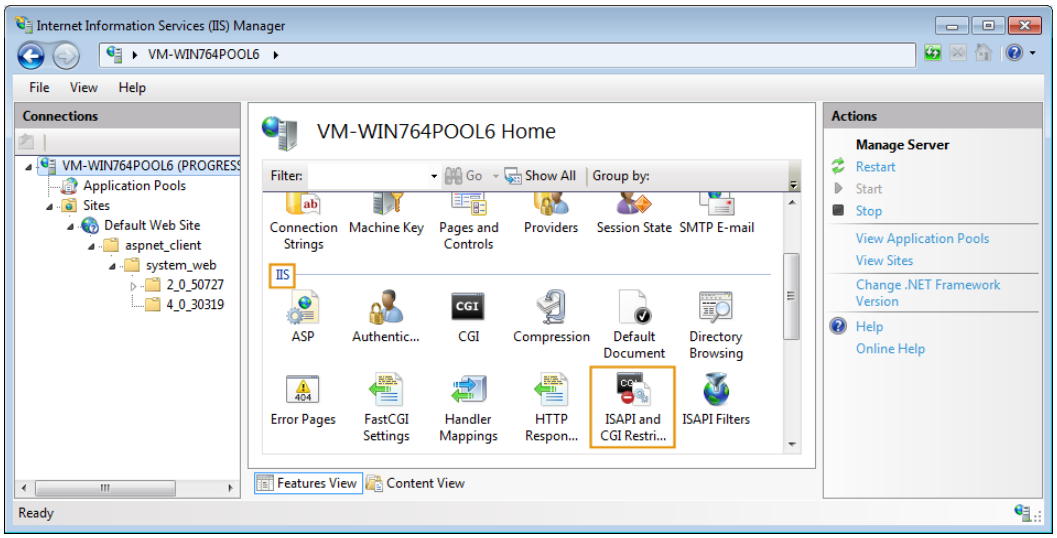
   Click on each Application Pool that is not a *Classic* type, and then select **Actions > Application Pool Tasks > Stop** in the right pane, as shown:



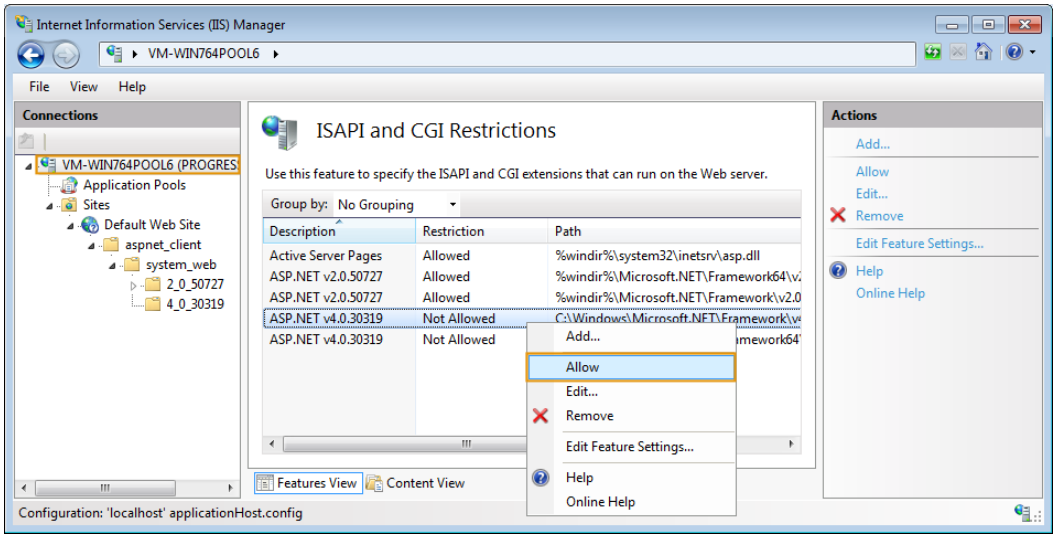
2. When completed, each integrated Application Pool has the status **Stopped**, as shown:



3. Click on the root level, locate the **IIS** section, and then click on **ISAPI and CGI Restrictions**, as shown:



4. In the **ISAPI and CGI Restrictions** panel, confirm that the ASP.NET v4.0.30319 is **Allowed**. If `Framework` or `Framework64` are listed as **Not Allowed**, right-click on each line, and then choose **Allow**, as shown:

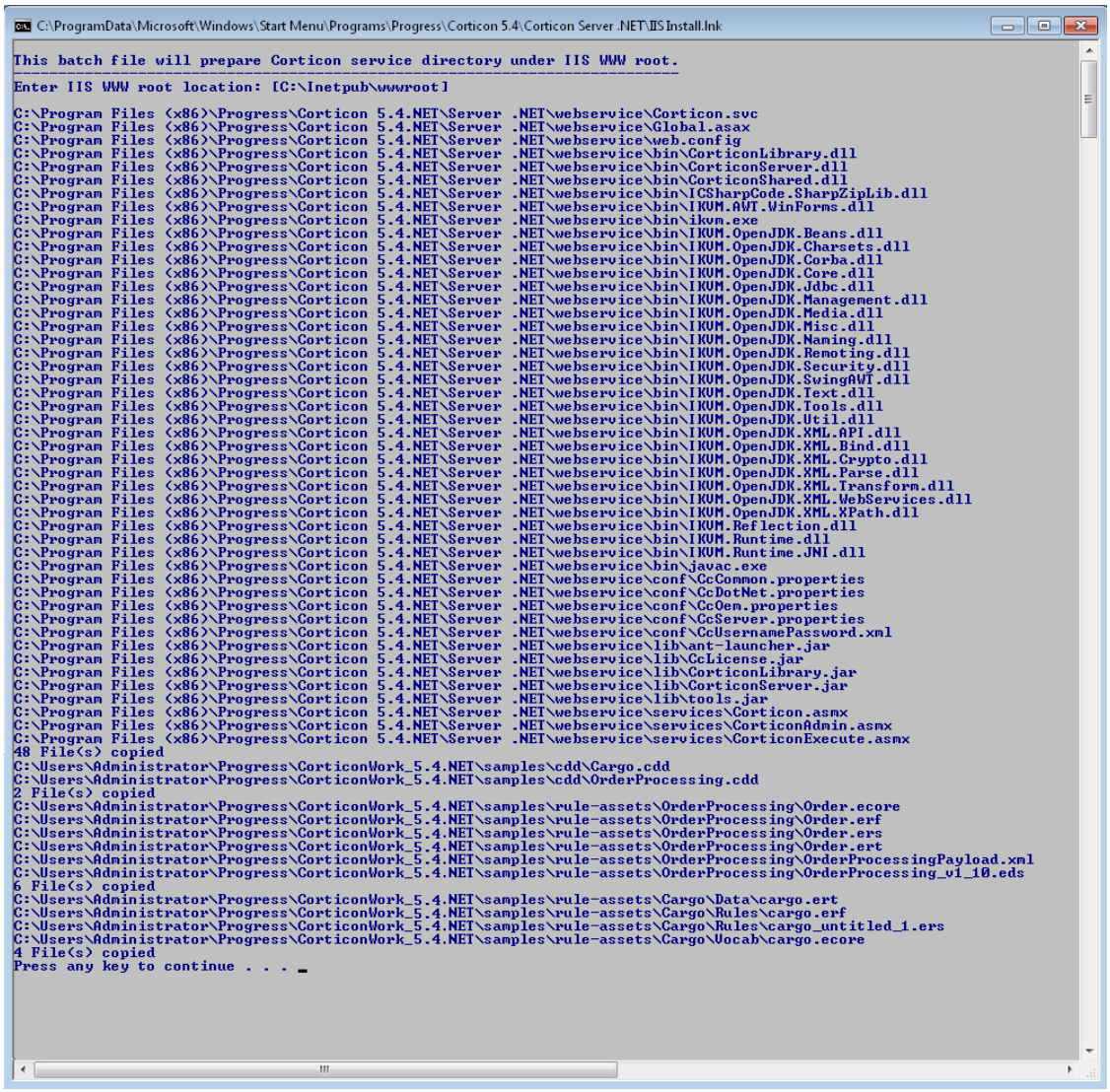


Installing Corticon Server into IIS on Windows 2008 or Windows 7

To install Corticon Server for .NET into your IIS installation:

1. In a Command Prompt, run `[CORTICON_HOME]\Server .NET\IIS\install.bat`, as shown:

2. When prompted, enter the location of the WWW root of the IIS, or press **Enter** to place it into standard IIS location in <IISRoot>, which is `C:\Inetpub\wwwroot` by default. The result is as follows:



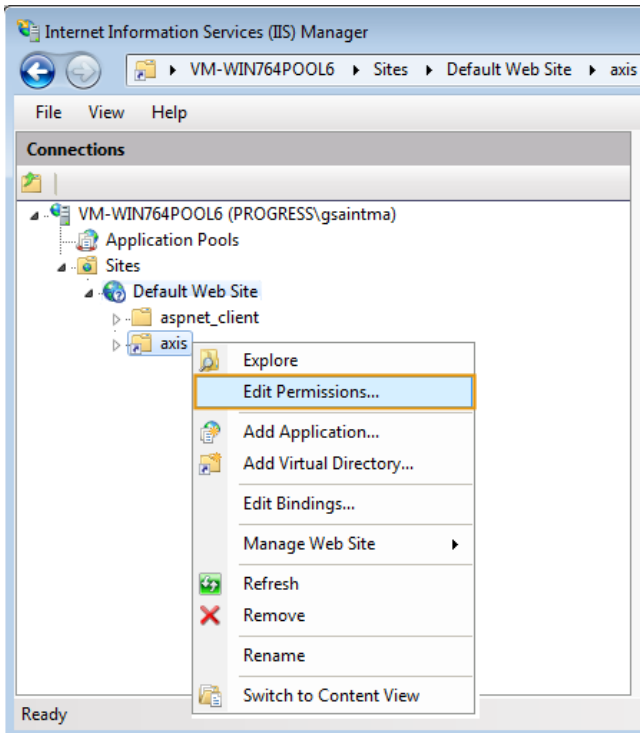
Confirm that the installers created a new directory named `axis` under `<IISRoot>\wwwroot`, typically as `C:\Inetpub\wwwroot\axis`. If not, create that directory yourself now.

Setting Access Permissions for axis directories on Windows 2008 or Windows 7

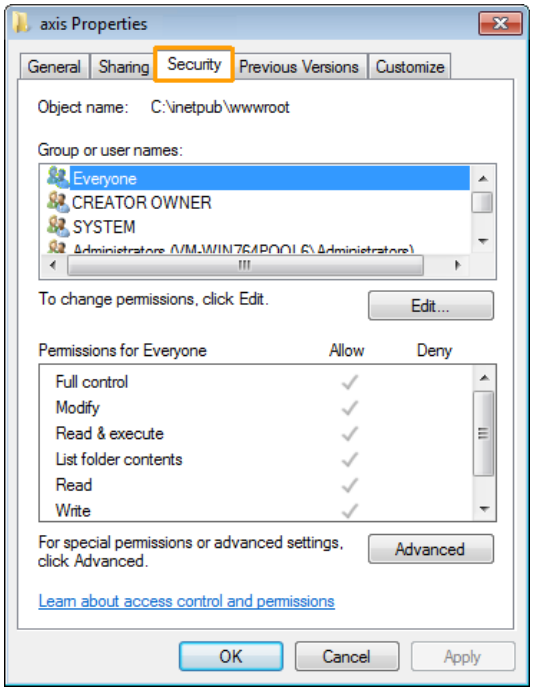
Corticon Server for .NET needs to be able to write to various subdirectories of the IIS installation_directory's `axis` subdirectory. The following task in the IIS Management Console might be required for your setup.

To set permissions for axis-related directories:

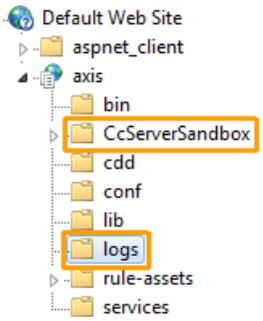
1. In the IIS Manager, right-click on **Default Web Site**, and then choose **Edit Permissions**, as shown:



2. In the **axis Properties** dialog box, choose the **Security** tab.
3. In the **Group** or user names section, click **Add**, add `Everyone`, and then click **OK**.
4. Click on the `Everyone` group, select **Full Control**, as shown:



5. Click **OK**.
6. Add users to the `Everyone` group to give them full server access.
7. Repeat these steps for the `CcServerSandbox` and `Logs` sub directories.



Note: If you are unable to make these changes, you may not have administrator rights to IIS. Contact your IIS administrator for assistance.

Note: These permissions might seem rather lax. It is recommended that you set them up this way to ensure functionality, and then you can choose to tighten the permissions to just read, write, and run scripts in a specialty group such as `IIS_IUSERS`.

Changing web.config on Windows 2008 or Windows 7

You only need to perform this step if either of the following is true:

- the <IISRoot>is a non-default location (that is, not `C:\inetpub\wwwroot`)
- the `axis` virtual directory is not under `wwwroot`

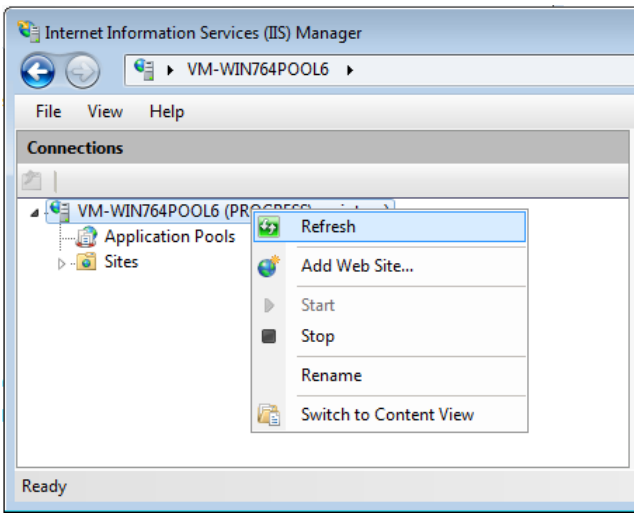
1. Using a text editor, edit the IIS `web.config` file to insert lines that add keys for Corticon home and Corticon work, as highlighted with *my* location:

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
   <add key="CORTICON_HOME" value="C:\my_inetpub\wwwroot\axis" />
   <add key="CORTICON_WORK_DIR" value="C:\my_inetpub\wwwroot\axis" />
    <add key="SOAP_LOGGING" value="false" />
    <add key="PRELOAD_CDDS" value="true" />
  </appSettings>
```

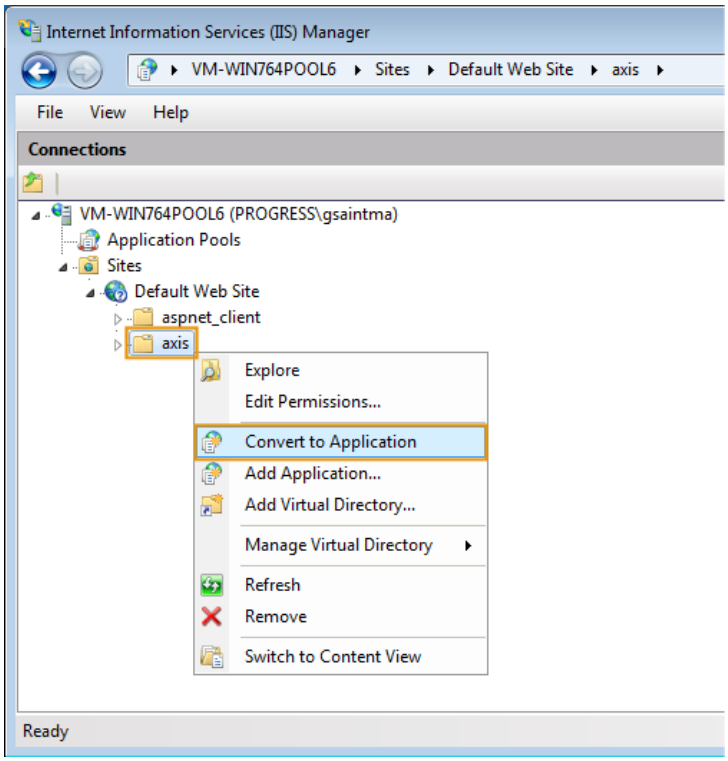
Creating the application and binding it to the Application Pool on Windows 2008 or Windows 7

To convert the axis directory into an application:

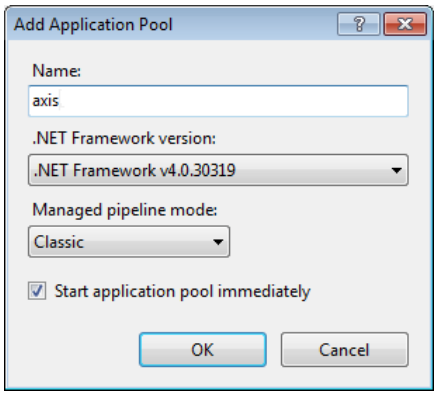
1. In the IIS Manager, right-click on the root, and then select **Refresh**.



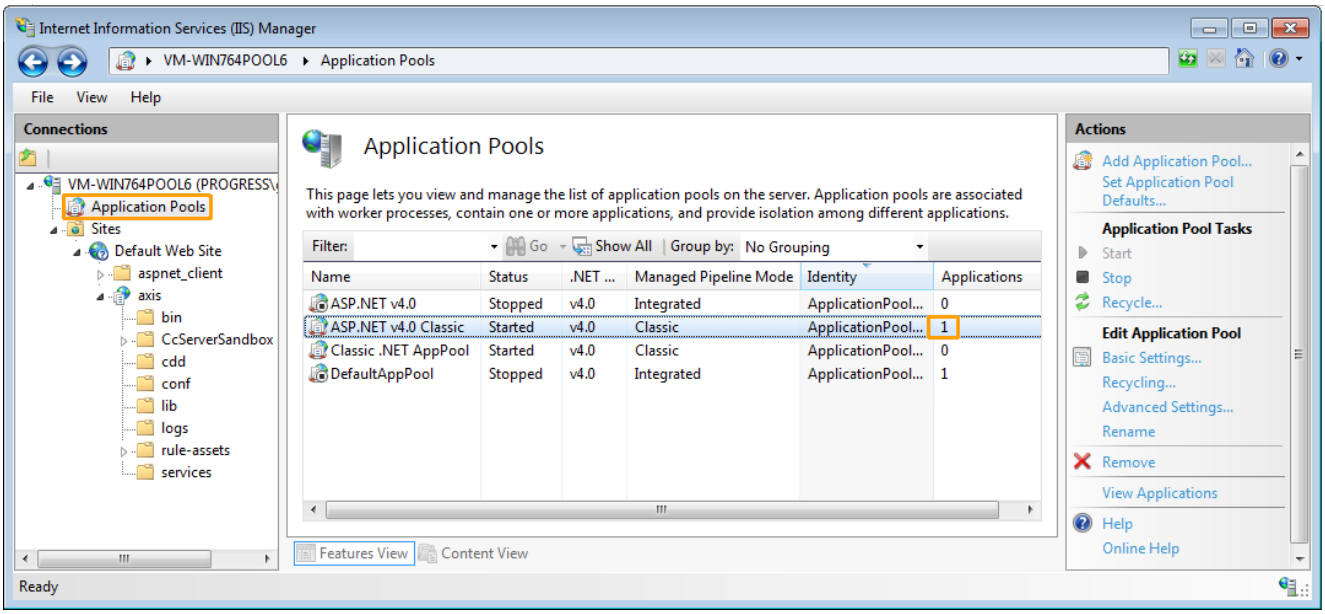
2. Expand the tree and navigate to the `axis` directory.
3. Right-click on **axis**, and then select **Convert to Application**, as shown:



4. In the **Add Application Pool** dialog t hat opened, enter `axis` as the name, and then choose the .NET Framework version **.NET Framework v4.0.30319**, and the Managed pipeline mode **Classic**, as shown:



5. Once completed, the ASP.NET v4.0 Classic indicates that it is bound to an application, as shown:

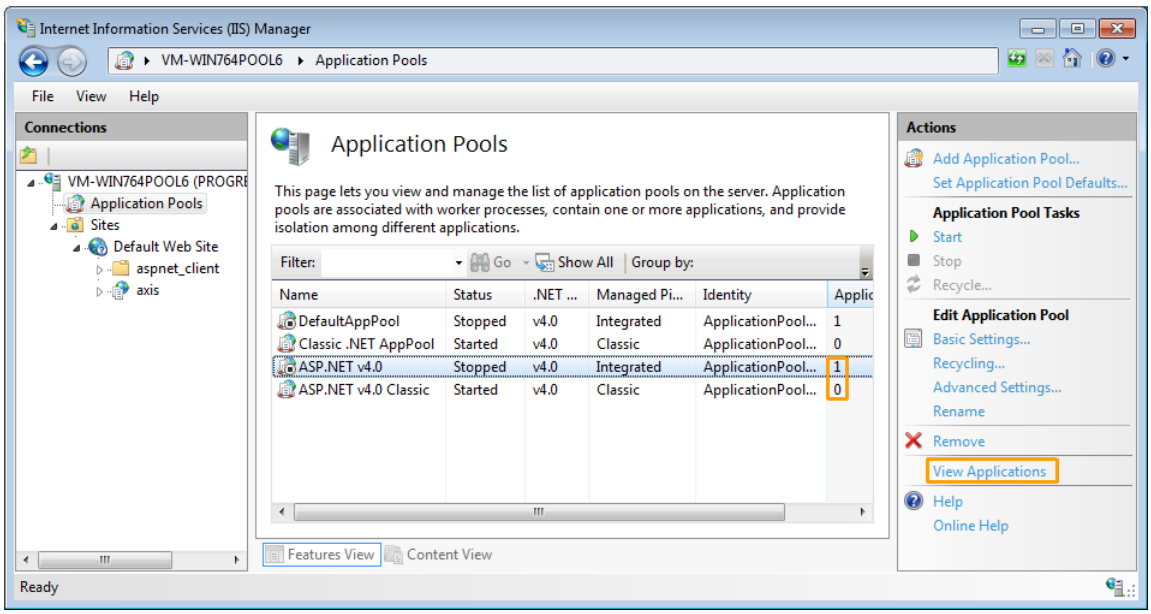


Adjusting the application's assigned pool on Windows 2008 or Windows 7

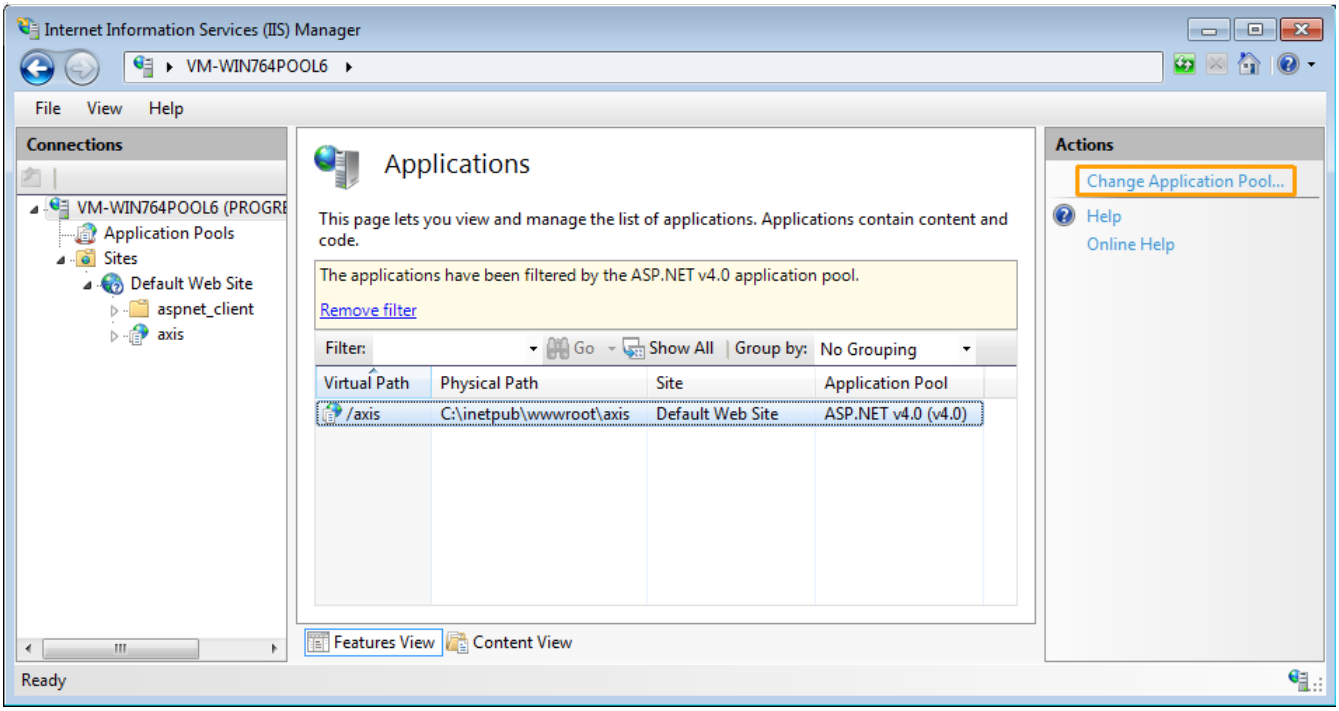
If the application is not assigned to the correct application pool, you can revise the application assignment. In the following illustrations, the stopped integrated pool hosts the application, and the running classic pool has no applications.

To change an application's assigned pool:

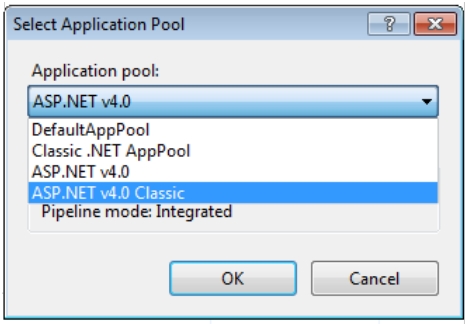
1. Click on **Application Pools**, select the inappropriately assigned pool for the application, and then click **View Applications** in the right panel, as shown:



2. The **Applications** panel lists each application's binding to the selected application pool. As the `axis` application is not set to **ASP.NET v4.0 Classic**, right click on the application, and then click **Change Application Pool**, as shown:



3. In the **Select Application Pool** dialog box, select **ASP.NET v4.0 Classic** in the dropdown list (as shown), and then click **OK**.

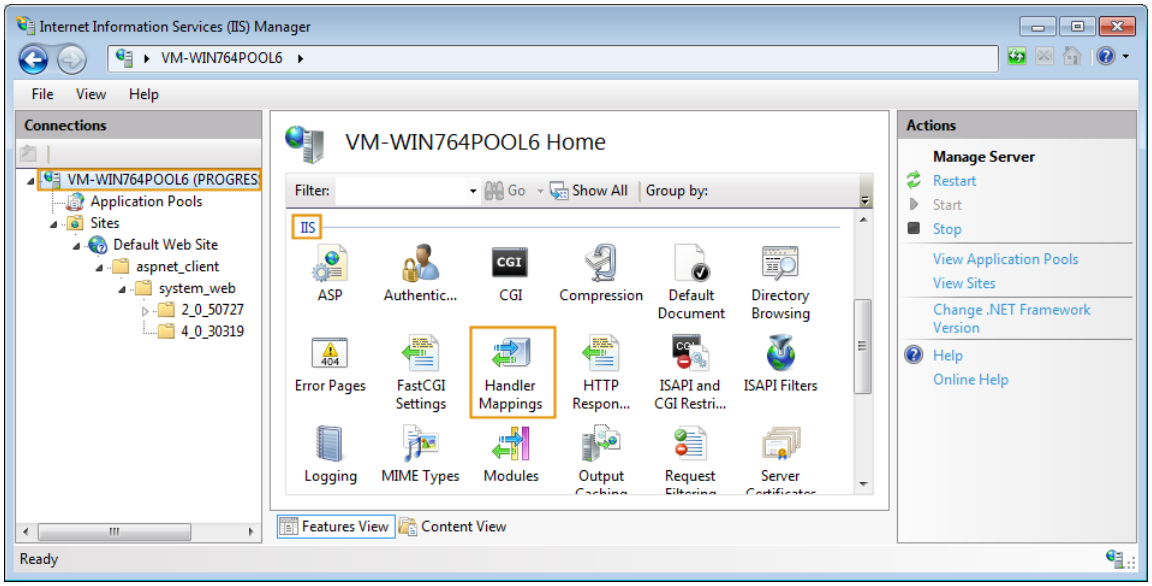


The application `axis` assigned to the **ASP.NET v4.0 Classic** application pool.

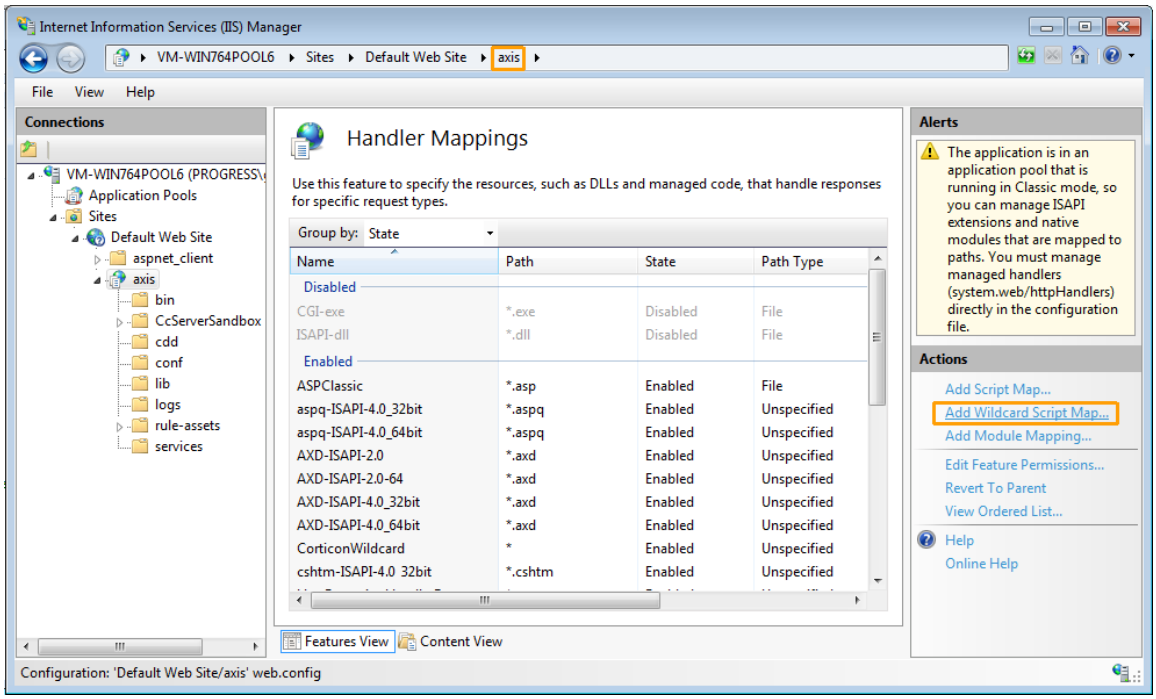
Configuring Corticon Wildcard on Windows 2008 or Windows 7

To configure Corticon Wildcard for `.asmx` file extensions:

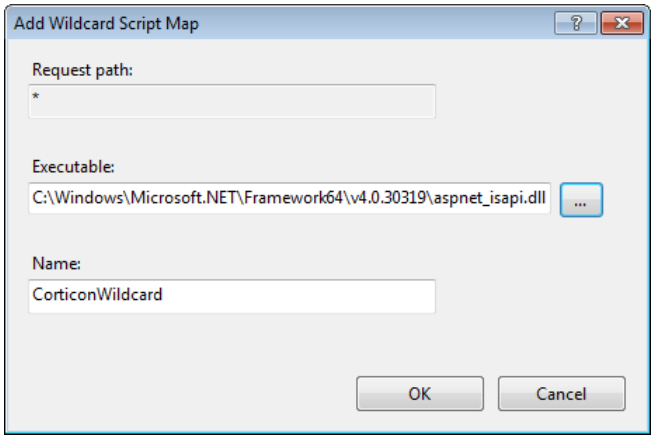
1. Click on the top level node in the tree on the left side, and then double click **Handler Mappings**, as shown:



2. On the **Handler Mappings** page's right panel, click **Add Wildcard Script Map**, as shown:



The **Add Wildcard Script Map** dialog box opens, as shown:



3. As shown:

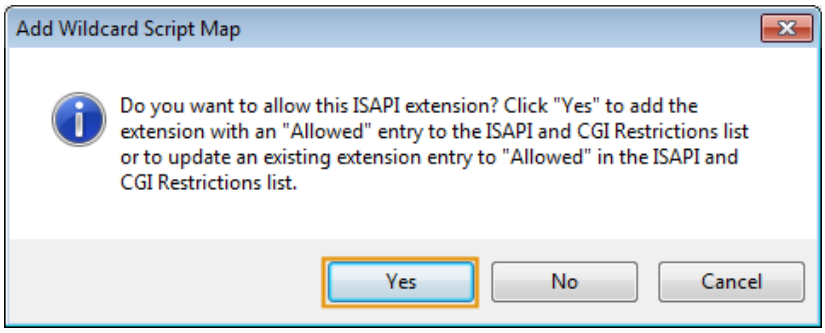
   - **Request path**: Read-only and defaulted to *****.

   - **Executable**: Browse to the .NET Framework 4.0 installation directory, and then select `aspnet_isapi.dll`.

   - **Name**: Enter an identifying name, such as `CorticonWildcard`.

   Click **OK**.

4. In the confirmation alert that opens, click **Yes**.



5. It is a good idea to restart IIS now. To do so in the IIS Manager, right-click on the root to choose **Stop**, and then after a few seconds, choose **Start**. (You might prefer to execute `iisreset` in a Command Prompt to restart IIS.)

The Windows 2008 or Windows 7 installation with Internet Information Servrices 7.5 is now running the `axis` web service and Corticon ASP.NET Webservices Source files.

Now, skip over Windows 2012 setup information, and then continue at .

Windows Server 2012, R1 and R2 (IIS 8.0 and 8.5)

The procedures in this topic apply to Windows Server 2012 (IIS 8.0) as well as to Windows Server 2012 R2 (IIS 8.5).

Note: Changes in Windows Server 2012 security that give an Administrator less global authority call for some additional tasks. These tasks are: (1) creation of a sandbox in the `Windows/System32` directory is prohibited; instead create it elsewhere, and (2) URL rewrite does not work on the server; instead add `/services/Corticon.asmx` so that SOAP requests perform correctly.

Confirm that .NET Framework 4.5 is installed

Microsoft .NET Framework 4.5 is the supported version for this release of Corticon on Windows Server 2012. It is included in the operating system installation. You might want to confirm its existence on the machine already by locating the v4.0.X folder at `C:\Windows\Microsoft.NET\Framework64\` or `C:\Windows\Microsoft.NET\Framework\`.

The following procedures outline the general instructions for setting up a Microsoft Windows installation to use Internet Information Services (IIS).

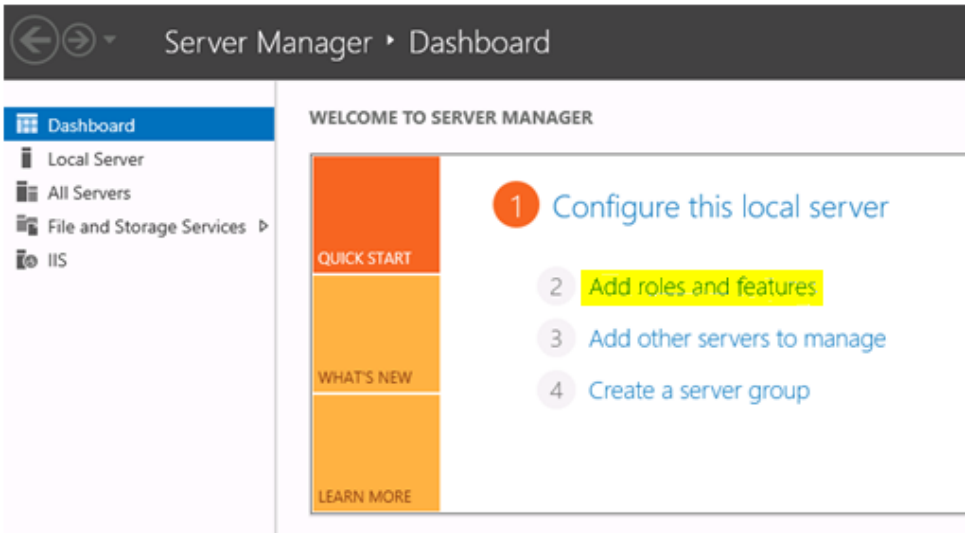
Adding the IIS feature in the Server Manager

Windows Server 2012 installations include Internet Information Server (IIS) 8.0 (8.5 in R2), but require user interaction to enable it.

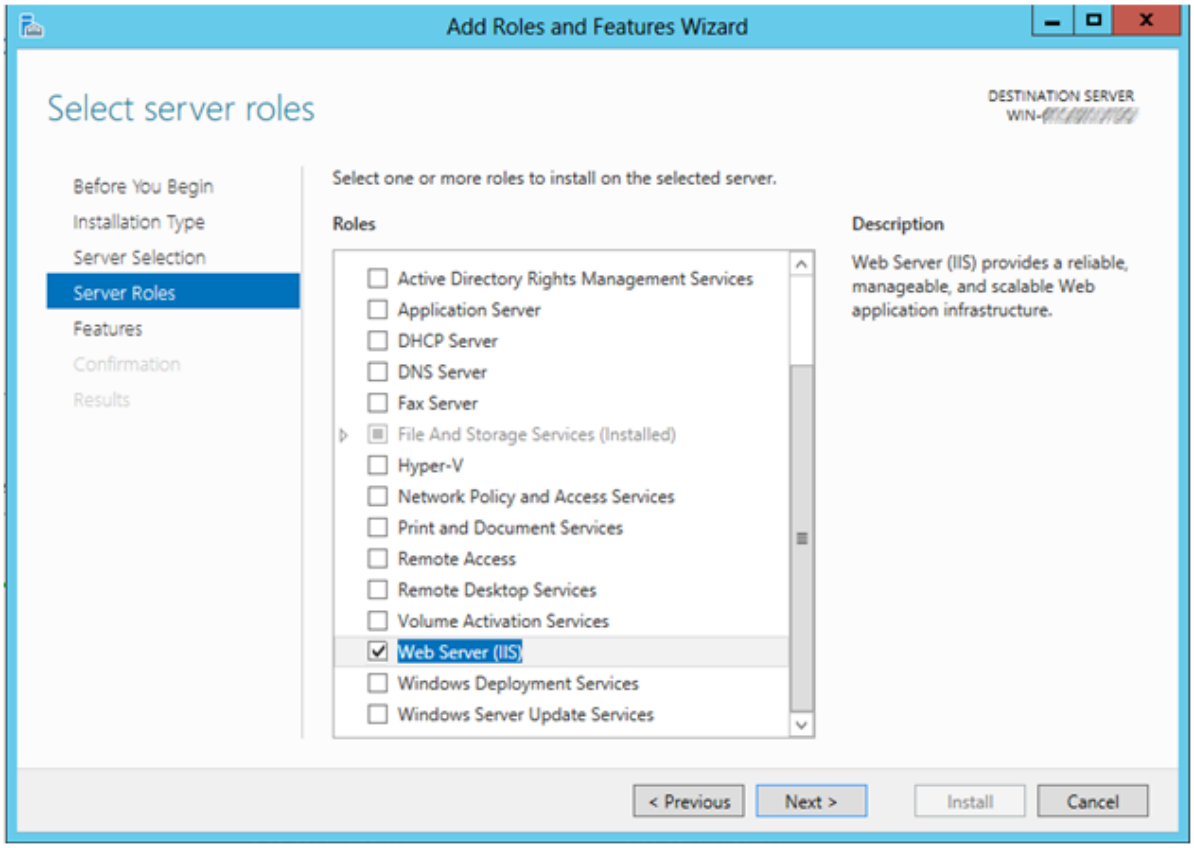
Note: If IIS is not installed, you might need to access download sites or appropriate operating system media.

To enable IIS on a host Windows Server 2012 machine for Corticon Server for .NET:

1. Choose **Start > Administrative Tools**, and then choose **Server Manager**. The **Server Manager** opens.
2. Choose **Add Roles and Features**, as highlighted:

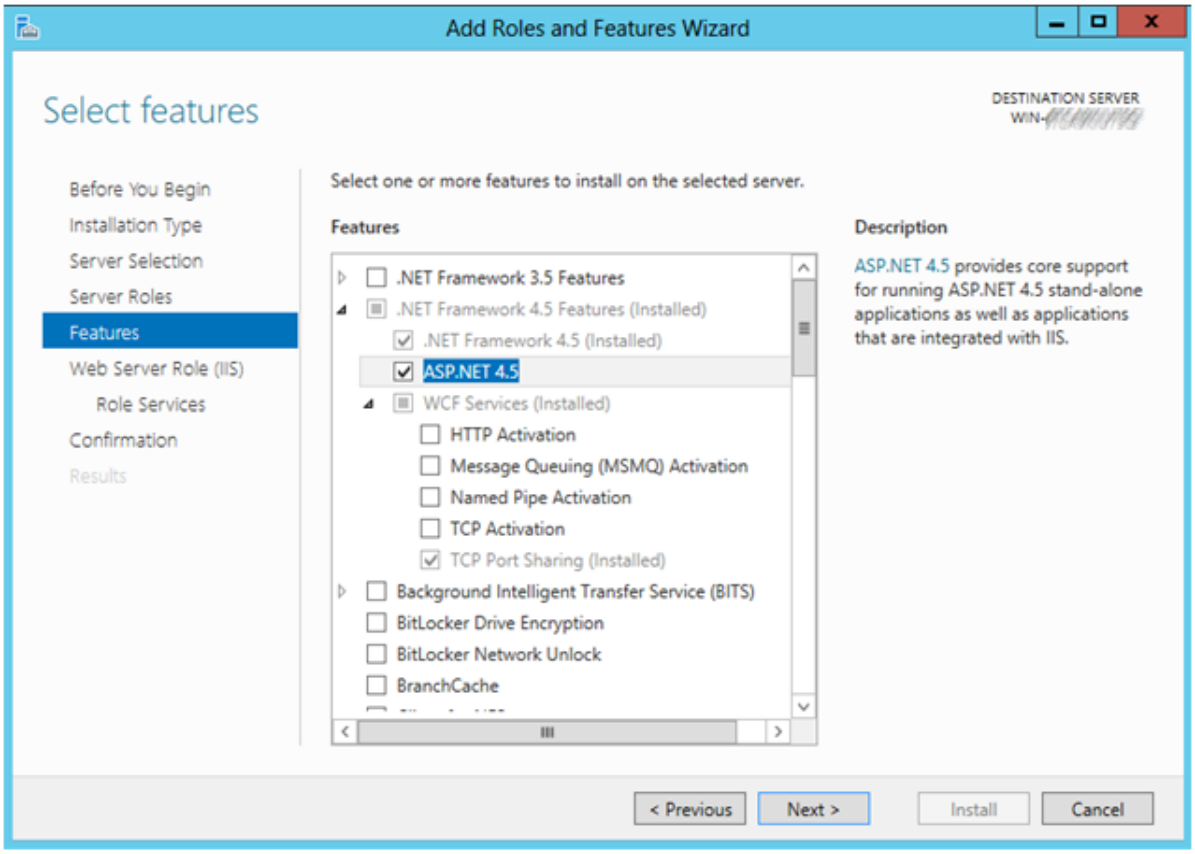


3. In the **Add Roles and Feature Wizard** that opens, choose the **Installation Type** option **Role Based**, and then click **Next**.
4. On the **Server Selection** panel, select the Windows Server 2012 installation you are setting up for IIS, and then click **Next**.
5. On the **Server Roles** panel, check **Webserver (IIS)**, as shown:



Click **Next**.

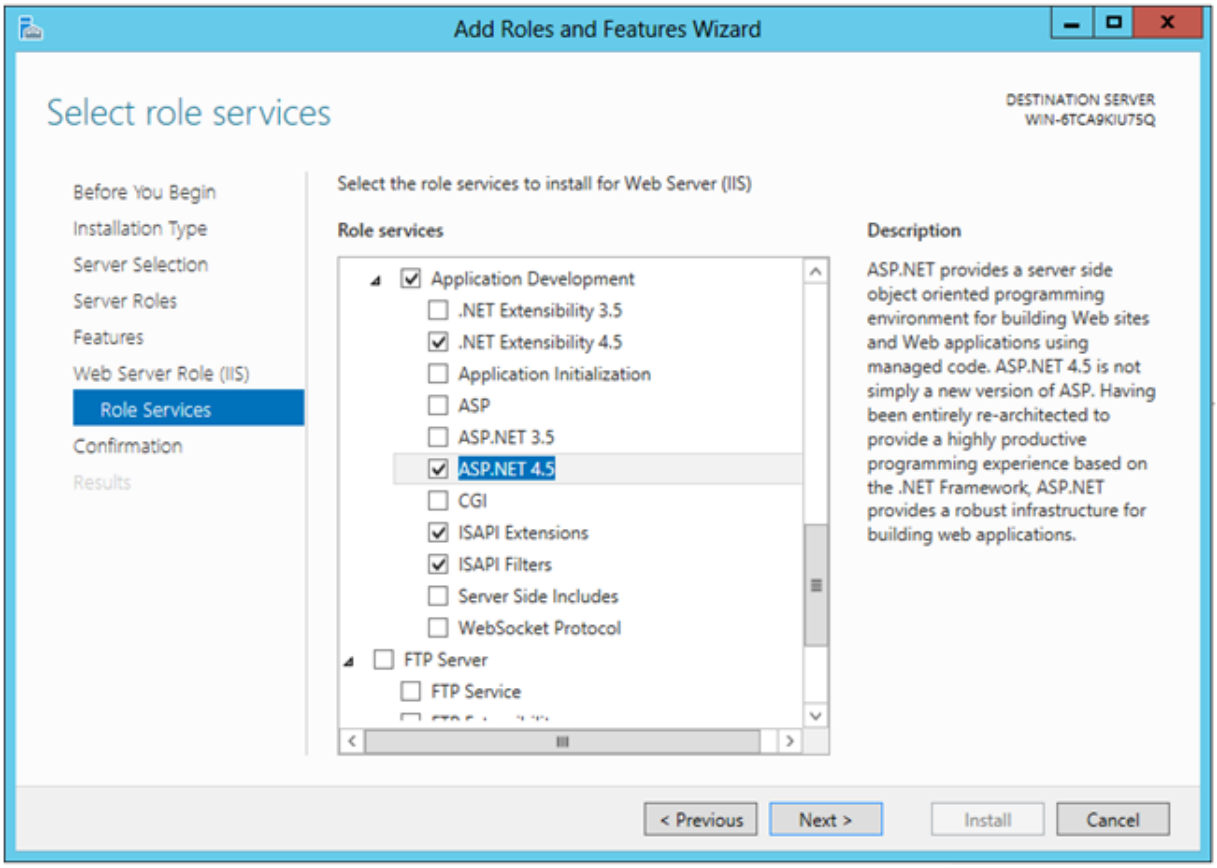
6. On the **Features** panel, check **ASP.NET 4.5**, as shown:



Click **Next** twice to advance to the **Role Services** panel.

7. In the **Role Services** options, in , select **Application Development** and its options:
   - **.NET Extensibility 4.5**
   - **ASP.NET 4.5**
   - **ISAPI Extensions**
   - **ISAPI Filters**

   as shown:



8. Click **Next**, and then click **Install**.

Windows Server 2012 installs the selected components. When you restart, IIS 8.x is enabled.

Installing Corticon Server into IIS 8.x

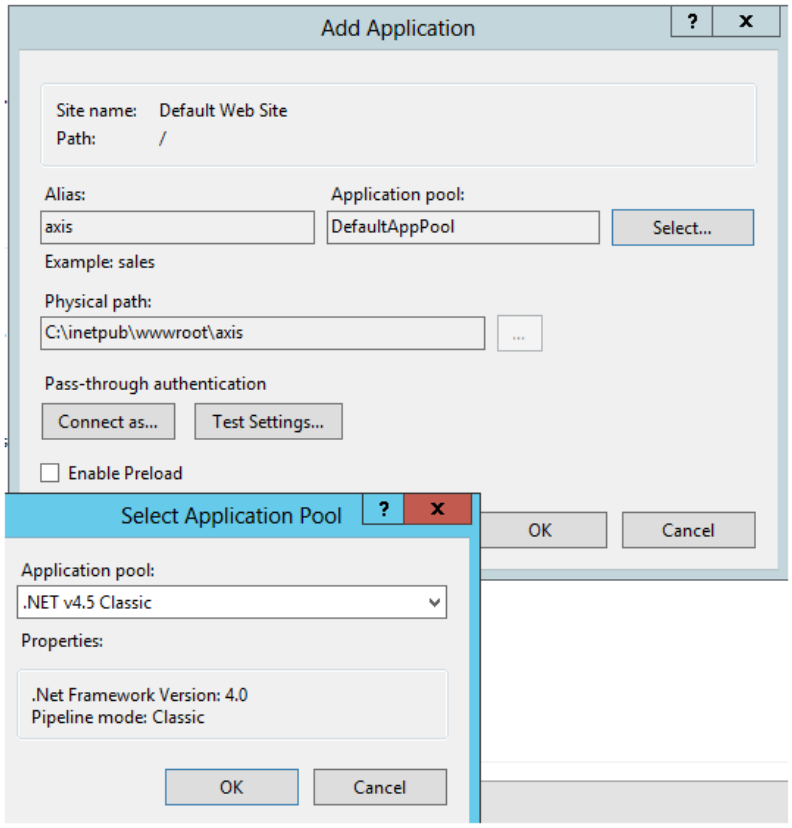
To install Corticon Server for .NET into your IIS 2012 installation, run `[CORTICON_HOME]\Server.NET\IIS\install.bat`, as shown:

When prompted, enter the location of the WWW root of the IIS, or press **Enter** to place it into standard IIS location in <IISRoot>, which is `C:\Inetpub\wwwroot` by default.

Creating the application in IIS 8.x

To convert the `axis` directory into an application:

1. In the IIS Manager, select the **Default Website**, then right-click on the root, and select **Refresh**.
2. Expand the tree and navigate to the `axis` directory.
3. Right-click on the `axis` folder, and then chose **Convert to Application**.
4. In the **Add Application Pool** dialog that opened, choose **.NET v4.5 Classic**, as shown:



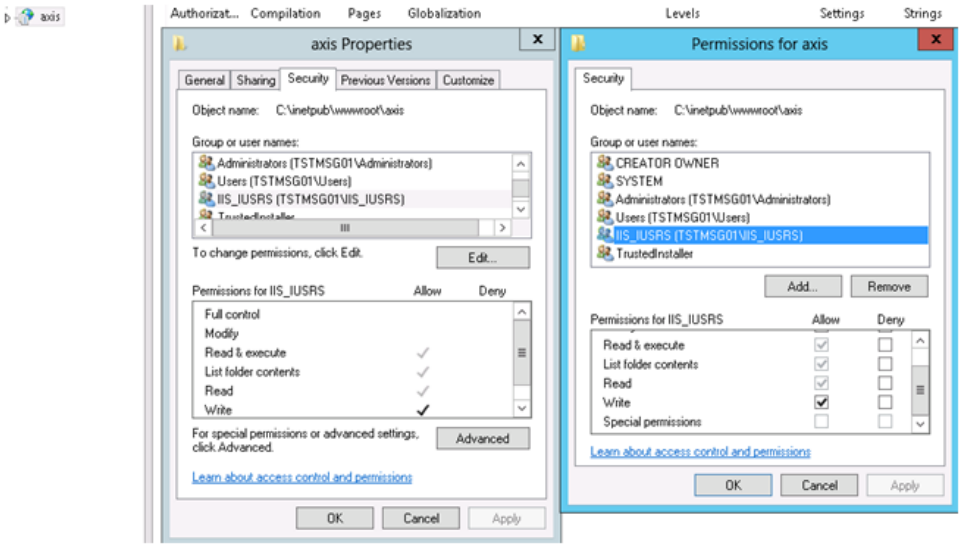
.

5. Click **OK** on both dialog boxes.

Setting Access Permissions for axis directories in IIS 8.x

Corticon Server for .NET needs to be able to write to various subdirectories of the IIS installation directory's `axis` subdirectory. The following task might be required for your Windows Server 2012 setup, as the V4.5 Classic Application Pool runs under the built in user account `ApplicationPoolIdentity`, a user that has no `WRITE` permissions for the axis folders, so these rights must be set.

Right-click the `axis` application and chose **Edit Permissions**. Assign `WRITE` permission to the account `IIS_IUSRS`, as shown:



The installation is now complete.

It is a good idea to restart IIS now. To do so in the IIS Manager, right-click on the root to choose **Stop**, and then after a few seconds, choose **Start**. (You might prefer to execute `iisreset` in a Command Prompt to restart IIS.)

The IIS -- Windows 2012 with Internet Information Services 8.0 or Windows 2012 R2 with Internet Information Services 8.5 -- is now running the `axis` web service and Corticon ASP.NET 4.5 Webservices Source files.

5

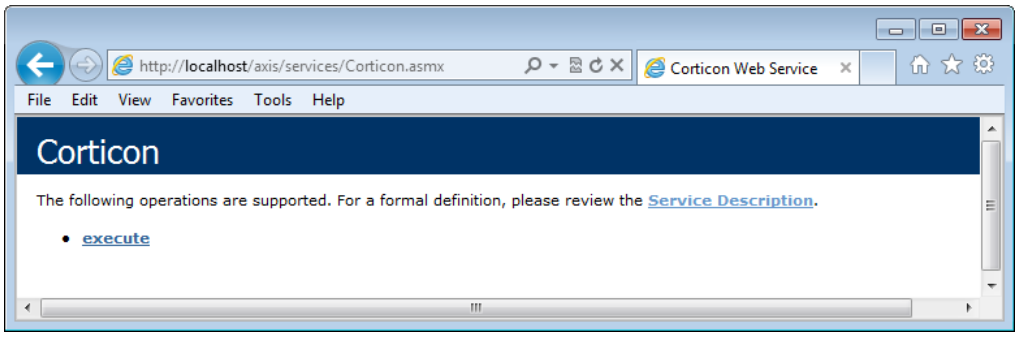
Testing the configuration

In a browser, access the following URLs (assuming that IIS is running on its default port 80):

- `http://localhost/axis/services/CorticonAdmin.asmx`
- `http://localhost/axis/services/Corticon.asmx`
- `http://localhost/axis/services/CorticonExecute.asmx`

Selecting any of these URLs displays a web page in the following format:

Figure 1: Testing the Corticon Web Service



With Corticon Server for .NET installed in IIS, it is a good practice to test the remote server setup to ensure it is running and listening. At this point, no Decision Services have been deployed, so Corticon Server for .NET is not yet ready to process transactions.

6

Updating your Corticon license JAR for .NET

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 Deployment Console

To configure Corticon .NET Server to access its license file:

1. Copy the license JAR with its default name, `CcLicense.jar`.
2. Navigate to the .NET Server installation's `[CORTICON_HOME]\samples\lib` directory to paste the file and overwrite the existing file in that location.
3. Navigate to the .NET Server installation's `[CORTICON_HOME]\webservice\lib` directory to paste the file and overwrite the existing file in that location.

When you launch the Corticon Deployment Console, your license with its registration information is registered for the Corticon Deployment Console. When 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 .NET Server's license location:

1. Navigate in the file system to the Corticon .NET Server installation's, `[CORTICON_HOME]\Server .NET\samples\bin` subdirectory.
2. Double-click on `Corticon-Api-Inprocess-Test.exe`, 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\myCorticonEDC_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. Once the .NET Server is running in remote mode, you can double-click on `Corticon-Api-Remote-Test.exe` in that same directory to perform the same `416` and `415` tasks as in Step 2 above

7

Corticon .NET 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** 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. 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.
- The **Corticon Server for deploying web services with .NET** facilitates deployment on Windows .NET framework 4.0 and Microsoft Internet Information Services (IIS) that are packaged in the supported operating systems. The guide *Corticon Server: Deploying Web Services with .NET* 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.

Note: This is the Corticon .NET Server documentation. For corresponding information on the Corticon Java Server, see *Deploying Web Services with Java*.

For details, see the following topics:

- Setting up Corticon .NET Server use cases
- The Corticon .NET Server home and work directories
- The Corticon .NET Server Sandbox
- Testing the installed Corticon .NET Server

Setting up Corticon .NET Server use cases

In most production deployments, Corticon Server JARs are bundled and given an 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 container where you intend to deploy the Corticon Server.

Corticon Studio makes in-process calls to the same Corticon Server classes (although packaged differently-- see the following topic) 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 .NET Server as an in-process server

If you choose to manage Corticon .NET 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 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 classes, it can easily be deployed in-process in an application. When deployed in-process, the following tasks are the responsibility of the client application:

- Management of application settings, ensuring the base set of Corticon Server classes is properly referenced.
- Lifecycle management, including server startup/shutdown
- Security (if needed)

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 files, see Installing Corticon Server for .NET on page 15 of this guide.

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

1. Place the following Corticon Server directories and their contents in a directory that is accessible by the application container.
   - `/bin`
   - `/lib`
   - `/conf`
2. Configure the application to reference all DLL files located in the `/bin` directory.
3. Change the `logPath` property in `brms.properties` (located at the root of <codeph>[CORTICON_WORK_DIR]</codeph>) to an explicit path to a directory for the Corticon Server log file.
4. Write code that:
   - Initializes Corticon Server
   - Sets the following three environment variables:

  - CORTICON_HOME - A directory value for the location of /bin, /lib, and /conf.
  - CORTICON_WORK_DIR - The explicit path to the working directory
  - CORTICON_LICENSE - The explicit path to the CcLicense.jar file.
- Deploys the Decision Services into the 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 .NET Server. This code is packaged as the executable Corticon-Api-Inprocess-Test.exe in the [CORTICON_HOME]\Server .NET\samples\bin directory.

The Corticon .NET Server 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.

.NET Configurations that use global environment settings

CORTICON_HOME and CORTICON_WORK_DIR

These properties are set in the web.config file when they are different from the standard IIS location, IISRoot. See Changing web.config for detailed instructions.

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 located at the root of [CORTICON_WORK_DIR].

It is a good practice to use global environment settings

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

The Corticon .NET 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 located at the root of `[CORTICON_WORK_DIR]`. 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 .NET Server

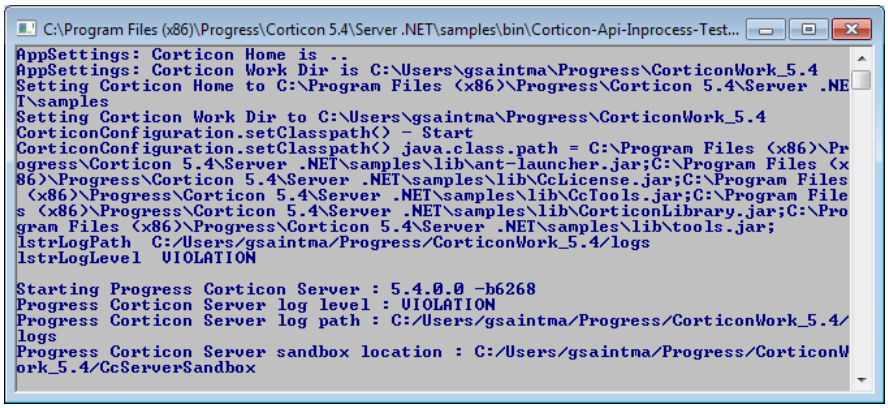
With Corticon Server installed in the environment and container of your choice, it is useful to test the installation to ensure Corticon Server is running and listening. At this point, no Decision Services have been deployed, so Corticon Server is not ready to process transactions. However, the Corticon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

Testing as in-process

Sample code is provided that demonstrates an in-process deployment of Corticon .NET Server. This code is packaged as the executable `Corticon-Api-Inprocess-Test.exe` in the `[CORTICON_HOME]\Server .NET\samples\bin` directory.

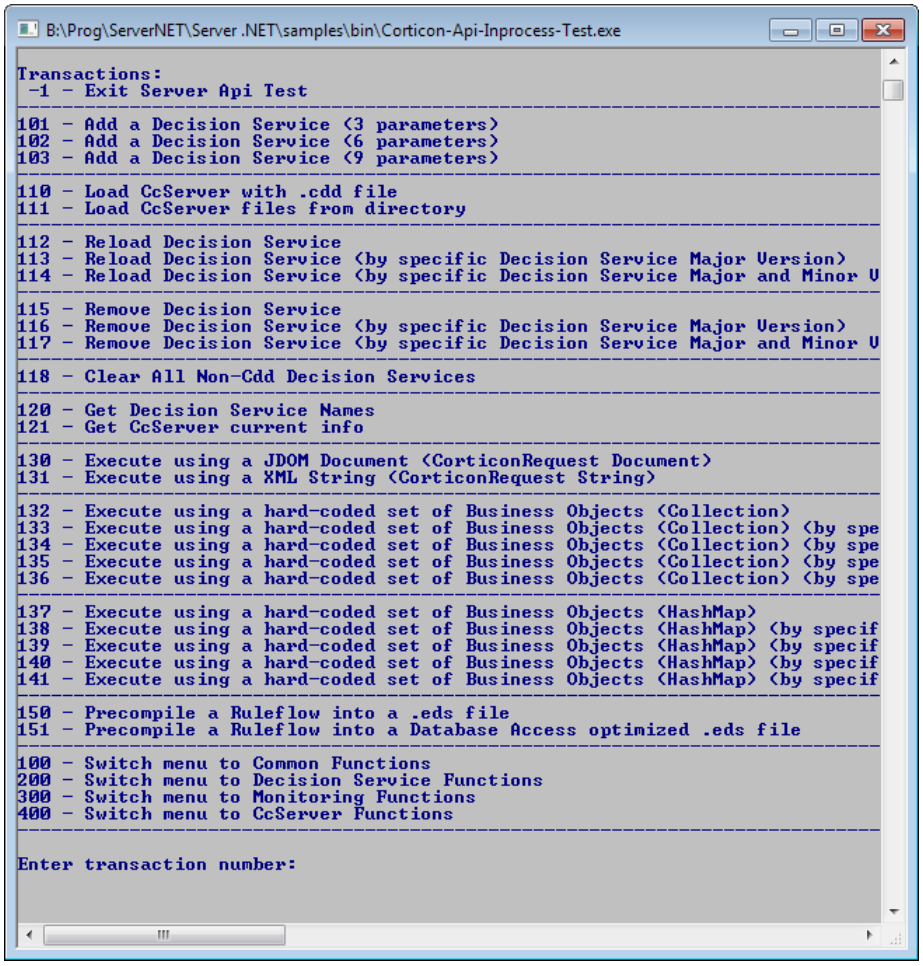
The API in-process test opens a Windows console and displays the API menu, as shown below:

Figure 2: Top Portion of the .NET Server in-process API console



The menu displayed in the Windows console is too large to fit on a single printed page, so it has been divided into two screenshots here. In the upper portion of the Windows console, shown in the figure above, the class loading process is visible. Once all classes are loaded, Corticon .NET Server starts up in the IIS.

Figure 3: Lower Portion of the .NET Server in-process API console



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:

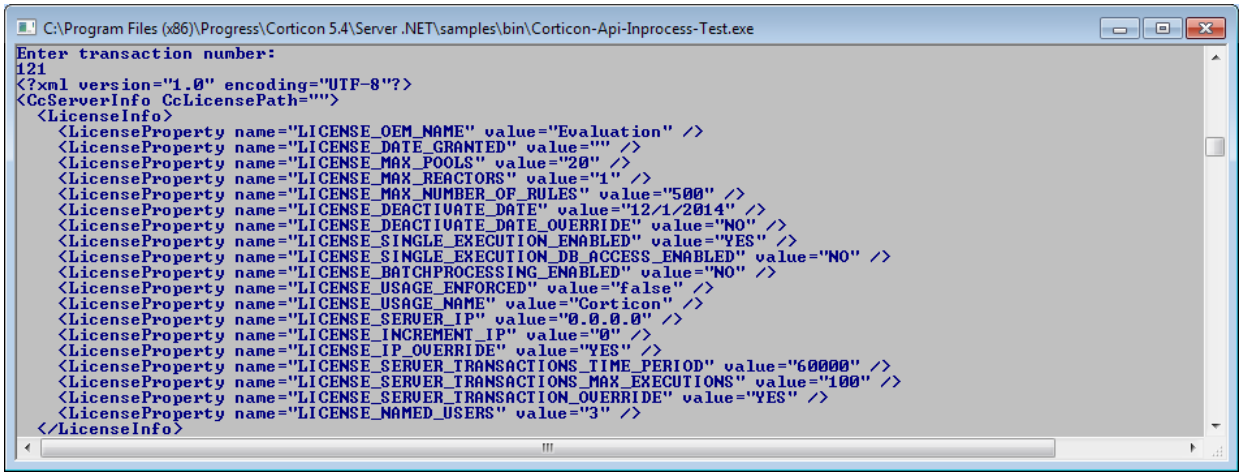
- Enter `200` to list the Decision Service Functions command set
- Enter `300` to list the Monitoring Functions command set
- Enter `400` to list the CcServer Functions command set
- Enter `100` to again list the Common Functions command set

Note: After you enter a transaction, the result is displayed followed 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 need to use an administrative method to test if Corticon Server is loaded in-process correctly. A good administrative method to call is option #121, **Get CcServer Info**. This choice corresponds directly to the corresponding API method `getCcServerInfo()`.

To try this, enter `121` in the command window. The `CcServerApiTest` class makes a call to the Corticon Server running in-process. 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 figure:

Figure 4: .NET Server in-process API console response to command 121



We haven't loaded any Decision Services, so Corticon Server is basically replying with an empty status message. But the important thing is that we have verified that Corticon .NET Server is running correctly in-process and is listening for, and responding to, calls. At this stage in the deployment, this is all we want to verify.

There is also a sample application test for the in-process Corticon Server. This code is packaged as the executable `Corticon-Api-Example.exe` in the `[CORTICON_HOME]\Server .NET\samples\bin` directory.

Testing a remote server on IIS

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

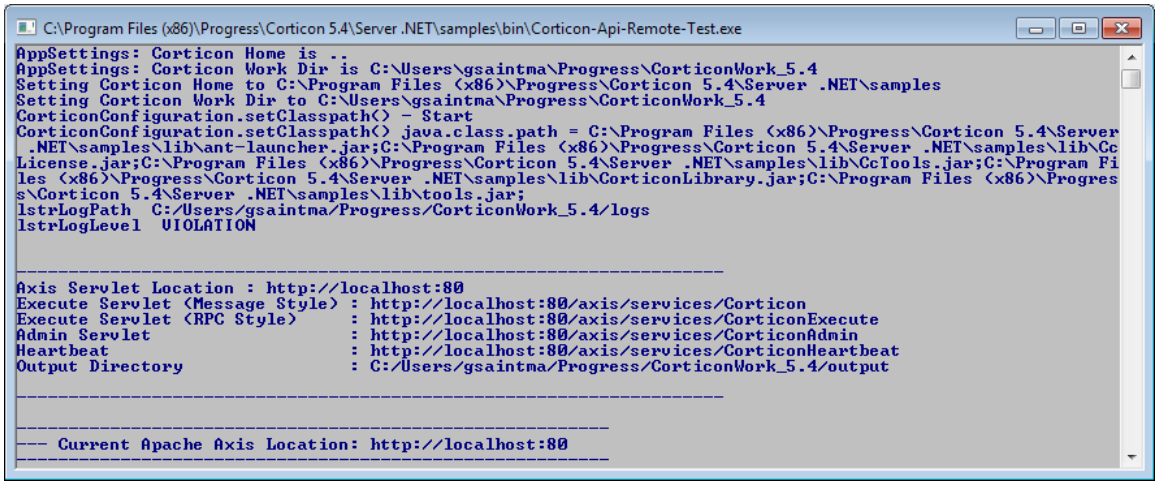
Testing the installation here assumes you have already installed and started Corticon Server as a Web Service, according to the instructions in this guide. Be sure that you have created an application from the axis directory, and that it is bound to application pools appropriately, as detailed in Setting up Windows Internet Information Services on page 21.

Because a SOAP service 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 send API calls through a SOAP message.

Sample code is provided that demonstrates a remote deployment of Corticon .NET Server on IIS. This code is packaged as the executable `Corticon-Api-Remote-Test.exe` in the `Server\bin` directory of your Corticon .NET Server installation directory.

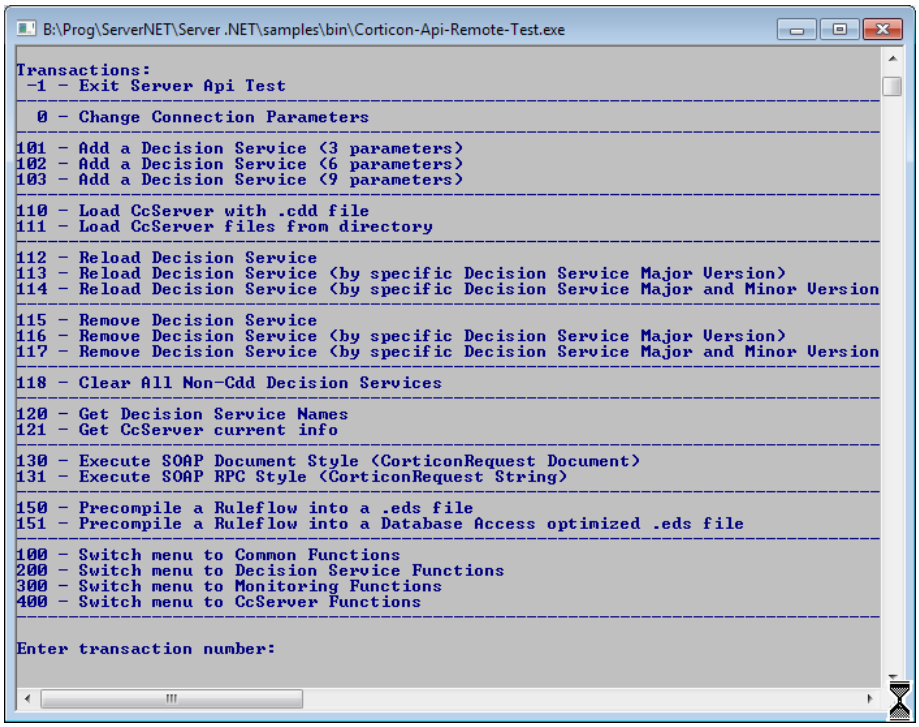
When executed, it opens a Windows console and displays the API menu, as shown below:

Figure 5: Top Portion of the .NET Server remote API console



The menu displayed in the Windows console is too large to fit on a single printed page, so it has been divided into two screenshots here. In the upper portion of the Windows console, shown in the figure above, the classpath definition process is visible. Once all classes are loaded, the Corticon .NET Server starts up in the IIS, which is needed by our simple SOAP client class.

Figure 6: Lower Portion of the .NET Server remote API console



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:

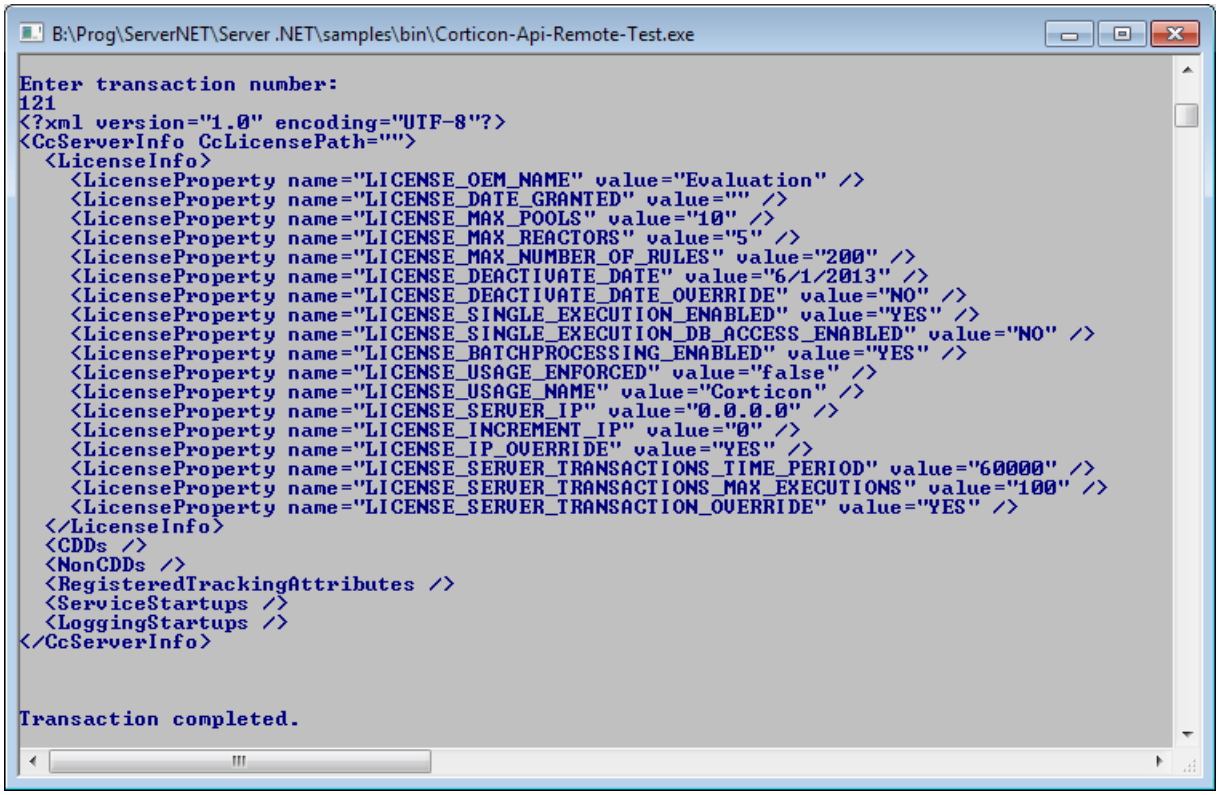
- Enter `200` to list the Decision Service Functions command set
- Enter `300` to list the Monitoring Functions command set
- Enter `400` to list the CcServer Functions command set
- Enter `100` to again list the Common Functions command set

Note: After you enter a transaction, the result is displayed followed 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 service inside our web server. A good administrative method to call is transaction #121, **Get CcServer current info**. This choice corresponds directly to the API method `getCcServerInfo()`.

To try this, confirm that IIS is running, and then enter `121` in the command window. The `CcServerAxisTest` class makes a call to the Corticon Server 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 figure:

Figure 7: .NET Server remote API console response to command 121



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.

8

Deploying a Ruleflow to the Corticon Server for .NET

Just because Corticon Server for .NET has been installed 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 two ways:

- **Deployment Descriptor files** - This is the easiest method and the one we will use in this Tutorial because it is also the method typically used in production web service deployments.
- **.NET APIs** - This method requires more knowledge of the Server for .NET API set, and is not discussed in this Tutorial.

Both 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

For purposes of this Tutorial, we assume you have already created a Ruleflow suitable for deployment. If you have completed the *Corticon Tutorial: Basic Rule Modeling* , then you have indeed created a sample Ruleflow that is ready for deployment to the Server for .NET.

In the *Corticon Tutorial: Basic Rule Modeling* we built and tested a new Ruleflow from scratch. We will use that Ruleflow here, but the same steps we outline must be followed regardless of the Ruleflow we use.

If you no longer have your original Ruleflow, then use `tutorial_example.erf` located in `[CORTICON_HOME]\Samples\Rule Projects\Tutorial\Tutorial-Done` as a substitute. The rules inside are essentially the same as those built in the first part of the *Corticon Tutorial: Basic Rule Modeling* .

Creating and installing a Deployment Descriptor file

A Deployment Descriptor file tells the Corticon Server for .NET 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 – it is important that this path **not** contain space characters. For example, a Ruleflow stored in `My Documents` cannot be referenced by a Deployment Descriptor file because its path contains a space. Even though the default storage location for your Ruleflow files is inside a Corticon Studio installation's `[CORTICON_WORK_DIR]\Samples\Rule Projects\Tutorial\Tutorial-Done` (which contains a space), we avoid the problem by substituting `../../` as a relative reference to the directory structure.

Deployment Descriptors are easily created using the Deployment Console, which is installed by the Server installer.

Using the Deployment Console tool's Decision Services on the .NET Server

To start the Corticon Deployment Console for .NET, choose the Windows **Start** menu command **All Programs** > **Progress** > **Corticon 5.4** > **Corticon .NET Deployment Console** to launch the executable file `Server .NET\samples\bin\DeploymentConsole.exe`.

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

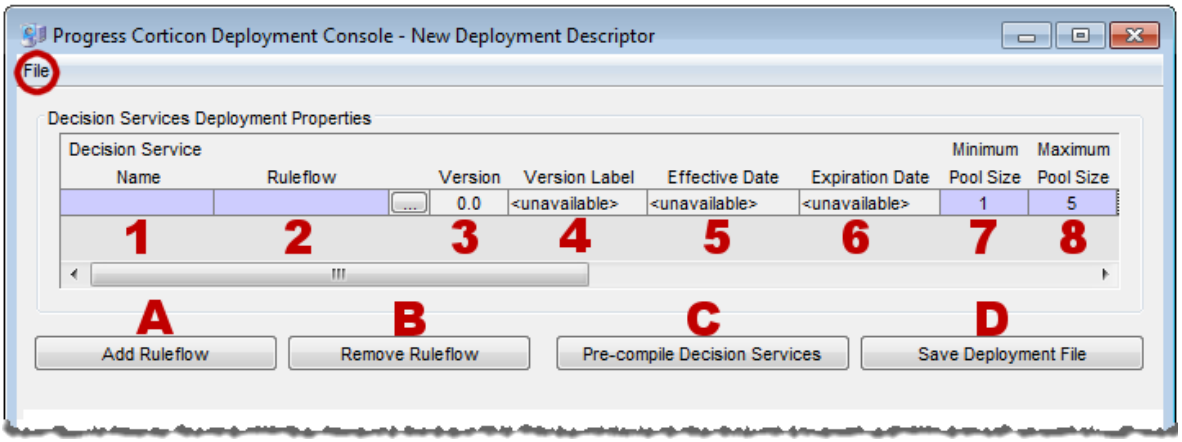
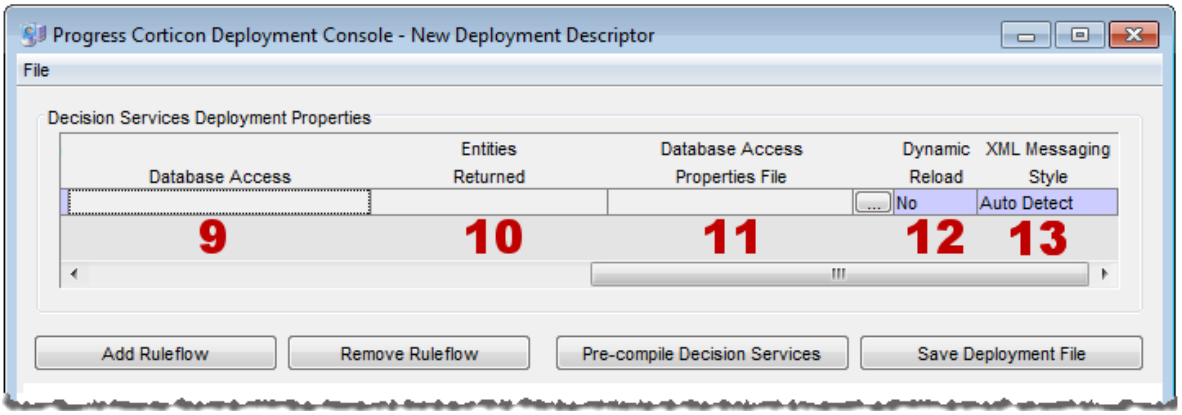


Figure 9: Right Portion of Deployment Console, with Deployment Descriptor File Settings Numbered



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 CcServer.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 for .NET has been installed and deployed on IIS, the following sequence occurs:

1. IIS server starts.
2. Corticon Server for .NET starts as a web service in IIS.
3. Corticon Server looks for Deployment Descriptor files in the `<IISRoot>\axis\cdd` directory.
4. Corticon Server for .NET 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 callable by external applications and clients.

In order for the Corticon Server for .NET to find Deployment Descriptor files when it looks in step 3, we must ensure that the `.cdd` files are stored in the default location, the `[CORTICON_WORK_DIR]\cdd` directory. When creating `.cdd` files, save them to this directory so they become accessible to the deployed Corticon Server for .NET.

Note: This location is configurable, but be aware that Deployment Descriptor files usually contain paths relative to where they were created; as such, copying or moving them to a different location can make the file behave incorrectly. See the Deploying Corticon Ruleflows chapter of the *Server Integration & Deployment Guide* for details.

Now, when the startup sequence reaches step 3 above, the server knows 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 IIS. A maintenance thread in the Corticon Server for .NET watches for additions, deletions, and changes and updates appropriately. A Ruleflow can be modified in Studio even while it is also simultaneously deployed as a Decision Service and involved in a transaction - Server can be configured to update the Decision Service dynamically for the very next transaction.

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.

9

Consuming a Decision Service

Let's review what we have accomplished so far:

1. We have installed Corticon Server for .NET files onto a workstation or server .
2. We have configured Corticon Server for .NET as a web service onto IIS.
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 for .NET 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 C# sample code to consume a Decision Service
- Path 3 - Using SOAP client to consume a Decision Service
- Limits of the default evaluation license
- 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 three paths for consuming a Decision Service:

- Path 1

  Use Progress 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 Basic Tutorial* 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 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 and display the response message. Corticon does not include such an application, so you must obtain one in order to complete this path.

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.

Configuring Studio to send a SOAP Message to IIS

Corticon Studio is configured by default to query a `localhost` web server on port `8850`. Because we are using IIS, we'll change the port used by Studio to send Test messages.

Note: Instead of `localhost`, you can use the static IP or DNS-resolvable name of the host -- a good idea as it emulates actual deployment.

To configure the port:

1. Navigate to the directory `[CORTICON_WORK_DIR]`.
2. Edit the file `brms.properties` in that location.

   Note: If you specified a preferred name and location of the override properties file in Studio preferences, edit that file as it the one that will be the last loaded.

   ---

3. Add the following line to the file so that your IIS points to its server port:

   ```
   com.corticon.deployment.soapbindingurl_2=http://localhost:80/axis
   ```
4. Save the edited file.
5. Restart Corticon Studio.

The edited value is added to the list of Remote Servers.

Creating a new .NET 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:

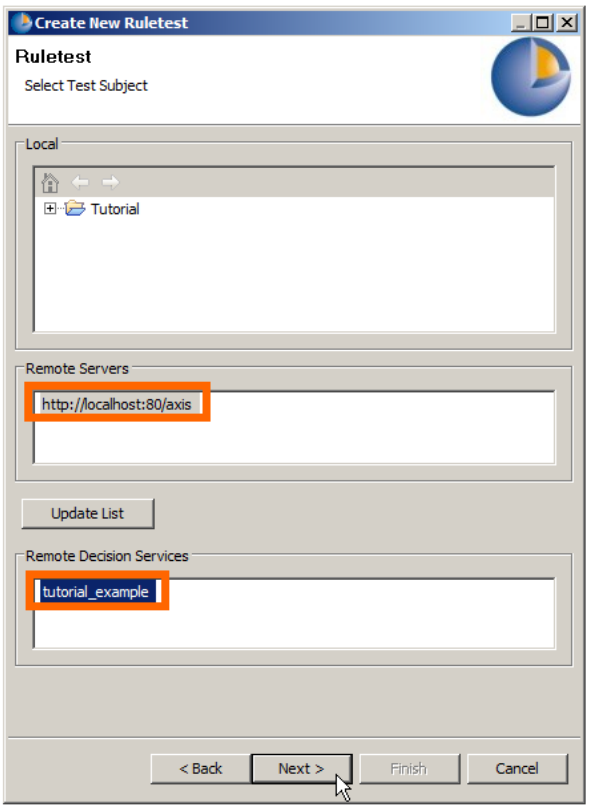
1. You will be asked to **Select Test Subject**. Be sure to select the http://localhost:80/axis in the **Remote Servers** box.
2. Select **Update List**. Corticon Studio will attempt to contact a Corticon Server instance at the location specified above. If a Corticon Server instance is running, it will respond with a list of available Decision Services, and display that list in the **Remote Decision Services** window.
3. Choose the Decision Service to invoke. In this Tutorial, we want `tutorial_example`.
4. Click **Next>**
5. 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.

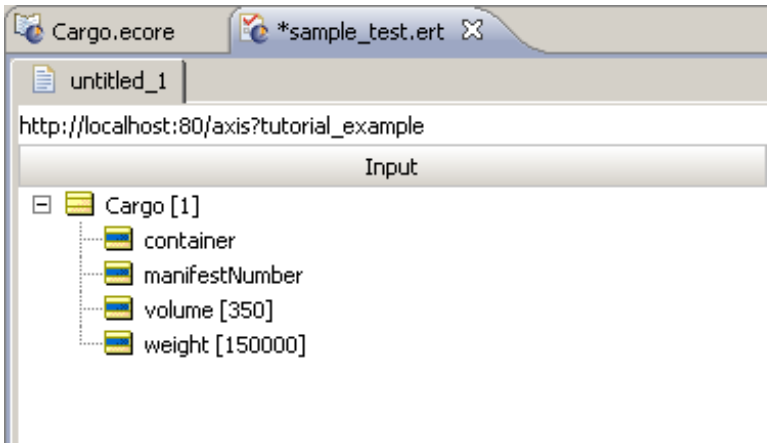
In step 1, you selected the default URL: Corticon Server running on `localhost`. If you want to change the URL to another address, see "Designer properties & settings" in *Server Integration & Deployment Guide* for more information about configuring Corticon Studio properties.

Figure 10: Requesting List of Remote Decision Services



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

Figure 11: Sample Data in a Studio Remote Ruletest

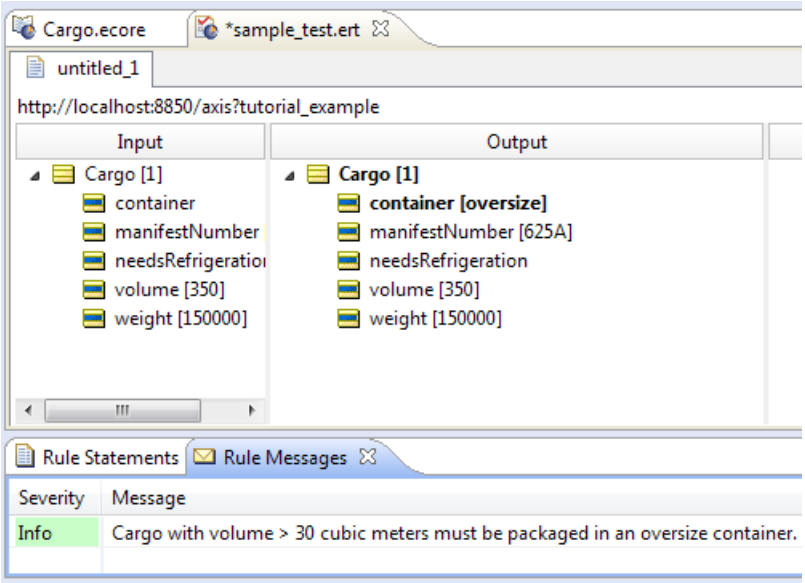


Executing the remote test

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 12: 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 C# sample code to consume a Decision Service

To use this path, you should have solid .NET programming skills and familiarity with the .NET Framework SDK environment. This portion of the Tutorial does not provide in-depth explanations of working within the .NET environment.

Sample web service client code is provided in `[CORTICON_HOME]Server .NET\samples\webservice-client`. This sample includes the following files:

- `Cargo_FlightPlan.wsdl` - WSDL generated by the Deployment Console
- `CargoDecisionProxy_ASPNET.cs` - C# web service proxy generated by `wsdl.exe`
- `CallCargoService.cs` - C# code demonstrating how to call the web service
- `GenProxy.bat` - Code to generate the decision service proxy from the WSDL

Creating the WSDL and proxy files

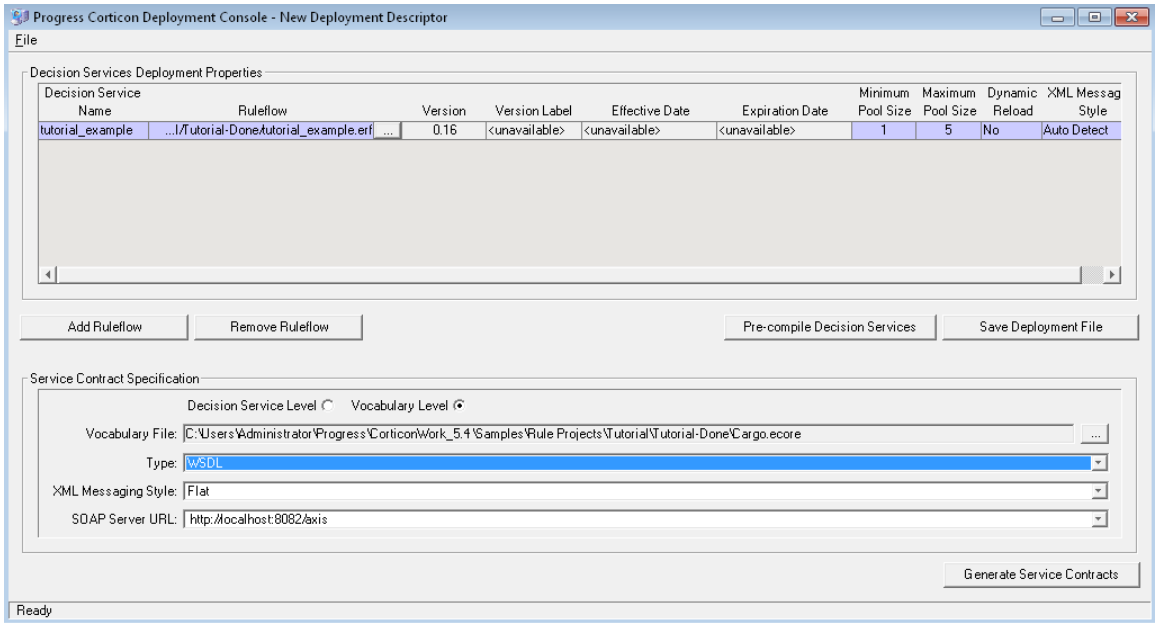
The WSDL and the proxy files are created as follows:

1. If your .NET Server and Studio are colocated, you have the Tutorial Ruleflow in `[CORTICON_HOME]\samples\Tutorial\Tutorial-Done\tutorial_example.erf`

   If your .NET Server and Studio are on separate machines, copy and stage that file so that it can be accessed on the .NET server machine.
2. Launch the Deployment Console on the Corticon Server .NET machine by choosing the **Start** menu command **All Programs > Progress > Corticon 5.4 > Corticon .NET Deployment Console**
3. Click the ... button to the right of the Ruleflow on the one empty line listed, and the locate the `tutorial_example.erf` file.
4. In the lower section, click the **Type** dropdown, and then choose `WSDL`.

   The window should now look like this:

   Figure 13: Creating a new WSDL using the Deployment Console



5. Click **Generate Service Contracts** to save the service contract file, which is named `Cargo_Cargo.wsdl`. It may be convenient to generate this file into a separate directory. Here, we use directory `[CORTICON_WORK_DIR]`.

Note: To generate a web service proxy, you need `wsdl.exe`. When you run `wsdl.exe Cargo_Cargo.wsdl`, the file `CargoDecisionService.cs` is created. Place that file in the .NET Server's `[CORTICON_HOME]`. Refer to the `GenProxy.bat` file located at `[CORTICON_HOME_\Server .NET\samples\wcf-client` for the WSDL options, typically `/namespace:` and `/out:`.

---

6. Write C# client code to call the web service. We provide a sample in `CallCargoService.cs`, which sets values of attributes used in the rules.

7. Compile `CargoDecisionService.cs` and `CallCargoService.cs` using the `csc *.cs` command. Generally, the compile process needs to occur in your .NET Framework root directory, so you may need to move both C# files to that directory prior to compilation. In our case, the .NET Framework is installed at `C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319`

   This generates an executable file named `CallCargoService-webservice.exe`. Store the file in your `[CORTICON_WORK_DIR]`.
8. If you have not already done so, deploy the `tutorial_example` Decision Service to Corticon Server for .NET on IIS. Follow the instructions for Creating and Installing a Deployment Descriptor File.
9. Run `CallCargoService-webservice.exe` to execute the call to Corticon Server. You will see the following output:

   Figure 14: Invoking Corticon Server for .NET via C# Sample Code

```
Calling service tutorial_example
---------------------------------------
Response
----------------------
WorkDocumentsType
Cargo weighing between 150,000 and 200,000 pounds must be packaged in
a container
```

Path 3 - Using SOAP client to consume a Decision Service

Web Services Service Contracts

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 much 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 messaging styles

There are also two types of 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. Some 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. Corticon Server for .NET's `execute` method is most commonly invoked through document-style messaging because of its ability to work with a complex data payload.

Important: Any SOAP client or SOAP-capable application used to consume a Decision Service deployed to the Corticon Server must use 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

Launch the **Deployment Console** as before and follow the instructions below to generate a service contract. All **Deployment Console** options below are also described in more detail in the *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 Java classes. Note, no output is produced when Java classes is selected because there is no standard method for describing service contracts in the Java world.
4. **Output directory**. The location where you want the Deployment Console to save this service contract.
5. **XML Messaging Style**. Enabled only for Vocabulary-level service contracts. Describes the message style, flat or hierarchical, in which the WSDL will be structured.
6. **SOAP Server URL**. URL for the SOAP node that is bound to the Corticon Server. Enabled for WSDL service contracts only. The default URLs http://localhost:8850/axis/services/Corticon and https://localhost:8851/axis/services/Corticon make a Decision Service available to the default Corticon Server installation performed earlier. Note: These URLs can be changed and additional URLs can be added to the drop-down list.
7. **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 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 Corticon Server will not know which Decision Service is being requested, and will return an error message.

Enter all required data into the request message. The `tutorial_example.erf` example requires the following data:

Vocabulary Term	Possible Values
`Cargo.weight`	A number less than or equal to `200,000`
`Cargo.volume`	Any real number

Sending a request message to the server

Make sure IIS is running and your Deployment Descriptor file is installed in the correct location as described earlier. Now, use your SOAP tool to send the request message to the Server for .NET.

Your SOAP tool should display the response from the Server for .NET. Are the results what you expected? If not, or if the response contains an error, proceed to the Troubleshooting section of this tutorial.

Limits of the default evaluation license

The license included in the default Corticon Server for .NET installation has preset limits on some Corticon Server and Decision Service parameters. These limits are:

- **Number of Decision Services** – Up to five 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 five.
- **Pool Size** – No Decision Service may have a maximum pool size setting of greater than five. Pool size is measured on a Decision Service-by-Decision Service basis, so you may have up to 5 Decision Services deployed (compliant with the Decision Service limitation above), each with up to five Reactors in its pool, without violating the default license restrictions.
- **Number of Rules** – All rules in all deployed Ruleflows (in other words, all deployed Decision Services) must not exceed 200. 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 will capture errors and exceptions caused by expired or "under-strength" licenses. These log messages are detailed in the *Troubleshooting* section of the *Rule Modeling Guide*.

If you are a 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 Progress Corticon support or your Progress representative for a license with expanded capabilities.

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.

10

Using .NET Business Objects as payload for Decision Services

Introduction

Microsoft .NET Classes provide applications with reusable, portable code. Classes are logical sections of an application. For instance, the call to a database and retrieval of table data is part of a data class. These classes can be used in other sections of the application, or can be used it in an entirely different software design.

Class properties that allow other areas of code to interface with the class are usually created with "get" and "set" keywords.

Corticon Server for .NET can communicate with .NET classes through a Java Object Messaging (JOM) interface. This document illustrates how a client program can instantiate a .NET class and then use that object instance to communicate with Corticon for .NET Server. In other words, this enables Corticon Server for .NET to natively bind to the application's .NET class definitions during rules execution.

To use the JOM interface from any C# client application, it is necessary to create Java stubs that match the C# object model. These stubs will enable Corticon Studio Vocabulary Editor to import the class metadata for mapping purposes; in other words, to pull in the names of the `get` and `set` methods needed. The stubs are also used to generate import statements in the compiled rules.

At deployment time, the Corticon for .NET Server will dynamically transform the Rule asset from Java byte code into .NET Common Intermediate Language (CIL), the object-oriented assembly language for .NET runtime environments. When this is done, the Java code stub references inside the stub library (a `.jar` file) are translated into references to the actual C# class file library (a `.dll` file). Finally, the translated Rule asset (CIL) will directly call get/set methods on the C# object during Rule asset execution.

IKVM.NET

To create Java Stubs from .NET class files, Corticon supplies an open source utility, IKVM.NET, which is an essential component of the Corticon Server for .NET runtime architecture enabling it to seamlessly operate in .NET environments. IKVM.NET is included in your Corticon 5.4 installation.

Note: For more information about IKVM.NET, see their web site, http://www.ikvm.net.

IKVM.NET is an implementation of Java for Mono and the Microsoft .NET Framework. It includes the following components:

- A Java Virtual Machine implemented in .NET
- A .NET implementation of the Java class libraries
- Tools that enable Java and .NET interoperability

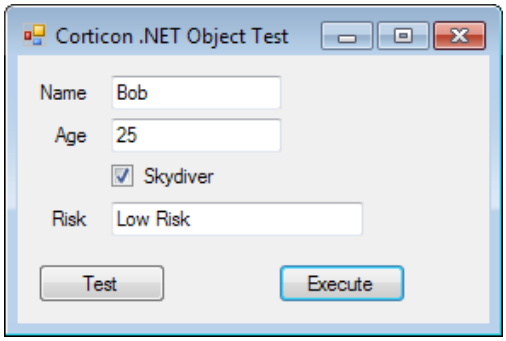
Running the sample Java Object Messaging (JOM) client application

The sample JOM Client uses C# objects to communicate with Corticon Server for .NET (in- process) via the object messaging interface.

Run this sample program by launching `[CORTICON_HOME]\Server .NET\samples\bin\JomClient.exe`)

When the application launches, the window lets you enter parameter values for executing this decision service:

- Name
- Age
- Is this person a skydiver?

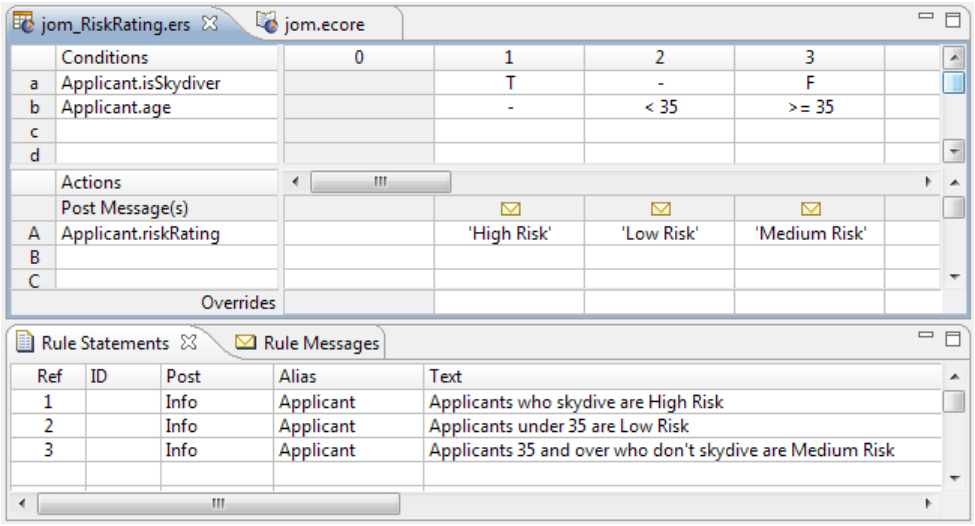


Enter values, and the click **Execute** to see the decision.

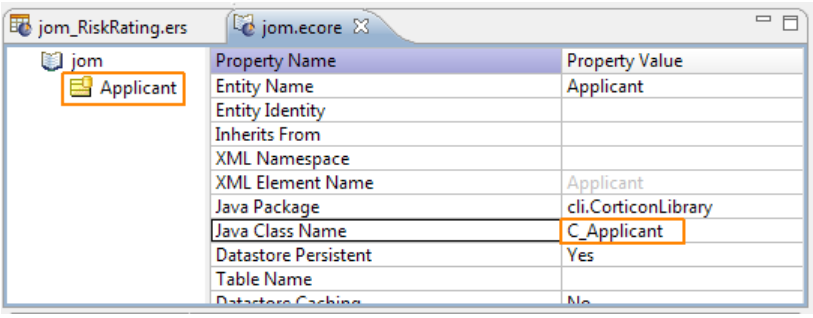
You might wonder how this application was created and how it works. Let's examine its building blocks. When you understand the solution architecture, it will be easy to create your own examples and solutions.

Examining the sample

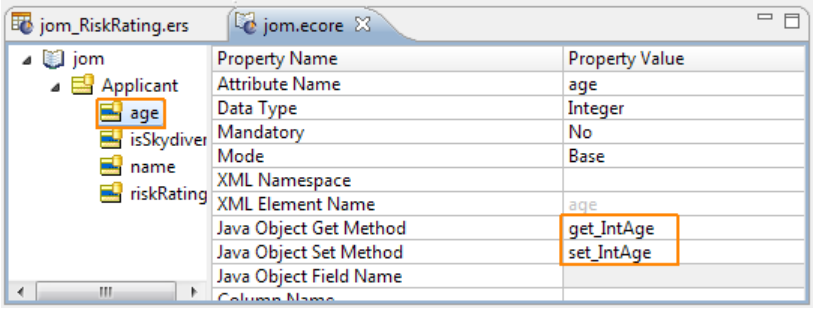
The sample `JomClient.exe` uses a Rulesheet. Open `[CORTICON_NET_WORK_DIR]\Samples\rule-assets\JOM\Rules\jom.RiskRating.ers`:



Open the Vocabulary `[CORTICON_NET_WORK_DIR]\Samples\rule-assets\JOM\Vocab\jom.ecore`. When your **Preferences** are set to **Integration and Deployment**, the `Applicant` Entity shows that it is bound to the Java Package `cli.CorticonLibrary`, and its Java Class name refers to the `C_Applicant` class.



Each Attribute within the Entity refers to a **Java Object Get Method** and a **Java Object Set Method** method, as shown:



So how do we get the `CorticonLibrary` that contains these Java classes exposed in the Corticon Vocabulary editor? Let's see.

Invoking Corticon Server for .NET

`JomClient.exe` is written in C#. Logic that invokes the in-process server is located for the sample at `[CORTICON_NET_HOME]\samples\api-client\jom\JomClientForm.cs`, as shown:

```
private void btnExecute_Click(object sender, EventArgs e)
{
    C_Applicant lC_Applicant = new C_Applicant();
    lC_Applicant.StrName = txtName.Text;
    lC_Applicant.IntAge = Convert.ToInt32(txtAge.Text);
    lC_Applicant.BlnSkydiver = chkSkydiver.Checked;

    ArrayList llistObjects = new ArrayList();
    llistObjects.add(lC_Applicant);
    iCcServer.execute(DECISION_SERIVCE_NAME, llistObjects);
    C_Applicant lC_ApplicantReturned = (C_Applicant)llistObjects.get(0);

    txtRating.Text = lC_ApplicantReturned.StrRiskRating;
}
```

Running the sample JOM application

Now when you launch `JOMClient.exe`, Corticon.NET Server will try to deploy the JOM Decision Service. As described above, the JOM `.ecore` has already imported the Java Stub Class Metadata. That's important for the next step. The JOM Client will first compile the `[CORTICON_NET_WORK_DIR]\samples\rule-assets\JOM\jom.erf` into a deployable `.eds` file. For the compilation to be successful, the `CorticonLibrary.jar` must be in the `\lib` directory. During deployment of the `.eds` file, the IKVM loader will convert the Java byte code into CLI and also load the C# class definitions in `CorticonLibrary.dll`, effectively allowing `JomClent.exe` to use C# object instances to communicate with Corticon Server for .NET.

Preparing the C# Class files

We want the client program to instantiate `C_Applicant`, and then use that object instance to communicate with Corticon for .NET Server. First we'll need a class file we are using in our application to expose all data objects. For that, we will use `[CORTICON_NET_HOME]\samples\api-client\CorticonLibrary.cs`, as shown:

```
using System;
namespace CorticonLibrary
{
    public class C_Applicant
    {
        private String strName;
        private Boolean isMarried;
        private Int32 intAge;
        private Boolean blnSkydiver;
        private String strRiskRating;
        public String strState;

        public String StrName
        {
            get { return strName; }
            set { strName = value; }
        }
        public Int32 IntAge
        {
            get { return intAge; }
            set { intAge = value; }
        }
        public Boolean BlnSkydiver
        {
            get { return blnSkydiver; }
            set { blnSkydiver = value; }
        }
        public String StrRiskRating
        {
            get { return strRiskRating; }
            set { strRiskRating = value; }
        }
        public Boolean IsMarried
        {
            get { return isMarried; }
            set { isMarried = value; }
        }
    }
}
```

Note: Having a namespace in the `CorticonLibrary.cs` file is mandatory.

Compile the `CorticonLibrary.cs` to create the `CorticonLibrary.dll`. To use the JOM interface from any C# client, it is necessary to create Java stubs that match your C# object model. For the JOM Client example, we used Visual Studio to compile the C# class `C_Applicant` into `CorticonLibrary.dll`.

Save `CorticonLibrary.dll` to both `[CORTICON_NET_HOME]\samples\gen stubs\` and `[CORTICON_NET_HOME]\samples\bin\`

Generating the Java Stubs

We can generate the Java stub file from `CorticonLibrary.dll` using the utility script `[CORTICON_NET_HOME]\samples\api-client\GenStubs.bat`:

```
*******
@echo off
SET IKVM_HOME=..\bin
SET INPUT_DLL_NAME=CorticonLibrary.dll
SET OUTPUT_JAR_NAME=CorticonLibrary.jar
ATTRIB -R %OUTPUT_JAR_NAME%
"%IKVM_HOME%\ikvmstub.exe" %INPUT_DLL_NAME%
ECHO Successfully completed: C# types in %INPUT_DLL_NAME% were converted into
 stubs in %OUTPUT_JAR_NAME%.
Pause
*******
```

The `GenStubs.bat` utility will generate the JAR `CorticonLibrary.jar`.

Add `CorticonLibrary.dll` to `[CORTICON_NET_HOME]\samples\bin\`

Add `CorticonLibrary.jar` to `[CORTICON_NET_HOME]\samples\lib\`.

Note: About putting the `CorticonLibrary.jar` in the `samples\lib` directory - It is important to understand where to locate the JAR file, especially when you are outside of Corticon installation directories. The JAR that the Corticon .NET Configuration (inside `CorticonShared.dll`) directs IKVM to load all JAR files that are located in the `[CORTICON_NET_HOME]\lib` directory. In the case of the `JomClient.exe`, the `JomClient.exe` defines `[CORTICON_NET_HOME]` equal to `[CORTICON_NET_HOME]\samples`. Where is this done? Each .NET Application has a `.exe.config` file where the user defines their Corticon Home and Work directories. For example `bin\JomClient.exe` has a `JomClient.exe.config`. Within that file is the following section:

```
<configuration>
 <appSettings>
  <add key="CORTICON_HOME" value=".." />
  <add key="CORTICON_WORK_DIR"
          value="C:\Users\{user}\Progress\CorticonWork_5.4 .Net"/>
 </appSettings>
</configuration>
```

That defines `CORTICON_HOME` as located up one directory level from where the `.exe` is currently located. Typically, the result is `C:\Program Files\Progress\Corticon 5.4 .Net\Server .NET\samples`. Then the `CorticonConfiguration` changes `CORTICON_HOME` by adding "`\lib`" which locates it in that directory where it loads all the JARs.

Supported .NET datatypes for Attributes

Corticon Attributes support .NET Business Objects with nullable datatypes. Using a trailing **?** character converts the Datatype into a Nullable Datatype of the same type.

```
Boolean
bool
Boolean?
bool?

Byte
byte
Byte?
byte?

Char
char
Char?
char?

DateTime
DateTime?

Decimal
decimal
Decimal?
decimal?

Double
double
Double?
double?

float
float?

Int16
Int16?
```

```
Int32
Int32?

Int64
Int64?

int
int?

long
long?

short
short?

String
String?
```

Supported .NET datatypes for Associations

Corticon Associations support .NET Business Objects.

C# Association Datatypes:

```
System.Collections.ArrayList
System.Collections.IList
<Business Object Name>[]  (Array of Business Objects)
```

IKVM Open JDK Datatypes:

```
java.util.ArrayList
java.util.Vector
java.util.HashSet
java.util.Collection
java.util.List
java.util.Set
```

All datatypes that are supported are demonstrated in the `CorticonLibrary.cs` file under the `ObjectA` and `ObjectB` objects.

Using GenStubs.bat for your .NET Business Objects

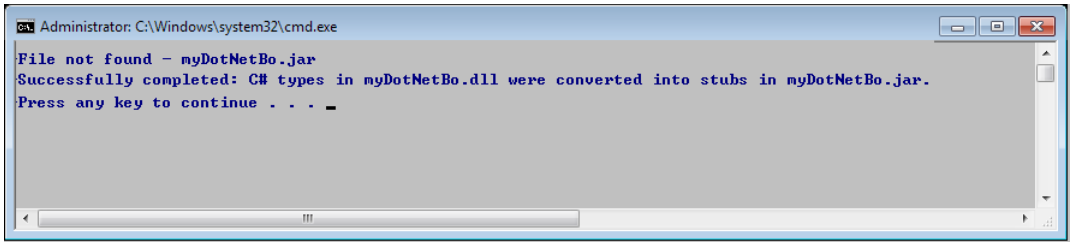
To generate stubs:

1. Compile the .NET Business Objects in `.cs` or `.vb` to create a `.dll` file. For example, use `myDotNetBo.cs` or `myDotNetBo.vb.` to compile `myDotNetBo.dll`.
2. Copy the .NET Business Object `.dll` to the `[CORTICON_NET_HOME]\samples\gen stubs\` directory.
3. Locate a text editor at `[CORTICON_NET_HOME]\samples\gen stubs\` to edit `GenStubs.bat`.
4. Modify the following two properties to match the `.dll` name and the `.jar` name, as shown here for `myDotNetBo`:

   ```
   SET INPUT_DLL_NAME= myDotNetBo.dll
   SET OUTPUT_JAR_NAME= myDotNetBo.jar
   ```

5. Save the file under an appropriate name(such as `GenStubs_MyBo.bat`).
6. Run your `GenStubs` script file to generate the Java stub JAR file.

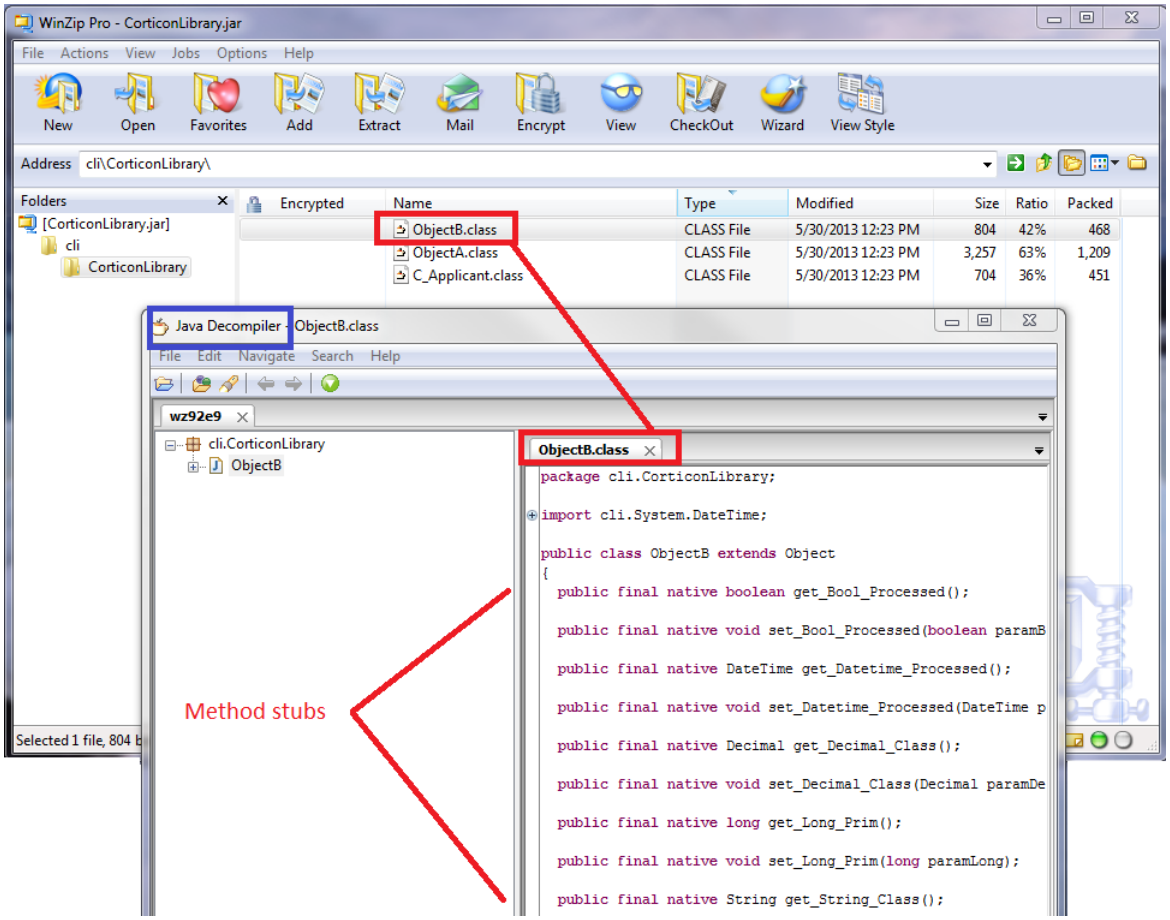
Note: The first time you do this, you get a warning message about not finding `myDotNetBo.jar`:



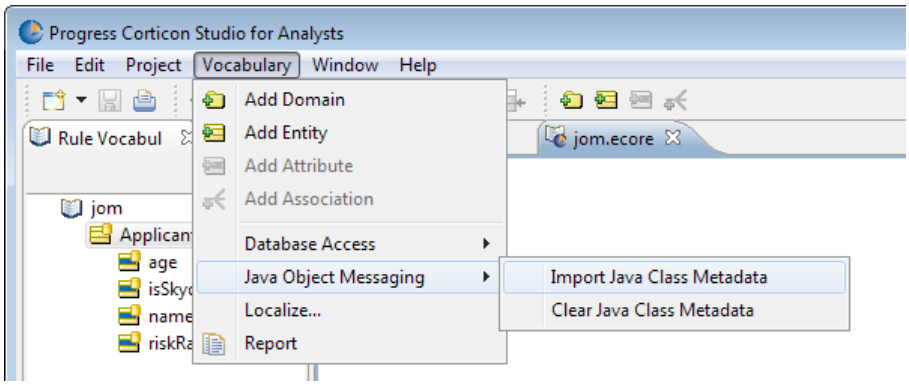
Ignore this -- it is trying to delete it first, but it isn't there.

The Java stub JAR file you created is saved at `[CORTICON_NET_HOME]\samples\gen stubs\`.

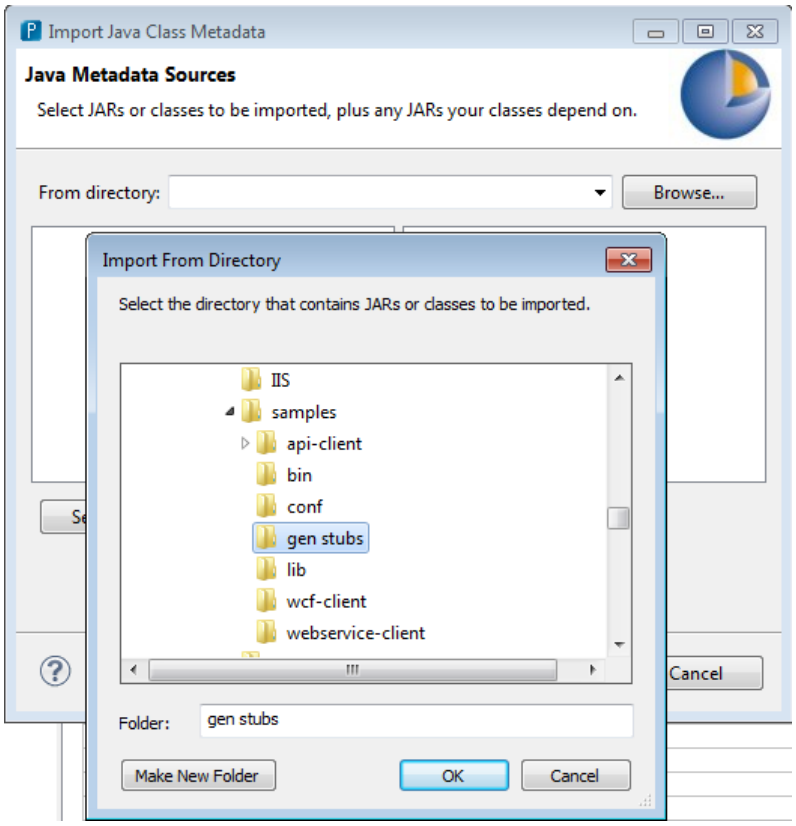
7. Copy your JAR (for example, `myDotNetBo.jar`) to `[CORTICON_NET_HOME]\samples\lib`.
8. Verify the creation of the JAR file. The following example illustrates a stub class in the `CorticonLibrary.jar` through a Java Decompiler:



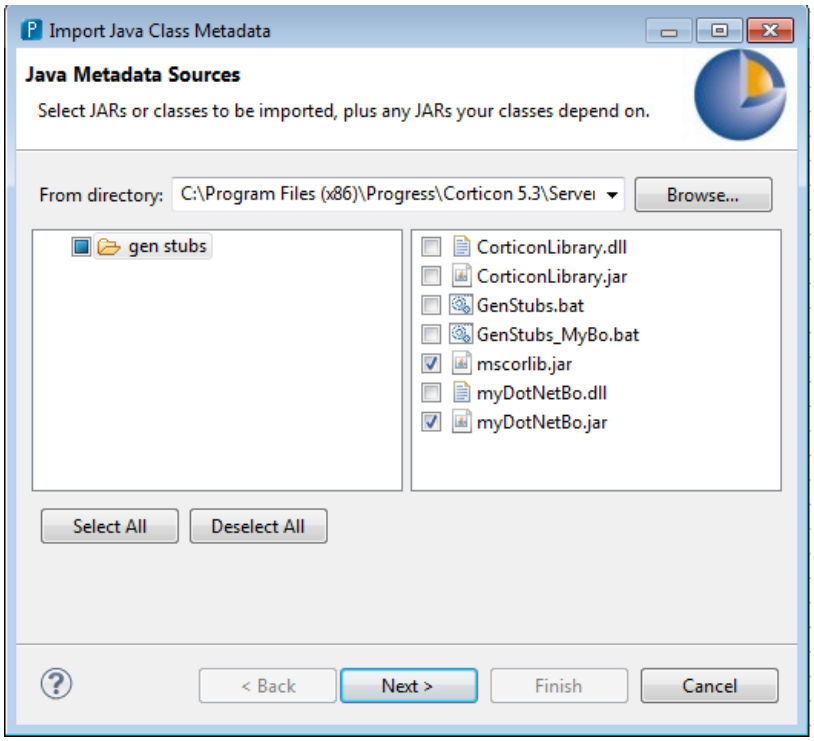
9. Open the Vocabulary in Corticon Studio, and then chose the menu command **Vocabulary > Java Object Messaging > Import Java Class Metadata**, as shown:



10. Browse to `[CORTICON_NET_HOME]\samples\gen stubs\`:

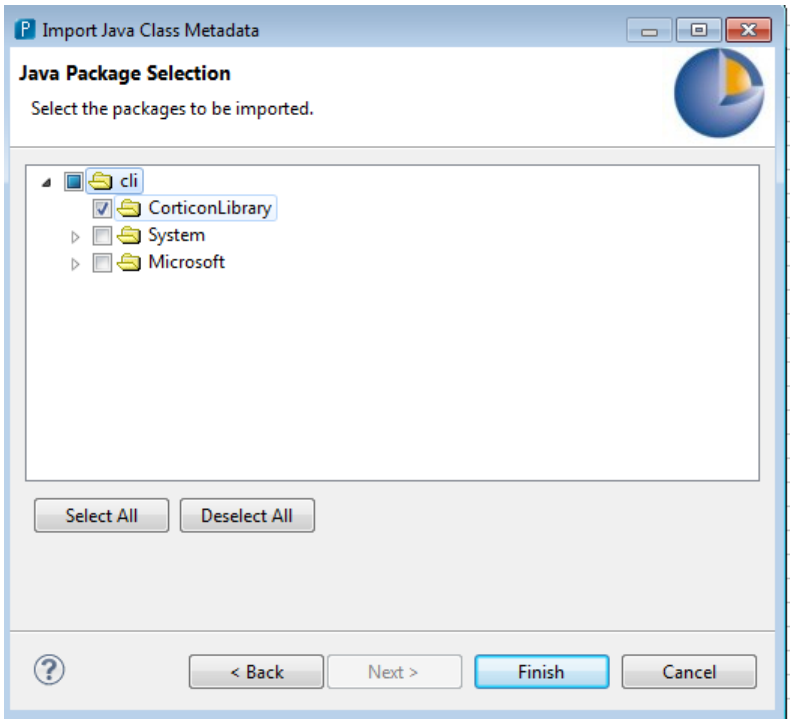


11. Select the JAR that you want imported into Corticon Studio, as well as `mscorlib.jar` that is also in the `gen stubs` directory. The `mscorlib.jar` needs to also be selected because the `.jar` file that was created through the `gen stubs` process depends on the classes inside the `mscorlib.jar`.

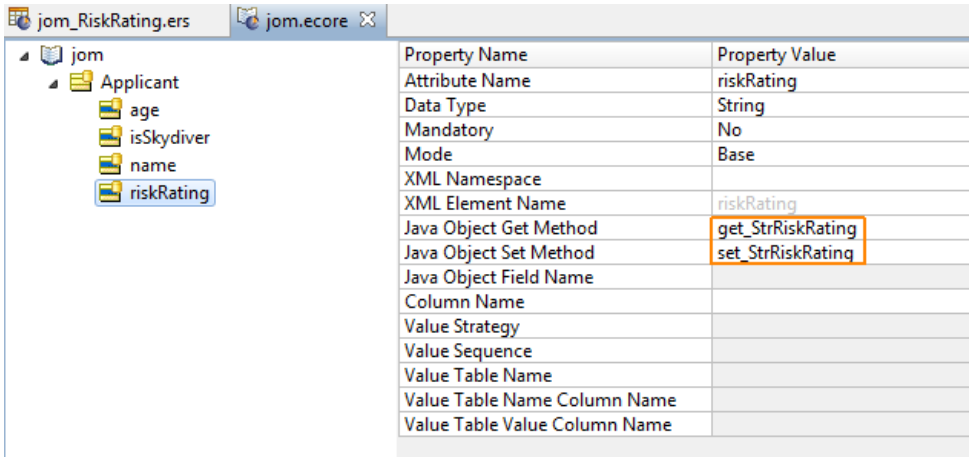


12. Select the packages that are associated with the Java Stub Classes that were created in the `gen stubs` process. In our example, the `CorticonLibrary.jar` contains classes under the `cli.CorticonLibrary` package.

Note: As noted earlier, Business Objects require a defined Namespace. If there is no defined Namespace, the Java Stub Classes will have a default package of only `cli`. That would force you to choose the `cli` checkbox, and import **ALL** the Java Class Metadata in the `CorticonLibrary.jar` and the `mscorlib.jar`, an unwarranted scope.



13. After import is complete, verify that the **Java Object Getter/Setter Methods** have been assigned to each of the Attributes and Associations.

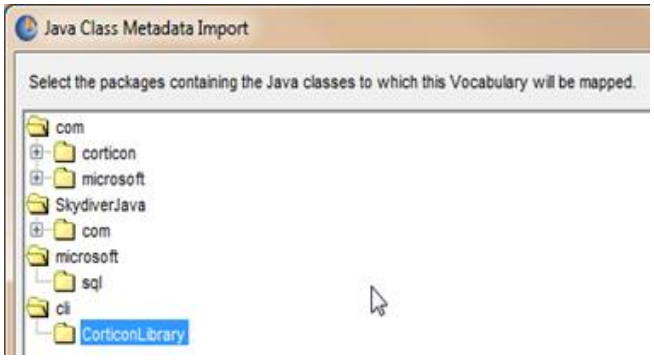


The Vocabulary tries to ***SmartMatch*** the Vocabulary Attribute and Association Names to match a Getter/Setter name in the imported Java Metadata. The ***SmartMatch*** looks for an appropriate Getter/Setter Method that begins with `get` or `set` as in `get<AttributeName>` or `get_<AttributeName>`. If this occurs, the Getter/Setter value inside the Vocabulary is colored light grey, indicating that it was determined through a ***SmartMatch*** lookup.

However, in the example above, ***SmartMatch*** is unable to match the Getter/Setter with Attribute `riskRating` because the appropriate Java Class Method does not conform to the ***SmartMatch*** algorithm:

- Vocabulary Attribute: `riskRating`
- Smart Match looks for: `getRiskRating()` or `get_RiskRating()`
- Actual Java Method: `get_StrRiskRating()`

14. Since the ***SmartMatch*** failed to locate an appropriate Getter/Setter, you need to select the appropriate Getter/Setter Method from the drop-down for that Attribute or Association. Since the user defined their own mappings, the Getter/Setter method value is colored black (as illustrated).
15. After deploying the rules to IIS server, ensure that the Business Objects are picked up by copying:
    - `myDotNetBo.jar` to `C:\inetpub\wwwroot\axis\lib`
    - `myDotNetBo.dll` to `C:\inetpub\wwwroot\axis\bin`
16. When the package has been created, it will look something like this:



Testing outside IIS

If you are testing your work outside of IIS, such as in a Visual Studio Environment, you to perform some extra tasks:

1. Copy the `myDotNetBO.dll` to a location where it can be referenced. If running in IIS and the `axis` application has been deployed, the `.dll` needs to be moved to the `c:\inetpub\wwwroot\axis\bin` directory.
2. Copy the `myDotNetBo.jar` into the running application's `[CORTICON_NET_HOME]\lib` directory.

The JAR is needed during compilation of the Ruleflow (`.erf`) into an Enterprise Decision Service (`.eds`) file. The compilation step depends on the `.jar` file being in the `[CORTICON_NET_HOME]\lib` directory.

Note: About putting the JAR in the `\lib` directory when Business Objects are used in IIS

The `CorticonConfiguration` looks for `CORTICON_HOME\lib`. With IIS, `CORTICON_HOME` *could be* defined or overridden in the `web.config` file under `[IIS]\axis`.

Parameters can be set in Corticon in the `<appSettings>` section. Note there are no `CORTICON_HOME` or `CORTICON_WORK_DIR` settings. That means that the “current directory” is the `CORTICON_HOME` and `CORTICON_WORK_DIR`. For IIS, put the JAR file in `C:\inetpub\wwwroot\axis\lib`.

11

Support for Windows Communication Framework (WCF)

To use this approach, you should have solid .NET programming skills and familiarity with the .NET WCF Framework. This portion of the Tutorial does not provide in-depth explanations of working within the .NET WCF environment.

Sample web service client code is provided in `[CORTICON_HOME]Server .NET\samples\wcf-client`. This sample includes the following files:

- `Cargo_FlightPlan.wsdl` - WSDL generated by the Deployment Console
- `CargoDecisionProxy_WCF.cs` - C# web service proxy generated by `svcutil.exe`
- `CallCargoService.cs` - C# code demonstrating how to call the web service
- `GenProxy.bat` - Code to generate the decision service proxy from the WSDL
- `App.config` - The configuration file for the decision service endpoint

For details, see the following topics:

- Creating WSDL and proxy files

Creating WSDL and proxy files

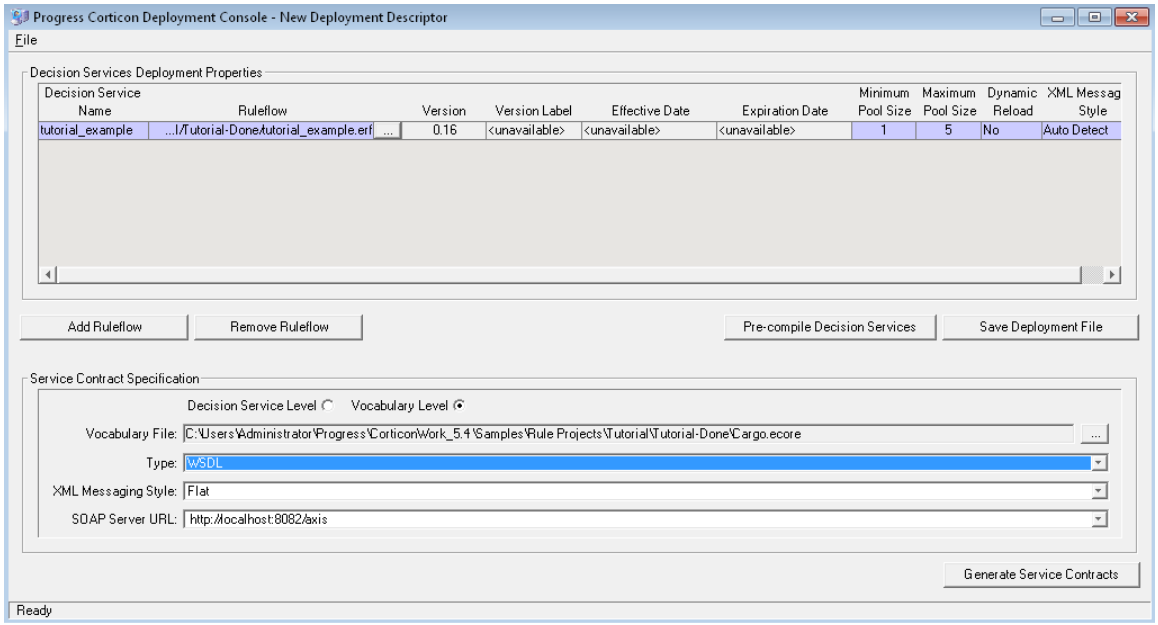
The WSDL and the proxy files are created as follows:

1. If your .NET Server and Studio are colocated, you have the Tutorial Ruleflow in `[CORTICON_HOME]\samples\Tutorial\Tutorial-Done\tutorial_example.erf`

   If your .NET Server and Studio are on separate machines, copy and stage that file so that it can be accessed on the .NET server machine.
2. Launch the Deployment Console on the Corticon Server .NET machine by choosing the **Start** menu command **All Programs > Progress > Corticon 5.4 > Corticon .NET Deployment Console**
3. Click the ... button to the right of the Ruleflow on the one empty line listed, and the locate the `tutorial_example.erf` file.
4. In the lower section, click the **Type** dropdown, and then choose `WSDL`.

   The window should now look like this:

   Figure 15: Creating a new WSDL using the Deployment Console



5. Click **Generate Service Contracts** to save the service contract file, which is named `Cargo_Cargo.wsdl`. It may be convenient to generate this file into a separate directory. Here, we use directory `[CORTICON_WORK_DIR]`.

Note: To generate a web service proxy, you need `wsdl.exe`. When you run `wsdl.exe Cargo_Cargo.wsdl`, the file `CargoDecisionService.cs` is created. Place that file in the .NET Server's `[CORTICON_HOME]`. Refer to the `GenProxy.bat` file located at `[CORTICON_HOME_\Server .NET\samples\wcf-client` for the WSDL options, typically `/namespace:` and `/out:`.

---

6. Write C# client code to call the web service. We provide a sample in `CallCargoService.cs`, which sets values of attributes used in the rules.

7. Compile `CargoDecisionService.cs` and `CallCargoService.cs` using the `csc *.cs` command. Generally, the compile process needs to occur in your .NET Framework root directory, so you may need to move both C# files to that directory prior to compilation. In our case, the .NET Framework is installed at `C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319`

   This generates an executable file named `CallCargoService-webservice.exe`. Store the file in your `[CORTICON_WORK_DIR]`.

8. If you have not already done so, deploy the `tutorial_example` Decision Service to Corticon Server for .NET on IIS. Follow the instructions for Creating and Installing a Deployment Descriptor File.

9. Run `CallCargoService-webservice.exe` to execute the call to Corticon Server. You will see the following output:

   Figure 16: Invoking Corticon Server for .NET via C# Sample Code

```
Calling service tutorial_example
---------------------------------------
Response
----------------------
WorkDocumentsType
Cargo weighing between 150,000 and 200,000 pounds must be packaged in
a container
```