



Corticon

Web Services

Copyright

Visit the following page online to see Progress Software Corporation's current Product Documentation Copyright Notice/Trademark Legend: <https://www.progress.com/legal/documentation-copyright>.

Last updated with new content: Corticon 7.1

Updated: 2025/03/17

Table of Contents

What is a web service.....	7
Overview of Corticon web services.....	9
Web services on Java.....	10
Set up Corticon Server for Java.....	10
Test the installed Corticon Server on Java.....	10
Secured deployment on Java web services.....	13
Web services on .NET.....	23
Set up Corticon Server for .NET on IIS.....	23
Set up Corticon Server for .NET for multiple applications.....	28
How to implement deployment security on .NET web services.....	34
Secured deployment on .NET web services.....	35
Corticon Server files and API tools.....	37
Deploy rules to the Corticon Server.....	39
How to call a Decision Service	43
How to invoke Corticon Server.....	45
REST call.....	45
Doc topics on REST /JSON integration.....	49
XML call.....	49
SOAP call.....	50

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 or REST messages, typically conveyed using HTTP or HTTPS with corresponding JSON-formatted or XML-serialized text files.

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. By definition, the rules within a Decision Service are complete and unambiguous; for a given set of inputs, decision integrity is ensured because the Decision Service addresses every logical possibility uniquely.

What is a web services caller?

A web services caller 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 call web services.

Overview of Corticon web services

Corticon Server hosts web services with Java on Windows using an embedded application server, and can host Java web services on several Windows, UNIX, and Linux application servers and with .NET on Microsoft IIS server, as detailed on the Progress [Corticon Supported Platforms Matrix](#).

The embedded Java application server is Apache Tomcat, which serves as the test application server. Corticon Studio acts as a web service client, which allows you to make REST and SOAP calls to a running application server.

Corticon Studio is where Decision Services are created and tested, and then packaged for deployment on an application server as web services. The Corticon Server can be a stand alone server or deployed in a .NET or java application.

Note: The [Corticon Server 7.x sample WAR installation for different Application Servers](#) provide detailed instructions on configuring Java web services on other application servers.

Deploying web services with Java

The Corticon Server installer contains an Apache Tomcat web server with a complete set of utilities and administrative tools. The installer creates a Tomcat instance configured with Corticon's default web services deployed using Corticon's web application axis.war. This web application allows web service clients to interact with the Corticon Server and its deployed Decision Services running on that instance.

The Tomcat application server (<http://tomcat.apache.org/>) includes a Java servlet container for hosting web applications. Tomcat is the most widely used application server on the market. Corticon uses a default Tomcat application server to demonstrate how web service clients can deploy and execute against Decision Services through the web service endpoints inside the axis.war web application.

Deploying web services with .NET

The Corticon Server for .NET contains all necessary files for the Corticon Server to run in a local Microsoft IIS server. The installer will not preconfigure IIS with Corticon's web services. You must run a script as Administrator to move all necessary files to an IIS Server (c:/inetpub/wwwroot) and deploy the Corticon web services.

For details, see the following topics:

- [Web services on Java](#)
- [Web services on .NET](#)

Web services on Java

The Corticon Server for Java can be installed into any application server that has a Java container. As part of the Corticon Server installer, a web application called `axis.war` is downloaded. This `axis.war` contains the web services that will call into the Corticon Server running in the application server. These web services allow clients to execute against Decision Services through SOAP and REST. The administrative web services (deploy and modify Decision Services) can be called through REST only.

As you approach production, you might want to access the server download bundle that enables the Corticon Server to run on other application servers: [Corticon Supported Platforms Matrix](#). See also [Corticon KnowledgeBase entries](#) that describe each application server setup.

Note: HTTPS - If you want to set up your Java server for TLS-secured communications, see the topic [How to set up encryption between Java servers and clients](#) on page 21

Set up Corticon Server for Java

The *Corticon install guide* describes the system requirements and options for Corticon Server installation. Download and run the installer for Corticon Server. Choose the Corticon Server for Java option. The installer will install utilities, scripts, and samples as well as resources that will be provisioned to the Java server.

Note: The default Corticon server installation does not include an evaluation license. You need a proper evaluation or production license from Progress to run the software. If not, Corticon Server will not work. Once you have a non-evaluation license JAR, navigate to the Corticon Server installation location, typically `C:\Progress\Corticon_Server_Work_x.x\License\Server\` directory to paste the file and, if necessary, overwrite the existing file in that location. See [Corticon runtime component licensing](#) "Corticon runtime component licensing" in the *Corticon Installation Guide* for additional information.

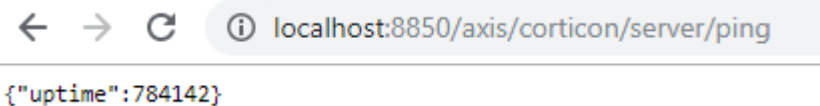
Note: The installer does not install Java or the Java application server that are required before you can test the Java server.

Test the installed Corticon Server on Java

Start the Corticon Server to launch the `axis.war` application server that was deployed. With the application server running, type the following URL in your preferred browser:

```
http://localhost:8850/axis/corticon/server/ping
```

That makes a REST call to the running Corticon Server to get the time (in milliseconds) that the Corticon Server has been running, as illustrated:



Additionally, you can use any REST or SOAP test tools, such as Postman, Swagger or SoapUI, to connect through the web services deployed inside the web application `axis.war`.

Testing with Swagger

Note: Corticon incorporates Swagger into its REST endpoints; however, you must take a few steps to enable it. See [The REST API Swagger documentation](#)

To access Corticon's Swagger page, enter the URL to the machine with its port, the web application name, and the string `swagger`. The Tomcat server that comes with the Corticon Server install has opened port 8850 to receive requests. The full URL to connect to the Corticon Swagger page is: <http://localhost:8850/axis/swagger>

Note: Remote Swagger target—Swagger can respond to a Corticon Server installation on a remote machine. To do so, edit the local Swagger configuration definition for Tomcat in the Corticon Server home at `C:\Progress\Corticonx.x\Server\tomcat\webapps\axis\swagger\index.html`. Modify the line:

```
url = "http://localhost:8850/axis/corticon/openapi.json"
```

On 6.1 and earlier:

```
url = "http://localhost:8850/axis/corticon/swagger.json"
```

to replace `localhost` with the IP address or DNS-resolvable machine name of your web/application server.

Swagger
Supported by SMARTBEAR

http://localhost:8850/axis/corticon/openapi.json Explore

Corticon Server REST APIs

http://localhost:8850/axis/corticon/openapi.json

Servers

/axis

Decision Service : Operations on a Decision Service Deploys a decision service

Executes a decision service Decision Service : Execute Decision Service

Batch Execution : Operation on Batch Execution Service Gets Batch Execution details

Server : Operations on the Server Get Server properties

GET /corticon/server/getProperties Gets Server properties.

GET /corticon/server/ping Get server uptime in milliseconds.

POST /corticon/server/setLicense Sets Server license

POST /corticon/server/setProperties Sets Server properties

For example, choose the **Server**, typically `axis`, and then choose the Server section and its **ping** command:
On the panel that drops down with implementation notes and response messages, click **Execute**.

The response is in the following form:

The screenshot shows a web service client interface. At the top, a GET request is defined for the endpoint `/corticon/server/ping` with the description "Get server uptime in milliseconds." Below this, a "Parameters" section indicates "No parameters" and includes a "Cancel" button. An "Execute" button is present. The "Responses" section shows the following details:

- Curl:** `curl -X 'GET' \ 'http://localhost:8850/axis/corticon/server/ping' \ -H 'accept: */*'`
- Request URL:** `http://localhost:8850/axis/corticon/server/ping`
- Server response:**

Code	Details
200	<p>Response body</p> <pre>{ "uptime": 384864 }</pre> <p>Response headers</p> <pre>connection: keep-alive content-length: 17 content-type: application/json date: Tue, 07 Sep 2021 18:15:47 GMT keep-alive: timeout=20</pre>

The Corticon Java web service is running.

Secured deployment on Java web services

When planning how you will deploy and manage Corticon Decision Services, you need to consider how you will secure the deployment. When deploying Corticon you can use the basic authentication and encrypted communication of your host application server to secure your deployment.

- **Authentication** requires anyone accessing a server to authenticate by supplying their username and password credentials. Configuring authentication gives you control over which users can access a server and the actions they can perform. For example, a user might be able call a Decision Service but not to deploy one.
- **Encryption** enables confidential communication between a server and a client. Enabling HTTPS requires that you add your signed CA certificate to each server, and a CA client certificate on each of the clients that will access it.

How to implement deployment security on Java web services

When deploying a Decision Service outside a firewall, in a cloud – anywhere unapproved users might try to access it -- you must consider how to secure the Decision Service and the Corticon Server's administrative API. Corticon Server lets you deploy secure Decision Services by:

- Securing calls to Decision Services
- Securing calls to management APIs
- Encrypting payloads
- Using user credentials for database access when using Enterprise Data Connections (EDC), or Advanced Data Connectors (ADC), or when using REST service call-outs

How to set up authentication for secure Java server access

To define the authentication mechanism and constraints, update the security configuration defined in the `web.xml` file inside the web archive on the Corticon Server. In a default installation that location is `[CORTICON_HOME]\Server\tomcat\webapps\axis\WEB-INF\web.xml`.

Within the `web.xml` is a commented-out block that defines common security constraints. Uncommenting this block enables basic authentication when you restart the server.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>All Corticon SOAP Servlet Access</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>ROLE_CorticonAdmin</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Corticon Server Realm</realm-name>
</login-config>

<security-role>
  <role-name>*</role-name>
</security-role>
```

Note: If you already uncommented this section to enable HTTPS, review the `web-resource-collection` defined, and then add the `auth-constraint` block, and uncomment the `login-config` and `security-role` sections.

With the above configuration, every time a user tries to access the server through a URL, a valid username/password must be supplied and verified. You need to decide whether to restrict defined user *roles* to specified URLs – the *endpoints* that perform specific actions. That is described in the next topic.

The default user definitions for Apache Tomcat are defined in the `tomcat-users.xml` file (in a default installation its location is `[CORTICON_HOME]\Server\tomcat\conf\tomcat-users.xml`) as follows:

```
<role rolename="ROLE_CorticonAdmin" />
<role rolename="ROLE_CorticonExecute" />

<user username="admin" password="admin" roles="ROLE_CorticonAdmin, ROLE_CorticonExecute"
/>
<user username="ccuser" password="ccuser" roles="ROLE_CorticonExecute" />
```

You can modify the passwords and add additional users to this file.

How to secure Java Server endpoints

You can choose to restrict defined user *roles* to specified URLs – the *endpoints* that perform specific actions. You can put a constraint on a particular URL or many URLs associated with the web application. Endpoint security is defined in `<security-constraint>` tags.

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>
```

You can define the URLs where the constraint will apply. For the constraint to be applied to all URLs associated with your web application, set the value to `/*`. However, a finer level of granularity can be applied. Here are the five core URL pattern matches associated with Corticon SOAP and REST API URLs:

1. Corticon Execute SOAP Servlet:

```
/services/CorticonExecute
```

You can execute Decision Services via SOAP requests. See [Execute SOAP example](#).

2. Corticon Decision Service Admin REST:

```
/corticon/decisionService/*
```

Similar to the Admin SOAP endpoint, this endpoint is over REST. Because REST has a distinct URL for each operation, this endpoint uses an `*` for URL matching of all Decision Service Admin operations. See [Admin DS example](#).

3. Corticon Server Admin REST:

```
/corticon/server/*
```

You can perform Admin operations against the Corticon Server over REST such as getting Corticon Server metrics and properties. See [Admin REST example](#).

4. Corticon Execute REST:

```
/corticon/execute
```

You can execute Decision Services via REST requests. See [Execute REST example](#).

```
<security-constraint>
  <web-resource-collection>
    <auth-constraint>
```

When the user passes in their credentials (username/password), the authentication process verifies that the credentials match a role that has been defined on the server. This section in the `web.xml` lets you limit access to the URL based on particular roles. If you don't want to limit based on particular roles, then this value should be set to `*`.

However, you may not want everyone to have access to the Admin URLs. In this particular case, you must add a specific `<security-constraint>` for the Admin URLs with an assigned `<role-name>` to limit which types of users can access the URLs.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Corticon Admin SOAP Servlet Access</web-resource-name>
    <url-pattern>/services/CorticonAdmin</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>ROLE_CorticonAdmin</role-name>
```

```
</auth-constraint>
</security-constraint>
```

Each role-name defined in any security-constraint in the web.xml must be defined in a <security-role> tag.

Example: (All role names)

```
<security-role>
  <role-name>*</role-name>
</security-role>
```

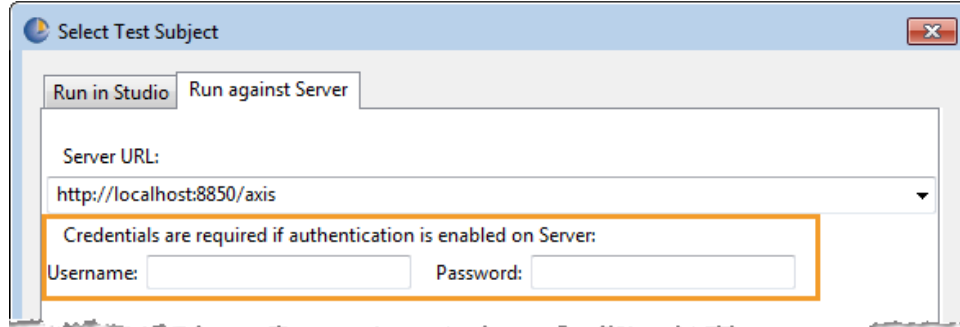
Example: (Two specific role names)

```
<security-role>
  <role-name>ROLE_CorticonAdmin</role-name>
</security-role>
<security-role>
  <role-name>ROLE_CorticonExecute</role-name>
</security-role>
```

Note: If you want all role names to be valid for the web application, then add * in the XML Element. Even if you have set the constraint's role-name to *, you have to define a <security-role> role-name with *.

How to use authentication in Corticon Studio

When a Corticon Studio Ruletest has to use a secured deployed server to test its rules, credentials are required to connect to the server. To support this functionality the **Select Test Subject** dialog's **Run against Server** tab lets you add credentials, as shown:



The credentials are saved within each Testsheet for automatic re-use later.

Note: The credentials must permit both Admin and Execution operations.

How to use authentication in client requests

When a request is submitted by a client to a Decision Service on a secured Corticon Server, the client must have credentials to enable connection to the server. The credentials must permit Execution operations.

You embed the username/password in the HTTP GET or an HTTP POST in the `Authorization` parameter in the header of the request, as follows:

1. Get the `HttpPost` object that is to connect to the URL on the Server:

```
HttpPost postRequest = new HttpPost("http://localhost:8850/corticon/execute");
```

2. Create an `encodedString` and add it to the specific Header:

```
// Encode the username/password in Base64. This is needed for REST calls
String username = "admin";
String password = "admin";
encodedString = Base64.encode((username + ":" + password).getBytes());

String base64EncodedCredentials = "Basic " + encodedString;

// Add the String to the appropriate Header
postRequest.setHeader("Authorization", base64EncodedCredentials);
```

How to use authentication within Web Console

When you manage servers in the Web Console, you need to maintain credentials that enable access to each managed server, as shown:

Authentication

☒ Server Requires Authentication

admin

.....

The credentials are saved within each Server definition in the Web Console repository.

Note: Using LDAP When your business needs require your users to be authenticated through an LDAP server, you can set up LDAP authentication for Web Console users. See the *"How to use LDAP for Web Console authentication" in the Corticon Web Console Guide*.

How to use API calls to a secure web application

The following are examples of each of the core URL patterns outlined in [How to secure Java Server endpoints](#) on page 15.

CorticonExecute SOAP example

In this example, the SOAP call is made to the `/axis/services/CorticonExecute` URL while passing a `CorticonRequest` String through a remote procedure call (RPC) to the remote server.

```
IMPORT STATEMENTS:
import java.net.URL;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

SAMPLE CODE:
Service service = new Service();
Call call = (Call) service.createCall();

String username = "ccuser";
String password = "ccuser";

// Add in username/password
call.setProperty(Call.USERNAME_PROPERTY, username);
call.setProperty(Call.PASSWORD_PROPERTY, password);
```

```
URL serverURL = new URL("http://localhost:8850/axis/services/CorticonExecute");
String methodName = "executeRPC";

String corticonRequest = "";
corticonRequest += "<CorticonRequest decisionServiceName=\"ProcessOrder\">";
corticonRequest += "    <WorkDocuments>";
corticonRequest += "        <Order id=\"Order_id_1\">";
corticonRequest += "            <dueDate>1/1/2008</dueDate>";
corticonRequest += "            <myItems id=\"Item_id_1\">";
corticonRequest += "                <price>10.000000</price>";
corticonRequest += "                <product>Ball</product>";
corticonRequest += "                <quantity>20</quantity>";
corticonRequest += "            </myItems>";
corticonRequest += "        </Order>";
corticonRequest += "    </WorkDocuments>";
corticonRequest += "</CorticonRequest>";

Object[] variables = new Object[]{corticonRequest};

call.setTargetEndpointAddress(serverURL);
call.setOperationName(methodName);

// Make the actual call to the Web Service
String targetNamespaceXSD = "http://soap.corticon.com";
String returnValue = (String) call.invoke(targetNamespaceXSD, methodName, variables);
```

Corticon Decision Service Admin REST example

In this example, the REST API call is made to the `/axis/corticon/decisionService/list` URL requesting a list of Decision Services that are deployed on the remote server.

```
IMPORT STATEMENTS:
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import com.sun.org.apache.xml.internal.security.utils.Base64;

SAMPLE CODE:
String serverURL = "http://localhost:8850/axis/corticon/decisionService/list";

HttpGet getRequest = new HttpGet(serverURL);

String username = "admin";
String password = "admin";

// Encode the username/password in Base64. This is needed for REST calls
String encodedString = Base64.encode((username + ":" + password).getBytes());

String base64EncodedCredentials = "Basic " + encodedString;

getRequest.setHeader("Authorization", base64EncodedCredentials);

// Send the request; It will immediately return the response in HttpResponse object //
// if any
CloseableHttpClient httpClient = HttpClients.createDefault();
HttpResponse response = httpClient.execute(getRequest);

// First verify for valid status code
int statusCode = response.getStatusLine().getStatusCode();
if (statusCode != 200)
{
    String error = "Failed with HTTP error code " + statusCode + ": " +
        response.getFirstHeader("error");
}
```

```

        throw new Exception(error);
    }

    HttpEntity httpEntity = response.getEntity();
    String responseObjectString = EntityUtils.toString(httpEntity, "UTF-8");

```

CorticonAdmin REST example

In this example, the REST API call is made to the `corticon/server/info` URL requesting a `JSONObject` containing information about what Decision Services are currently deployed on the remote server.

```

IMPORT STATEMENTS:
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import com.sun.org.apache.xml.internal.security.utils.Base64;

SAMPLE CODE:
String serverURL = "http://localhost:8850/axis/corticon/server/info";

HttpGet getRequest = new HttpGet(serverURL);

String username = "admin";
String password = "admin";

// Encode the username/password in Base64. This is needed for REST calls
String encodedString = Base64.encode((username + ":" + password).getBytes());

String base64EncodedCredentials = "Basic " + encodedString;

getRequest.setHeader("Authorization", base64EncodedCredentials);

// Send the request; It will immediately return the response in HttpResponse object //
// if any
CloseableHttpClient httpClient = HttpClients.createDefault();
HttpResponse response = httpClient.execute(getRequest);

// First verify for valid status code
int statusCode = response.getStatusLine().getStatusCode();
if (statusCode != 200)
{
    String error = "Failed with HTTP error code " + statusCode + ": " +
response.getFirstHeader("error");

    throw new Exception(error);
}

HttpEntity httpEntity = response.getEntity();
String responseObjectString = EntityUtils.toString(httpEntity, "UTF-8");

```

CorticonExecute REST example

In this example, the REST call is made to the `/corticon/execute` URL while passing a `JSONObject` as the payload to the remote server.

```

IMPORT STATEMENTS:
import java.io.StringWriter;

import javax.ws.rs.core.MediaType;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;

```

```
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONObject;

import com.sun.org.apache.xml.internal.security.utils.Base64;

SAMPLE CODE:
String serverURL = "http://localhost:8850/axis/corticon/execute";

HttpPost postRequest = new HttpPost(serverURL);

//Set the API media type in http content-type and the Decision Service Name to the header
postRequest.addHeader("content-type", MediaType.APPLICATION_JSON);
postRequest.addHeader(ICcServer.REST_HEADER_DECISION_SERVICE_NAME, "ProcessOrder");

String username = "ccuser";
String password = "ccuser";

// Encode the username/password in Base64. This is needed for REST calls
String encodedString = Base64.encode((username + ":" + password).getBytes());

String base64EncodedCredentials = "Basic " + encodedString;

postRequest.setHeader("Authorization", base64EncodedCredentials);

// Create JSON Payload
StringWriter writer = new StringWriter();

/*
String jsonPayload = {"name":"ProcessOrder",
  "Objects": [{
    "myItems": {
      "product": "Ball",
      "price": "10.000000",
      "quantity": "20",
      "__metadata": {
        "#id": "Item_id_1",
        "#type": "Item"
      }
    },
    "__metadata": {
      "#id": "Order_id_1",
      "#type": "Order"
    },
    "dueDate": "1/1/2008",
  }]
}
*/

JSONObject jsonPayload = new JSONObject();
jsonPayload.put("name", "ProcessOrder");
// Create Order object
JSONObject order = new JSONObject();
order.put("dueDate", "1/1/2008");

JSONObject orderMetadata = new JSONObject();
orderMetadata.put("#id", "Order_id_1");
orderMetadata.put("#type", "Order");

order.put("__metadata", orderMetadata);

// Create Item object
JSONObject item = new JSONObject();
item.put("product", "Ball");
item.put("price", 10.0);
item.put("quantity", 20);
```

```

JSONObject itemMetadata = new JSONObject();
itemMetadata.put("#id", "Item_id_1");
itemMetadata.put("#type", "Item");

item.put("__metadata", itemMetadata);

// Add Item object to rolename myItems
order.put("myItems", item);

// Add Order to main payload
JSONArray rootObjects = new JSONArray();
rootObjects.put(order);
jsonPayload.put("Objects", rootObjects);

// Attach JSONObject to the postRequest
jsonPayload.write(writer);
StringEntity userEntity = new StringEntity(writer.getBuffer().toString(), "UTF-8");
postRequest.setEntity(userEntity);

// Send the request; It will immediately return the response in HttpResponse object //
// if any
CloseableHttpClient httpClient = HttpClients.createDefault();
HttpResponse response = httpClient.execute(postRequest);

// First verify for valid status code
int statusCode = response.getStatusLine().getStatusCode();
if (statusCode != 200)
{
    String error = "Failed with HTTP error code " + statusCode + ": " +
        response.getFirstHeader("error");

    throw new Exception(error);
}

HttpEntity httpEntity = response.getEntity();
String responseObjectString = EntityUtils.toString(httpEntity, "UTF-8");

```

How to set up encryption between Java servers and clients

Encryption between Java web services and clients has two tasks:

- [Enabling HTTPS](#)
- [Enabling a client to publish to a secure Corticon Server](#)

Enabling HTTPS

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

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

To enable HTTPS on Corticon Server for Java:

1. Obtain a private key and a Web server digital certificate.
2. Install the Web server digital certificate in the Web server.

3. Start the Corticon Server. When startup is complete, stop it. The initial startup creates the `web.xml` file.
4. Edit the file `web.xml` located at `[CORTICON_HOME]\Server\tomcat\webapps\axis\WEB-INF\` to uncomment the following section:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Corticon Server</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  ...
</security-constraint>
```

Add in the following block to replace ... above:

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

Note: If you already uncommented this section to enable Basic Authentication, review the `web-resource-collection` defined, and then add the `user-data-constraint` `transport-guarantee` block.

5. Save the file.

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

Enabling a client to publish to a secure Corticon Server

Corticon Studio supports encrypted communications to a Corticon Server. To enable HTTPS communication between the Server and the Client, you must obtain and install public key certificates for the Corticon Studio. The public certificate then needs to be imported to the Java keystore for the Corticon Studio.

If you want to enable HTTPS communications, your Corticon Studio clients must also be configured to allow secure HTTPS communications with the Corticon Server. Enabling the Corticon Studio to publish to a secure Corticon Java Server, Corticon Studio supports Transport Layer Security (TLS)-enabled communications to a Corticon Server. To enable TLS communication between the Server and the Client, you must obtain and install public key certificates for the Corticon Studio. The public certificate then needs to be imported to the Java keystore for the Corticon Studio. The typical technique for doing that is to first create a keystore by running the following command from a command prompt:

```
keytool.exe -genkey -keystore my.keystore -storepass myPwd
```

Then, from the command prompt, run the following command to import the certificate:

```
keytool.exe -import -alias [server] -file [server].der -keystore my.keystore -storepass myPwd
```

Launch the Eclipse client from a command prompt using the following command:

```
eclipse -vmargs -Djavax.net.ssl.trustStore="drive:\path\to\certificate\my.keystore"
-Djavax.net.ssl.trustStorePassword=myPwd
```

Corticon Java Server exposes several REST endpoints that can be leveraged through Swagger to perform certain administrative tasks, execute a decision service, or perform batch processing.

How to enable Kerberos Authentication in Studio and Server

When your organization has established a Kerberos realm on an EDC or ADC Datasource, you can enable your Studio and Server Windows installations to successfully connect using the Kerberos Authentication option instead of user credentials by enabling the `AllowTgtSessionKey` registry key, and then running Corticon as Administrator.

To enable the `AllowTgtSessionKey` registry key on Studio or Server:

1. Open `regedit.exe`, and then navigate to the node:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters
```

2. Right-click on `Parameters`, then choose **New > DWORD**, and name it `AllowTgtSessionKey`. Double-click on the new value name to enter the **Value Data** 1, and then click **OK**.
3. Restart the computer to allow the registry changes to take effect and to refresh the JVM.

To run Corticon Studio or Server as Administrator:

1. Locate the shortcut to **Start Corticon Server** or **Start Corticon Studio**.
2. Right-click, and then choose **Run as Administrator**.

Test the connection where the Datasource connection has elected to use Kerberos authentication in Studio or execute a request against a Corticon Server Decision Service.

Note: If you see errors when connecting Corticon with Kerberos authentication, see [the Kerberos authentication troubleshooting article in the KnowledgeBase](#).

Web services on .NET

Corticon Server for .NET uses the Microsoft Internet Information Service (IIS)¹⁰ as its application server for web services. The first step is to get IIS up and running. On Windows 10 and newer systems, IIS and its supporting .NET Framework are pre-installed but need to be enabled to start working. More than an on/off switch, the multiple features of IIS need to be selected to support the functions it will perform. The Corticon Server for .NET has no application server of its own.

The features you set up for IIS on Windows 10 or Windows 2016 are detailed in the topic *"How to set up IIS server" in the Installation Guide*.

Set up Corticon Server for .NET on IIS

The *Corticon install guide* describes the system requirements and options for Corticon Server installation. Download and run the installer for Corticon Server for .NET. That installer will install utilities, scripts, and samples as well as resources that will be provisioned to the IIS server.

See *"Installing Corticon Server for .NET"*.

Checklist

1. **System requirements are met**—See *"System requirements" in the Install Guide*.
2. **IIS is setup and verified**.—See *"How to set up IIS server" in the Install Guide*.

3. **Corticon Server for .NET is installed on the IIS machine**—See *"Installing Corticon Server for .NET"*. The installer sets up the REST API doc, utilities, scripts, and samples as well as resources that will be provisioned to the IIS server.
4. You have a **Corticon Server for .NET license** that was issued to you.

Note: IMPORTANT: The default Corticon server for .NET installation does not include an evaluation license. You need a proper evaluation or production license from Progress to run the software. If not, Corticon Server for .NET will not work. Once you have a non-evaluation license JAR, navigate to the IIS installation location, typically `C:\inetpub`, and then navigate to its `\wwwroot\Corticon\lib` directory to paste the file and, if necessary, overwrite the existing file in that location.

Note: If you are **replacing an existing Corticon .NET IIS installation**, stop the IIS, then click on `Sites\Default Web Site`, and then right-click on any Applications and choose **Remove**. In the File Explorer, delete all folders except `aspnet_client`. Close the IIS Manager, and then relaunch it.

To install Corticon for .NET, the license, and Java to the IIS server:

1. **INSTALL:** Launch `C:\<Corticon_Server>\Server .NET\IIS\install.bat` as **Administrator**. Press **Enter** to run the script.
2. **LICENSE:** Use the File Explorer to copy the .NET-enabled license file, `CcLicense.jar`, to `C:\inetpub\wwwroot\Corticon\lib`.
3. **JAVA:** Use the File Explorer to copy all the contents of your JRE or JDK 17 folder into your `inetpub\wwwroot\Corticon\JRE` folder.

Create the axis application in IIS

To convert the `axis` directory into an application:

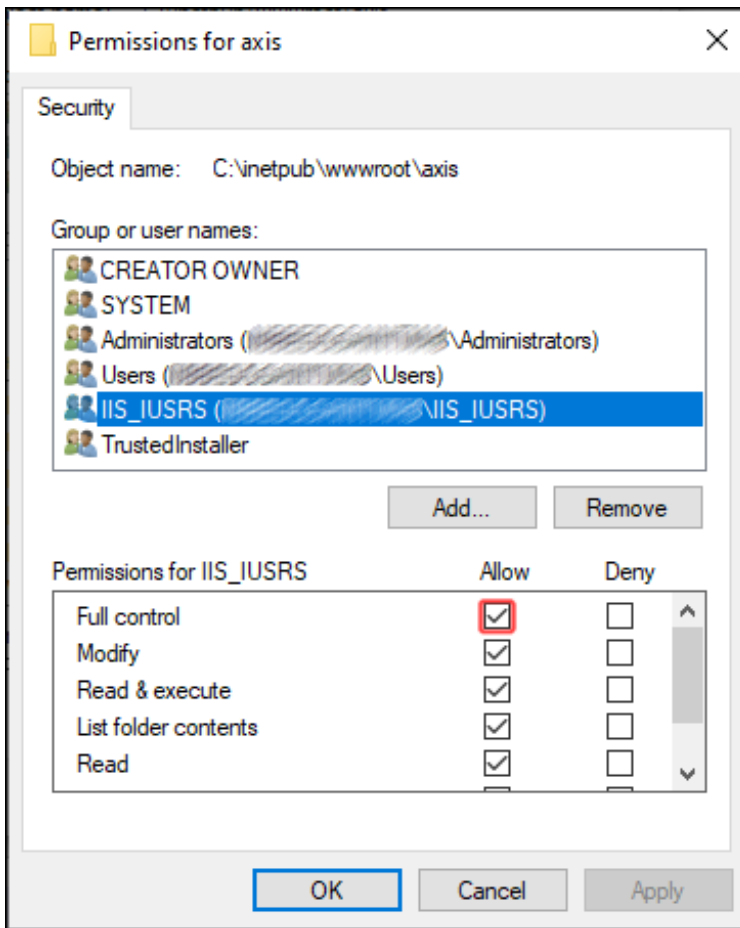
1. Open the IIS Manager app by searching for **IIS Manager**.
2. Expand the connection and then **Sites**.
3. Select the **DefaultWebsite**, then right-click on the root, and select **Refresh**.
4. Right-click on `axis`, and then choose **Convert to Application**.
5. In the dialog, confirm or select **Application pool .NET v.4.5 Classic**, and then click **OK** in both dialogs.

Set Access Permissions for directories in IIS

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 setup, as the 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.

To set access permissions:

1. Right-click on `axis`, and then choose **Edit Permissions**. In the dialog, click the **Security** tab. Click **Edit**, and then choose **IIS_IUSRS**. Click the **Allow - Full control** box, as shown:

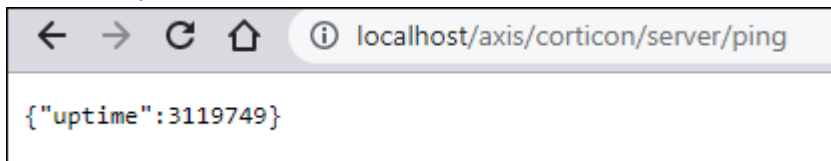


Note: If you do not have a user IIS_IUSRS, click **Add**, and then enter an object name IIS_IUSRS(*yourUserName*\IIS_IUSRS).

- Click **Apply**, then **OK** to both dialogs.
- Similarly, right-click on the `Corticon` folder, and then choose **Edit Permissions**. In the dialog, click the **Security** tab. Click **Edit**, and then choose **IIS_IUSRS**. Click the **Allow - Full control** box. Click **Apply**, then **OK** to both dialogs.
- 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 Internet Information Services is now running the `axis` web service for Corticon.

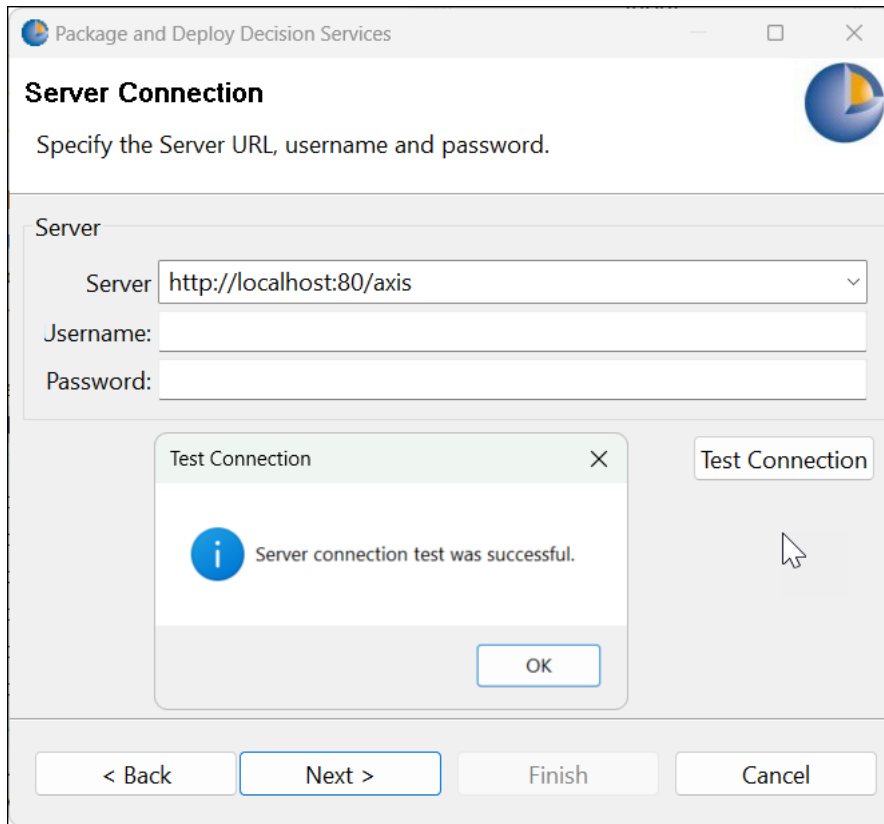
To check that your IIS installation with Corticon has been successful, ping your Corticon IIS Server at <http://localhost:80/axis/corticon/server/ping>. Before you do so, you may want to wait a minute to let IIS finish the startup of Corticon IIS Server. Then, you will see the number of milliseconds in the response that the server has been up, as illustrated:



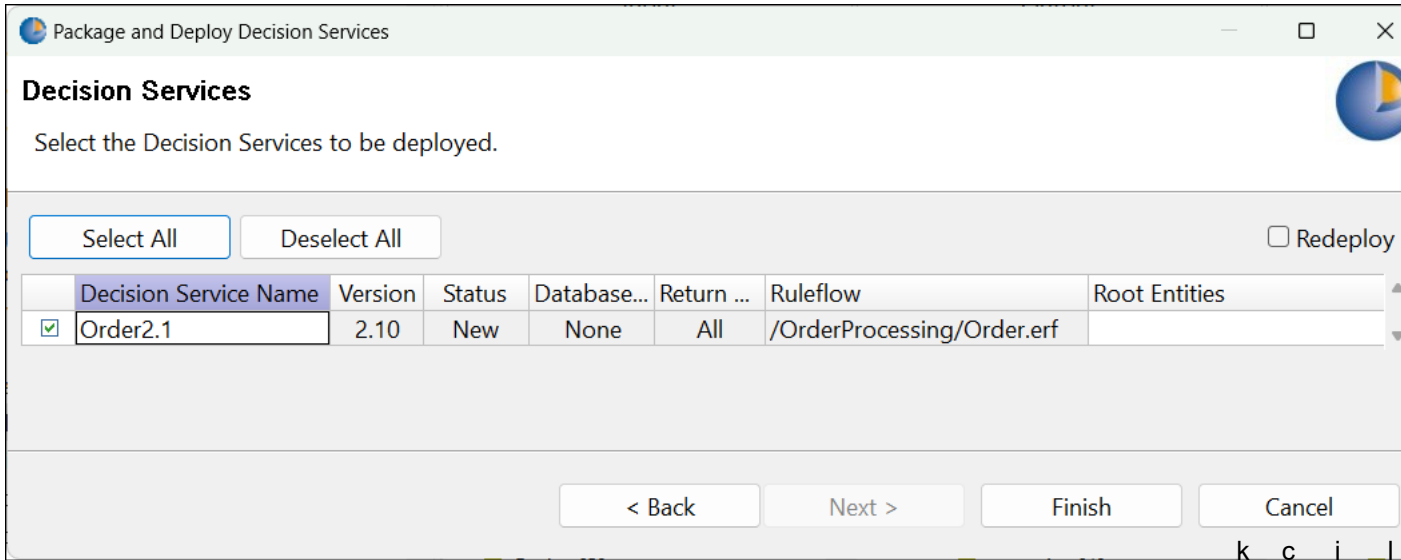
Test IIS server Corticon applications

To test the application running in IIS, produce a decision service, and then review the activity, as described:

1. In Studio, right-click on a sample project **OrderProcessing**, and then choose **Package and Deploy Decision Services**.
2. Choose **Deploy to a Corticon Server**, and then click **Next**.
3. Check that the IIS Manager has been started.
4. Enter the server URL `http://localhost:80/axis`, and then click **Test Connection**, as illustrated:

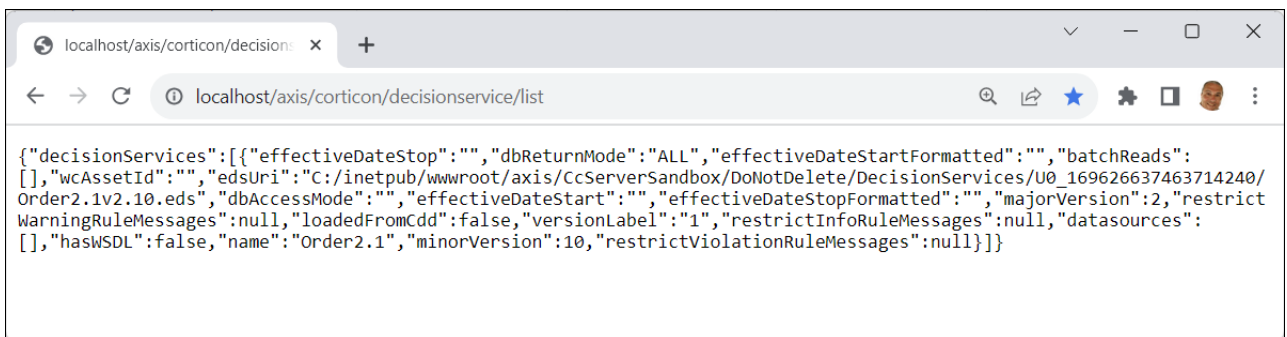


- If you have a good connection, click **Next**, and then select the listed ruleflow. Rename it to add the application name such as `Order2.1`:



Finish..

- In your browser, use <http://localhost/axis/corticon/decision/service/list> to fetch all development decision services:



You have confirmed that the `axis` application is running on IIS server with its decision services and logs.

Note: You should now adjust the log level in the `brms.properties` file in each application space so that the logs do not get overloaded in `DEBUG` mode.

Test rules from Studio Tester

- Open the `Order.ert` Ruletest in the project.
- Select **Ruletest > Testsheet > Change Test Subject**.
- Choose the **Run against Server** tab>.
- Enter the Server URL `http://localhost:80/axis`, and then click **Refresh**:

Select Test Subject

Run in Studio Run against Server

Server URL:
http://localhost:80/axis

Credentials are required if authentication is enabled on Server:
Username: Password:

Decision Services:
Refresh

Name	Major	Minor	Effective Start Da...	Effective Stop Da...
Order2.1	2	10		

Optional Overrides

Major Version: 2
Minor Version: 10
Effective Target Date: / / Time: 0 0 0 AM

OK Cancel

5. Click on the `Order_2.1` line, and then click **OK**.

6. Run the test.

The results show that the rules are running on IIS server.

Set up Corticon Server for .NET for multiple applications

Using multiple IIS servers for deployment stages

Traditionally, you would have multiple IIS Servers that keep each step in development distinct, moving code between servers until it is in production. This time-tested tactic to prevent sharing one server is the only way to insure that a call for reboot on development testing doesn't bring down the production instance. To achieve autonomy it is generally (and highly) recommended that each instance be a separate machine: separate IIS, separate licensed Corticon Server for .NET install, and separate instances of CDDs and decision services. The sequence of operations is where Studio generates a versioned decision service that is tested on `DEV`. When `DEV` accepts it, then production picks up that decision service version for their tests, and takes down any prior version.

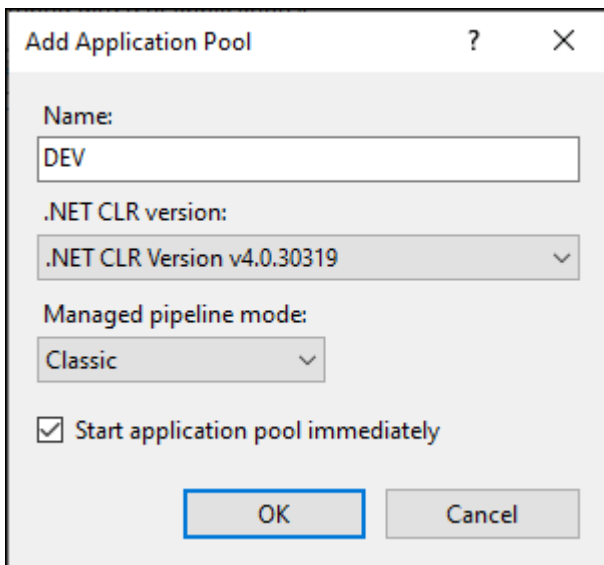
How to set up additional applications on one IIS server

There is a technique you can use that shares an IIS server, keeping the stages of production isolated through IIS application pools. Each application will share the IIS resources and the Corticon resources while maintaining application-specific sandboxes and logs.

Let's assume that you want development (DEV) to be isolated from production (PROD). You will create application pools, then ensure that your files and folders are ready to be replicated, and then create the applications.

Create Application pools

1. In the **IIS Manager**, select **Application Pools**, then choose **Add Application Pool**.
2. In the dialog box:
 - a. Name the pool as in the illustration, **DEV**.
 - b. Choose **.NET CLR Version v4.0.n**.
 - c. Choose **Managed pipeline mode: Classic**

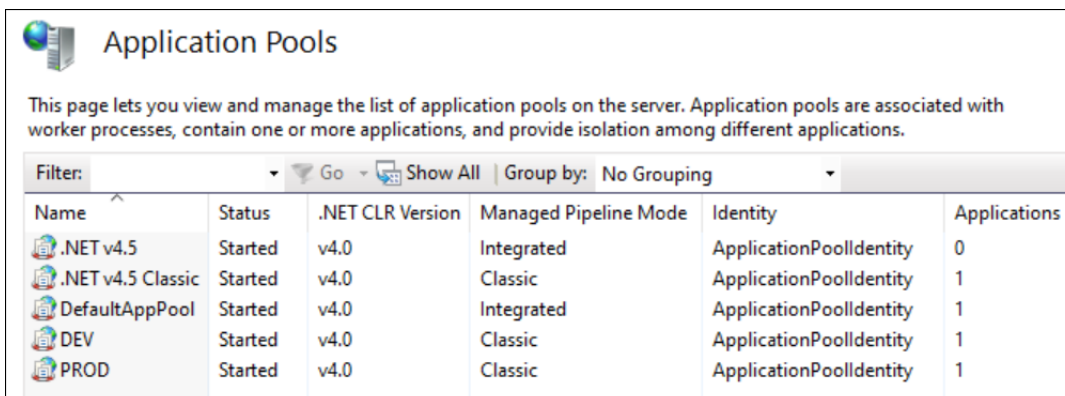


The dialog box titled "Add Application Pool" contains the following fields and controls:

- Name:** A text box containing "DEV".
- .NET CLR version:** A dropdown menu showing ".NET CLR Version v4.0.30319".
- Managed pipeline mode:** A dropdown menu showing "Classic".
- ☒ **Start application pool immediately**
- OK** and **Cancel** buttons at the bottom.

Click **OK**.

3. The new Application Pool is listed with no application.
4. Repeat the steps to create all the Application Pools you want, as illustrated where **PROD** was added:



The screenshot shows the "Application Pools" management page in IIS Manager. It includes a description and a table of existing application pools.

This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

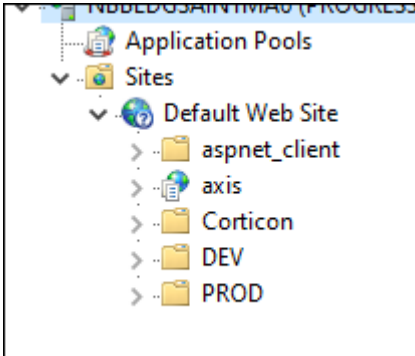
Name	Status	.NET CLR Version	Managed Pipeline Mode	Identity	Applications
.NET v4.5	Started	v4.0	Integrated	ApplicationPoolIdentity	0
.NET v4.5 Classic	Started	v4.0	Classic	ApplicationPoolIdentity	1
DefaultAppPool	Started	v4.0	Integrated	ApplicationPoolIdentity	1
DEV	Started	v4.0	Classic	ApplicationPoolIdentity	1
PROD	Started	v4.0	Classic	ApplicationPoolIdentity	1

Create the Applications

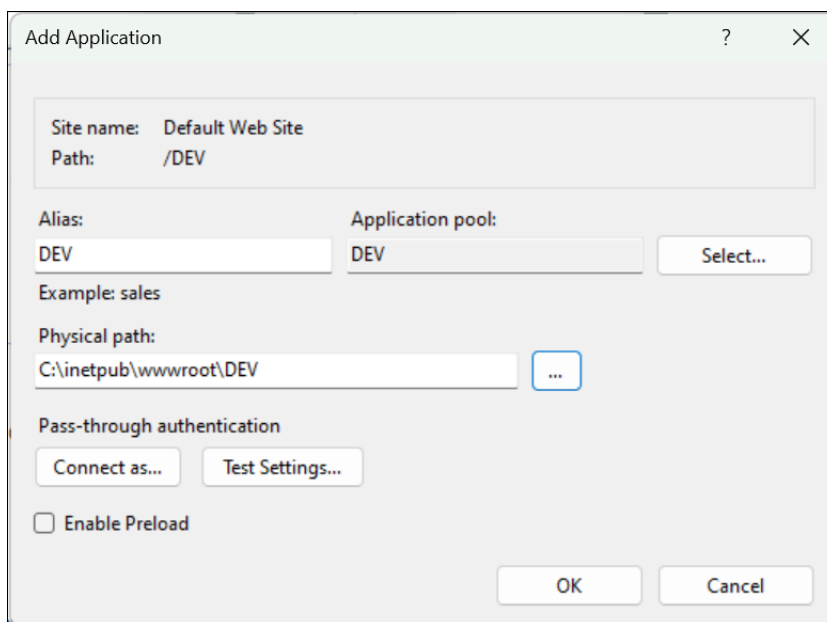
1. Stop the **IIS Manager**.
2. In the **File Explorer** at `C:\inetpub\wwwroot`, copy the `axis` folder, and then paste it.
3. Rename the copied folder to your additional application's name, such as `DEV`.
4. Repeat the copy/paste/rename steps to specify all your applications, such as `PROD`.

Note: Unless you want to retain the `axis` directory as a template, you can delete the `axis` directory by first deleting its directory, and then in the IIS Manager, removing it.

5. Restart IIS Manager to see the new folders:

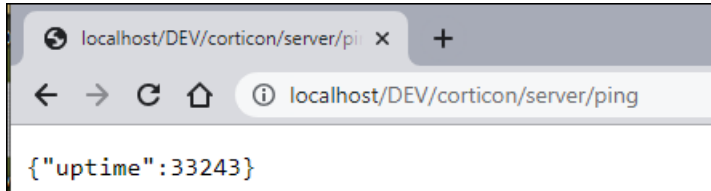


6. On each of the application folders (`DEV` and `PROD`), do the following steps:
 - a. Right-click on the application folder and choose **Edit Permissions**.
 - b. Choose **Security**, then **Edit** to allow `IIS_IUSRS` to have **Full control**. These permissions are expected to be inherited by the `CcServerSandbox`, the `cdd` folder, and `logs` folder.
 - c. Right-click on the application folder, and then choose **Convert to Application**.
 - d. In the **Add Application** dialog, enter a name, and then click **Select** to choose an application pool, as illustrated:



Restart IIS and test the URLs.

In your browser, enter the URL <http://localhost/DEV/corticon/server/ping> to make a REST call to the running development Corticon Server to get the time (in milliseconds) that the Corticon Server has been running on IIS, as illustrated:



Do the same with <http://localhost/PROD/corticon/server/ping>.

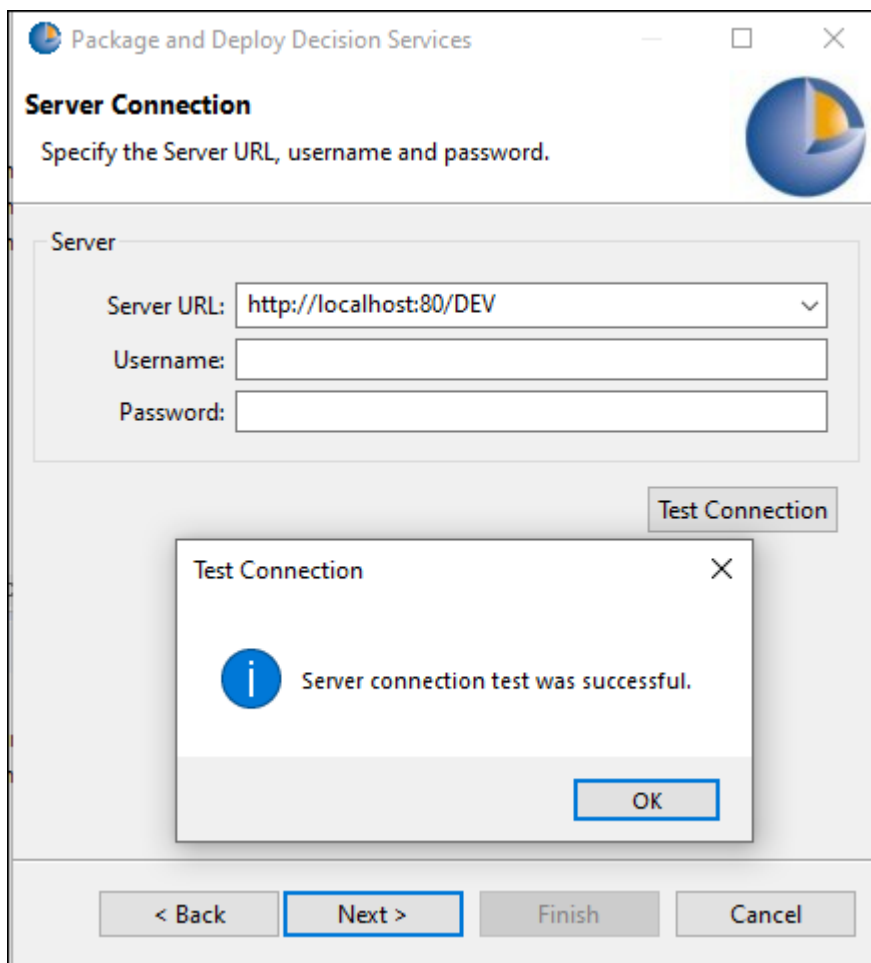
When you see that both applications are running, proceed to testing the applications by publishing a decision service to each.

Note: Use logs to confirm the distinct application spaces It is good idea to edit `brms.properties` in each application, and then set `logLevel=DEBUG`. When you run a test in each application, it will test that the application-specific logs are working. You will then want to change the setting to the default or your preferred level as `DEBUG` produces enormous output.

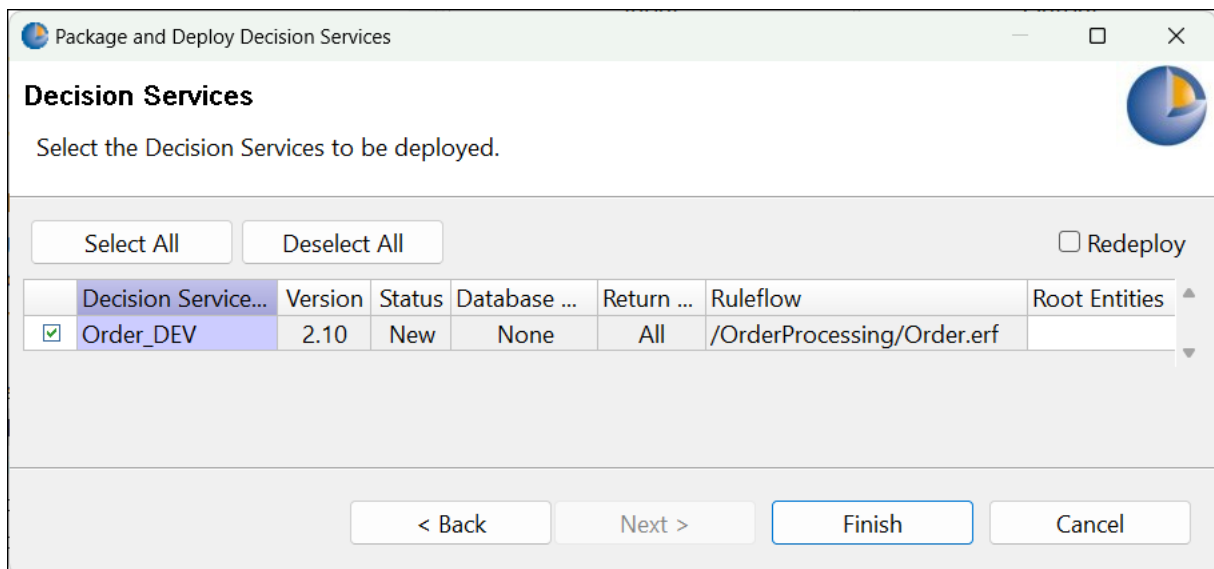
Test IIS server for multiple Corticon applications

To confirm that there are separate applications running in IIS, produce decision services to each of the applications and then review the distinct separation in the IIS server, as described:

1. In Studio, right-click on a sample project **OrderProcessing**, and then choose **Package and Deploy Decision Services**.
2. Choose **Deploy to a Corticon Server**, and then click **Next**.
3. Check that the IIS Manager has been started.
4. Enter the server URL <http://localhost:80/DEV>, and then click **Test Connection**, as illustrated:



- If you have a good connection, click **Next**, and then select the listed ruleflow. Rename it to add the application name such as `OrderDEV`:



Click **Finish**.

- Repeat steps 1 through 5 for the application PROD.

7. In your browser, use <http://localhost/DEV/corticon/decision/service/list> to fetch all development decision services:



8. Use <http://localhost/PROD/corticon/decision/service/list> to fetch all production decision services:



9. In the File Explorer, look at the DEV and PROD folders. Inside each of those folder are their respective sandboxes and logs. Examine the logs to see that they are just for that partition.

Note: You should now adjust the log level in the `brms.properties` file in each application space so that the logs do not get overloaded by DEBUG mode.

You have confirmed that two applications are running in one IIS server, each with distinct decision services and logs.

Test rules in two applications from Studio Tester

1. Open the `Order.ert` Ruletest in the project.
2. Select **Ruletest > Testsheet > Change Test Subject**.
3. Choose the **Run against Server** tab>.
4. Enter the Server URL `http://localhost:80/DEV`, and then click **Refresh**:

Select Test Subject

Run in Studio | **Run against Server**

Server URL:
http://localhost:80/DEV

Credentials are required if authentication is enabled on Server:
Username: Password:

Decision Services:
Refresh

Name	Major	Minor	Effective Start Date	Effective Stop Date
Order_DEV	1	10		

OK Cancel

5. Click on the `Order_DEV` line, and then click **OK**.
6. Select **Ruletest > Testsheet > Add Testsheet**.
7. Enter the Server URL `http://localhost:80/PROD`, and then click **Refresh**.
8. Click on the `Order_PROD` line, and then click **OK**.

You can now run development and production rule tests side by side.

How to implement deployment security on .NET web services

Authentication is done by the application or IIS web server at the wrapper layer (endpoints) to get access to the Corticon Endpoint(s). Once you have been granted permission into the Corticon Endpoint, the endpoint calls into Corticon Server to execute a Decision Service.

If you want to enable HTTPS communications, your Corticon Studio clients must also be configured to allow secure HTTPS communications with the Corticon Server .NET. Enabling the Corticon Studio to publish to a secure Corticon Server .NET, Corticon Studio supports Transport Layer Security (TLS)-enabled communications to a Corticon Server. To enable TLS communication between the Server and the Client, you must obtain and install public key certificates for the Corticon Studio. The public certificate then needs to be imported to the Java keystore for the Corticon Studio. The typical technique for doing that is to first create a keystore by running the following command from a command prompt:

```
keytool.exe -genkey -keystore my.keystore -storepass myPwd
```

Then, from the command prompt, run the following command to import the certificate:

```
keytool.exe -import -alias [server] -file [server].der -keystore my.keystore -storepass myPwd
```

Launch the Eclipse client from a command prompt using the following command:

```
eclipse -vmargs -Djavax.net.ssl.trustStore="drive:\path\to\certificate\my.keystore"  
-Djavax.net.ssl.trustStorePassword=myPwd
```

Corticon Server .NET exposes several REST endpoints that can be leveraged to perform certain administrative tasks, execute a decision service, or perform batch processing.

Secured deployment on .NET web services

When planning how you will deploy and manage Corticon Decision Services, you need to consider how you will secure the deployment. When deploying Corticon you can use the basic authentication and encrypted communication of your host application server to secure your deployment.

- **Authentication** requires anyone accessing a server to authenticate by supplying their username and password credentials. Configuring authentication gives you control over which users can access a server and the actions they can perform. For example, a user might be able call a Decision Service but not to deploy one.
- **Encryption** enables confidential communication between a server and a client. Enabling HTTPS requires that you add your signed CA certificate to each server, and a CA client certificate on each of the clients that will access it.

Corticon Server files and API tools

Corticon Server is a set of Java classes, packaged in Java archive files. The required set of JARs needed to deploy and call Corticon Server is:

JAR	Description
CcServer.jar	The main Corticon Server JAR, containing the core engine logic.
CcConfig.jar	Contains a set of .property files that list and set all the configuration properties needed by Corticon Server. These properties pages are not intended for user access. Instead, the file <code>brms.properties</code> , installed by every product at the root of <code>[CORTICON_SERVER_WORK_DIR]</code> , enables you to add your override settings to be applied after the default settings have been loaded.
CcLicense.jar	An encrypted file containing the licensing information required to activate Corticon Server. Note that you need to obtain a specific license for .NET server.
CcThirdPartyJars.jar	Contains third-party software such as WSDL.
CcDrivers.jar	Contains the database drivers used by Corticon Server.
CcExtensions.jar	Supports Service Callouts and extended language operators.

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

Corticon environment

The installer establishes a common environment configuration file, `\bin\corticon_env.bat`, at the program installation location. That file defines the Progress Corticon runtime environment so that most scripts simply call it to set common global environment settings:

CORTICON_HOME	The installation directory -- either the default location, <code>C:\Progress\Corticon x.x</code> , or the preferred location you specified.
CORTICON_SERVER_WORK_DIR	The work directory -- either the default location, <code>C:\Progress\Corticon_Server_Work_x.x</code> , or the preferred location you specified.

Note:

It is a good practice to use global environment settings

Many file paths and locations are determined by the `CORTICON_HOME` and `CORTICON_SERVER_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.

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

Corticon Server Sandbox

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

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

```
com.corticon.ccserver.sandboxDir=%CORTICON_SERVER_WORK_DIR%\%CORTICON_SETTING%\CcServerSandbox
```

In a default Windows installation, the result for this is

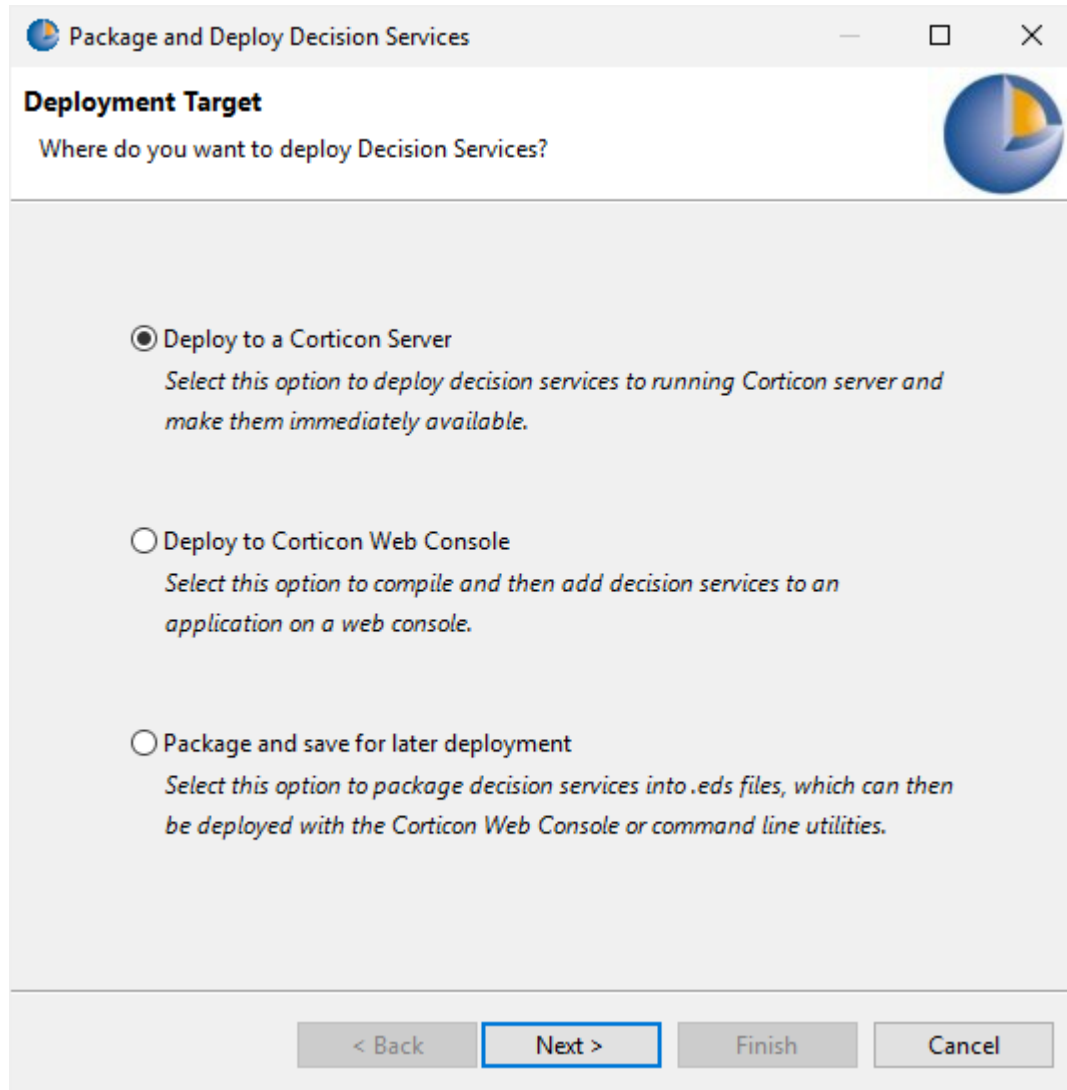
`C:\Progress\Corticon_server_Work_x.x\SER\CcServerSandbox`. In other words, in the `SER` subdirectory of the `CORTICON_SERVER_WORK_DIR`. This directory is created (as well as peer directories, logs and output) during the first launch of Corticon Server.

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.

Deploy rules to the Corticon Server

Once you have installed the Corticon Server, it is ready to have rules deployed to it that can then run when it receives requests. Rules are defined in the Corticon Studio, and tested there against its embedded application server. Ruleflows are then versioned and packaged together with supporting files as compiled Decision Services for runtime on any Corticon Server regardless of platform and application server. The compilation into an encrypted Decision Service ensures that the rules are consistent and reliable wherever they run, whether as a web service on an application server or as an in-process application. The absence of the source files on the server and ad hoc compilation ensures security from unwarranted changes to the rules as well as exposure of the valuable rule assets.

The Corticon Studio's **Package and Deploy Decision Services wizard** summarizes the tactics that get rules onto servers:



- **Deploy to a Corticon Server**

Does the compilation and server transfer in one step. Great for pre-production testing. But many users prefer to keep the development functions and production control distinct.

In many cases, several Decision Services need to be created in the same run. You can use compile and deploy APIs and command line utilities to create scripted runs of builds. See the topics in the *"Packaging and deploying Decision Services"* section in the *Deployment Guide*

- **Deploy to the Corticon Web Console** While not required, the Web Console is a good way to monitor and manage servers and Decision Services. By deploying to Web Console, the development hands off the compiled Decision Service for eventual deployment to a web service it has under management. Staging a Decision Service this way is discussed in *"Decision Services and Applications"* in the *Web Console Guide*.
- **Package and save for later deployment** - Compiling the assets into a Decision Service file on a network location, this intermediate step typically uses Deployment Descriptor files that assembles several Decision Services and miscellaneous files into a easily edited text file.

Create and install Deployment Descriptor files

Deployment Descriptor files (.cdd) let you provide a server with lists of Decision Service, related files, and optional property settings for each one. When Corticon Server starts, it looks for Deployment Descriptor files at [CORTICON_SERVER_WORK_DIR]\cdd.

The Samples directory in a server installation provides some samples that include prewritten CDD files as well as precompiled Decision Services. Try them out to see how the Web Console shows that they immediately deploy and accept requests.

Here is the TradeAllocation sample's Deployment Descriptor file:

```
<?xml version="1.0" encoding="UTF-8"?>
<cdd soap_server_binding_url="http://localhost:8850/axis">
  <decisionservice>
    <name>AllocateTrade</name>
    <path>AllocateTrade.eds</path>
    <options>
      <option name="PROPERTY_AUTO_RELOAD" value="true" />
      <option name="PROPERTY_MAX_POOL_SIZE" value="1" />
    </options>
  </decisionservice>
  <decisionservice>
    <name>Candidates</name>
    <path>Candidates.eds</path>
    <options>
      <option name="PROPERTY_AUTO_RELOAD" value="true" />
      <option name="PROPERTY_MAX_POOL_SIZE" value="1" />
    </options>
  </decisionservice>
</cdd>
```

Try it! To explore this deployment technique, the Web Console is the best way to visualize what happens:

1. Start the Web Console in a clean installation. Note that there are no Decision Services listed.
2. In the server install, go to [CORTICON_SERVER_WORK_DIR]\Samples\Rule Projects\TradeAllocation
3. Copy the Deployment Descriptor CDD file and the two Decision Service EDS files to [CORTICON_SERVER_WORK_DIR]\cdd
4. Refresh the Web Console. The **Discovered Decision Services** list both Decision Services that were deployed to the server stated in the CDD file:

▼ Discovered Decision Services

Decision Services not deployed using the Web Console, can be viewed here but not fully managed.

Decision Service Name ▲	Version ▲	Effective ▲	Expires ▲	CDD ▲	Server
✓ AllocateTrade	1.14			✓	✓ localhost:8850/axis
✓ Candidates	1.14			✓	✓ localhost:8850/axis

For more information, see the topics in *"Use Deployment Descriptors to deploy Decision Services" in the Deployment Guide.*

Setting Server startup to auto load CDD files

The following property can be set on your Corticon Server installation by adding the property and an appropriate value as a line in its `brms.properties` file, and then restarting Server.

Properties related to "auto loading" CDD files into the CcServer during initialization:

`com.corticon.ccserver.autoloadaddir.enable`

Specifies whether the CcServer will try to auto load CDD files

Default value: true

`com.corticon.ccserver.autoloadaddir`

The directory where the CDD files are located.

Note: This property can be changed using following method, which will override this setting.

- `ICcServer.setDeploymentDescriptorDirectoryPath(String)`

Note: Use forward slashes as path separator. Example: `c:/Progress/cdd`

Default value: If the property value is empty, the following path is used: `user.dir/cdd`

where `user.dir` is the value of environment variable "user.dir" which in Windows and Unix returns the directory where container application was started.

```
com.corticon.ccserver.autoloadaddir.enable=true
com.corticon.ccserver.autoloadaddir=%CORTICON_SERVER_WORK_DIR%/cdd
```

How to call a Decision Service

Once the Corticon Server is installed and running deployed Decision Services, you can call this Decision Service by sending a request messages, and then inspect the response messages that return.

A Decision Service in a process or application works best when it is using 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 *called* the Decision Service.

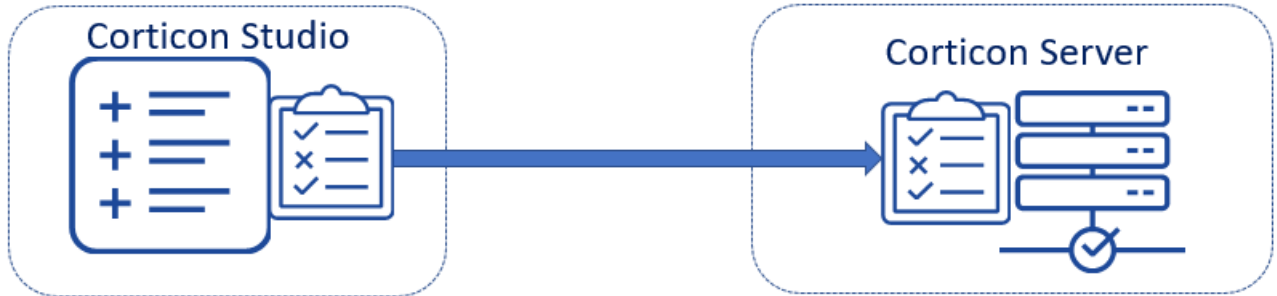
To call a Decision Service as a web service:

- **Within Corticon Studio -**



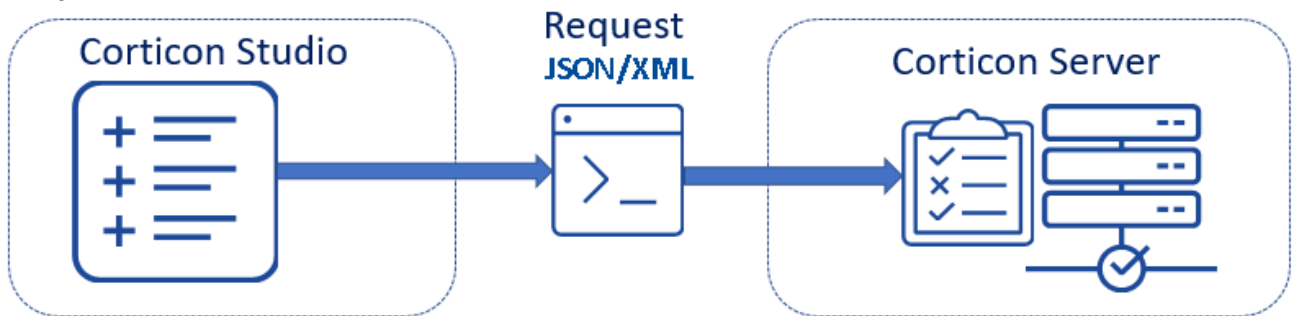
When you run a Ruletest for a Ruleflow in Corticon Studio, the embedded in-process Corticon server is called to process the request and return the response. You did this in the Basic Tutorial. It is one of Corticon's greatest strengths, the ability to real-world test right on the workbench. See the Quick Reference topics and *"Choose a test subject in the Studio workspace"* and *"Execute tests"*.

- **From Corticon Studio to a Corticon Server -**



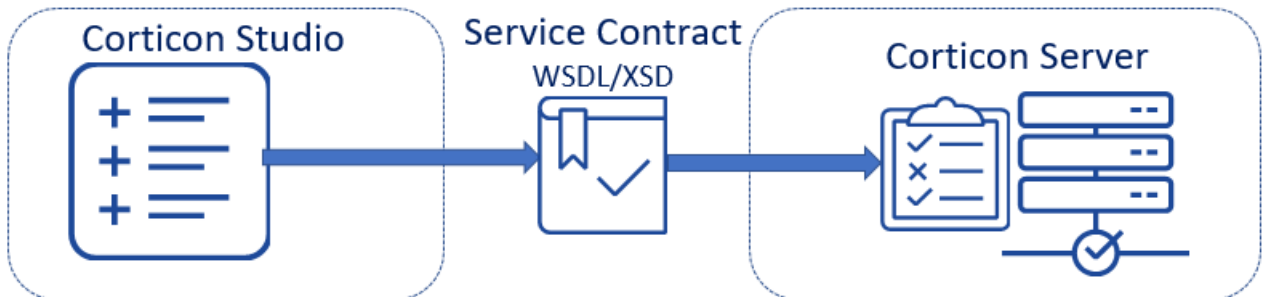
When you have an installed Corticon Server, whether Java or IIS, of the same version as Studio, you can run Ruletests as simulated Decision Services directly from the Studio. This technique enables pre-production testing of a Decision Service while it is still in development. See *"Choose a test subject that is a deployed Decision Service"* in the Quick Reference topics.

- **As a request document submitted to a Corticon Server -**



You can abstract the tests from the Studio by exporting the JSON, SOAP and XML requests to text files. With a few minor edits, you can run them against a deployed Server from the Web Console. See the Quick Reference topics on *"How to export a testsheet."*

- **As a service contract request submitted to a Corticon Server -**



See *"How to integrate Corticon Decision Services"* topics in the Deployment topics.

How to invoke Corticon Server

To make a call to invoke Corticon server, you can use REST or XML/ SOAP. Regardless of which invocation method you choose to call, keep in mind that it, by default, acts in a request—response mode. This means that one request sent from the client will result in one response sent by the Server back to the client. Multiple calls may be made by different clients simultaneously, and the Server will assign these requests to different Reactors in the pool as appropriate. As each Reactor completes its transaction, the response will be sent back to the client.

The processing is consistent, whether within the Studio's built-in server, a deployed Java server, or a .NET server application.

The form of the response will be the same as the request. If making a SOAP call passing XML, the response will be XML. If making a REST call passing JSON, the response will be JSON.

For details, see the following topics:

- [REST call](#)
- [XML call](#)
- [SOAP call](#)

REST call

Corticon supplies a REST API that lets client applications execute against remote Decision Services.

Note: Swagger—Corticon Server installs Swagger for exploring and testing the Corticon Server REST API, but is not enabled by default. Swagger is optional and can be enabled in an instance of Corticon Server. See *"The REST API Swagger documentation" in the Server Guide* for details. The use of Swagger is best limited to development deployments of Corticon Server. It is typically not needed, or wanted, in production deployments. The endpoint to the enabled REST API is:

```
http://localhost:8850/axis/swagger
```

REST API online documentation—The REST API in static format is available in Java Server's install directory for Java and .NET at `C:\Progress\Corticon%207.1\Server\RESTDdoc\index.html` (but not propagated to the IIS Server), and at <https://documentation.progress.com/output/Corticon/7.1/RESTDdoc/>

application/json

The payload set to the REST API is a JSON object that contains the Decision Service and related data to execute against. The format of the JSON object is:

```
{
  "effectiveTimestamp": "string",
  "majorVersion": "string",
  "minorVersion": "string",
  "objects": [
    {}
  ],
  "name": "string"
}
```

```
{
  "name": <string>,
  "majorVersion": <number - optional>,
  "minorVersion": <number - optional>,
  "Objects": <JSONArray of JSONObject>
}
```

where `name`, `majorVersion`, and `minorVersion` values specify which Decision Service will be used for the execution, and the `Objects` value is the payload of the data to be processed.

Note: Parameters—Other optional parameters can be added to before `Objects`:

```
#enableRuleTrace
#restrictInfoRuleMessages
#restrictWarningRuleMessages
#restrictViolationRuleMessages
#restrictResponseToRuleMessagesOnly
#locale
#timezoneId
```

See [Swagger help](#) for details.

On the Corticon Server, you can use REST to execute a JSON-formatted Corticon Request by specifying the target Decision Service. The REST method supported is `HTTP POST` in the form:

```
HTTP POST:base/axis/corticon/execute
```

where *base* is the Corticon Server's IP or DNS-resolvable name and port.

Native JSON

You can use simplified JSON by adding the appropriate decision service name and version parameters in the request URL:

```
curl -X 'POST' \
'http://localhost:8850/axis/corticon/execute/nativejson?name=MyAdvancedTutorial&majorVersion=1&minorVersion=1'

\
-H 'accept: */*' \
-d '{
```

Note: Parameters—Other optional parameters can be added to the header:

```
#enableRuleTrace
#restrictInfoRuleMessages
#restrictWarningRuleMessages
#restrictViolationRuleMessages
#restrictResponseToRuleMessagesOnly
#locale
#timezoneId
```

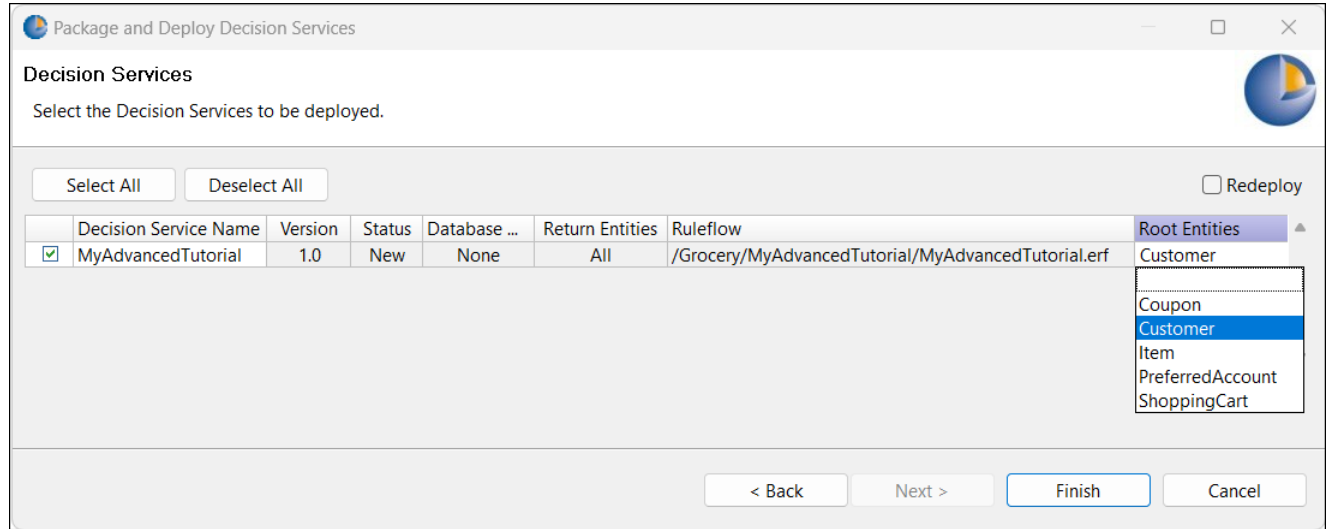
See [Swagger help](#) for details.

List the request payload without metadata tags:

```
{
  "preferredCard": {
    "cumulativeCashBack": 9.24,
    "cardNumber": "12"
  },
  "ShoppingCart": {
    "totalAmount": null,
    "Item": [
      {
        "price": 55,
        "name": "Filet Mignon",
        "department": null,
        "barCode": "39-280-12345"
      },
      {
        "price": 14.99,
        "name": "Beach Towel",
        "department": null,
        "barCode": "32-300-23456"
      },
      {
        "price": 12.5,
        "name": "Ginger Ale Case",
        "department": null,
        "barCode": "32-285-34567"
      }
    ],
    "useCashBack": true,
    "savings": null
  },
  "Name": ""
}
```

Root Entities in the Decision Service package

Where the root entity is not specified and must be inferred, you can save processing time by setting the **Root Entities** field on the Ruleflow while packaging, as illustrated:



application/xml

A REST call can handle XML when the XML-formatted request is identified in the request header as application/xml. Then, the format of the XML object is:

```
<?xml version="1.0" encoding="UTF-8"?>
<CorticonRequest decisionServiceName="string">
  <effectiveTimestamp>string</effectiveTimestamp>
  <majorVersion>string</majorVersion>
  <minorVersion>string</minorVersion>
  <WorkDocuments>
  </WorkDocuments>
</CorticonRequest>
```

Success and error responses

The response returned by the server has two parts:

- A HTTP status code
- An HTTP header containing message execution info

Success response - In addition to the processed request, when execution is successful, the response message's body contains the updated JSON string. A successful execution's shows its status (200) and header.

Error behavior

For all REST executions, an error (an HTTP status code other than 200) provides messages in the response body.

Example:

```
{ "error": {
  "type": "com.corticon.eclipse.rest.delegates.CcRestDelegateRequestErrorException",
```



```

"message": "retrieveServiceProperties(String,String,String)
           Could not find decision service: \"Process Order \",
"cause":{
"stackTrace": [...],

"type":
"com.corticon.service.ccserver.exception.CcServerDecisionServiceNotRegisteredException",
"message": "CcServer.lookupCcServerPoolHighestVersionNumber()
           Decision Service: Process Order is not registered. Update failed."},
"stackTrace": [...], "parentError": {"stackTrace": [...],

"type":
"com.corticon.service.ccserver.exception.CcServerDecisionServiceNotRegisteredException",
"message": "CcServer.lookupCcServerPoolHighestVersionNumber()
           Decision Service: Process Order is not registered. Update failed."}}}

```

Doc topics on REST /JSON integration

When integrating Corticon using REST/JSON, several topics contain helpful information:

Deployment Guide

- *"About creating a JSON request message for a Decision Service".*
- *"Sample JSON request/Response messages".*
- *"How to pass null values in a JSON request".*

Web Services Guide

- *"Execute remote Decision Services using REST/JSON".*
- *"How to use API calls to a secure web application"*
- *"How to use authentication in client (consumer App) requests"*

Server Guide

- *"The REST API Swagger documentation" for enabling and running Swagger.*
- *"Error handling in the REST Management API"*
- *"How to access the Vocabulary metadata of a Decision Service" (useful when mapping data from a consumer application)*

XML call

The structure and contents of a SOAP message are a wrapper on an XML request. For example:

1. In the `OrderProcessing` sample project, open its Ruletest `Order.ert`.
2. Choose **Ruletest > Testsheet > Data > Input > Export Request XML**.
3. Save the file as `ProcessOrderXML.xml`
4. Choose **Ruletest > Testsheet > Data > Input > Export Request SOAP**.
5. Save the file as `ProcessOrderSOAP.xml`
6. Compare the two files.

The SOAP request wrapped the XML `CorticonRequest` content within:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <CorticonRequest/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Once the SOAP request message has been prepared, a SOAP client must transmit the message to the Corticon Server (deployed as a Web Service) via HTTP.

If your SOAP tools have the ability to assemble a compliant request message from a WSDL you generated in Corticon Studio, then it is very likely they can also act as a SOAP client and transmit the message to a Corticon Server deployed to a supported application server. The Web Services section describes how to test this method of invocation with a third-party tool or application as a SOAP client.

When developing and testing SOAP messaging, it is very helpful to use some type of message interception tool (such as **tcpTunnelUI** or **TCPTrace**) that allows you to grab a copy of the request message as it leaves the client and the response message as it leaves Corticon Server. This lets you inspect the messages and ensure no unintended changes have occurred, especially on the client-side.

SOAP call

The structure and contents of a SOAP message are a wrapper on an XML request. For example:

1. In the `OrderProcessing` sample project, open its Ruletest `Order.ert`.
2. Choose **Ruletest > Testsheet > Data > Input > Export Request XML**.
3. Save the file as `ProcessOrderXML.xml`
4. Choose **Ruletest > Testsheet > Data > Input > Export Request SOAP**.
5. Save the file as `ProcessOrderSOAP.xml`
6. Compare the two files.

The SOAP request wrapped the XML `CorticonRequest` content within:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <CorticonRequest/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Once the SOAP request message has been prepared, a SOAP client must transmit the message to the Corticon Server (deployed as a Web Service) via HTTP.

If your SOAP tools have the ability to assemble a compliant request message from a WSDL you generated in Corticon Studio, then it is very likely they can also act as a SOAP client and transmit the message to a Corticon Server deployed to a supported application server. The Web Services section describes how to test this method of invocation with a third-party tool or application as a SOAP client.

When developing and testing SOAP messaging, it is very helpful to use some type of message interception tool (such as **tcpTunnelUI** or **TCPTrace**) that allows you to grab a copy of the request message as it leaves the client and the response message as it leaves Corticon Server. This lets you inspect the messages and ensure no unintended changes have occurred, especially on the client-side.

