

What's New in Corticon

Notices

© 2016 Progress Software Corporation and/or one of 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, Deliver More Than Expected, Icenium, Kendo UI, Making Software Work Together, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Business Making Progress, Progress Software Developers Network, Rollbase, RulesCloud, RulesWorld, SequeLink, Sitefinity (and Design), SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test 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, BravePoint, BusinessEdge, DataDirect Spy, DataDirect SupportLink, Future Proof, High Performance Integration, OpenAccess, ProDataSet, Progress Arcade, Progress Profiles, Progress Results, Progress RFID, Progress Software, ProVision, PSE Pro, SectorAlliance, Sitefinity, 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.

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 Studio 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. The 64-bit Studio is recommended because it provides better performance when working on large projects.

Note: When adding Corticon to an existing Eclipse, the target Eclipse must be an installation of the same bit width. Refer to the *Corticon Installation Guide* for details about integrating Corticon Studio into an existing Eclipse environment.

- **Corticon Servers** implement web services and in-process servers for deploying business rules defined in Corticon Studios:
 - **Corticon Server for 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.
 - **Corticon Server for .NET** facilitates deployment of Corticon Decision Services on Windows .NET Framework and Microsoft Internet Information Services (IIS).
 - **Corticon Web Console** enables administration of multiple remote Corticon Servers. A Web Console server is deployed into a Progress Application Server that it installs, and then accessed by users through authenticated web browser connections.

Use with other Progress Software products

Corticon releases coordinate with other Progress Software releases:

- [Progress OpenEdge](#) is available as a database connection. You can read from and write to an OpenEdge database from Corticon Decision Services. When Progress Developer Studio for OpenEdge and Progress Corticon Studio are integrated into a single Eclipse instance, you can use the capabilities of integrated business rules in Progress OpenEdge. See the OpenEdge document [OpenEdge Business Rules](#) for more information. OpenEdge is a separately licensed Progress Software product.
- [Progress DataDirect Cloud](#) (DDC) enables simple, fast connections to cloud data regardless of source. DataDirect Cloud is a separately licensed Progress Software product.
- [Progress RollBase](#) enables Corticon rules to be called from Progress Rollbase. Rollbase is a separately licensed Progress Software product.

What's new and changed in Corticon 5.5.2

This section summarizes the new, enhanced, and changed features in Progress[®] Corticon[®] 5.5.2, and provides links to those sections of the documentation.

For details, see the following topics:

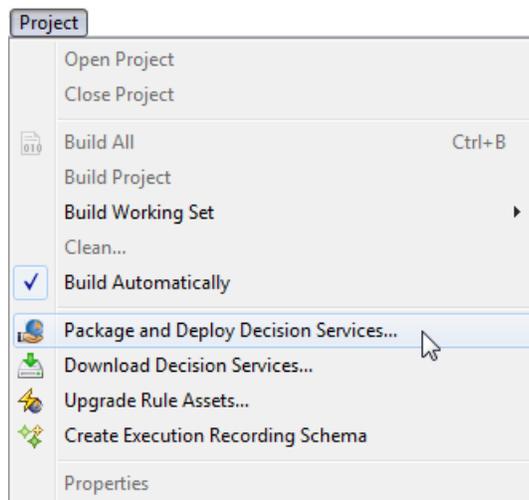
- [Deployment of Decision Services into Web Console Applications from Studio](#)
- [Server management utility for deploying Decision Services through the Web Console](#)
- [Managing in-process servers in the Web Console](#)
- [Servers can use custom context URLs that are managed in the Web Console](#)
- [Creating a rulesheet by importing an Excel worksheet](#)
- [Introducing Rule Execution Recording](#)
- [Specifying Ruletest target date against deployed Decision Services](#)
- [Performing logical analysis on Ruleflow branch containers](#)
- [Rulesheet option can show or hide vocabulary details](#)
- [Limiting tables in database metadata import](#)
- [Clearing database mappings](#)
- [EDC support for MySQL](#)
- [REST implementation supports JSONObject.NULL](#)
- [New EDC Tutorials and revised EDC documentation](#)
- [Joining the Web Console Customer Experience Improvement Program](#)

Deployment of Decision Services into Web Console Applications from Studio

This release introduces the ability to deploy from Studio to servers managed by a Web Console. While the Studio's Publish wizard enables compiling a Ruleflow into a Decision Service to be staged locally or deployed to a running Server, this feature enables deploying one or more Ruleflows into an Application on a Web Console server. Servers and server groups that are hosting the application immediately deploy (or redeploy) the Decision Services to all running servers.

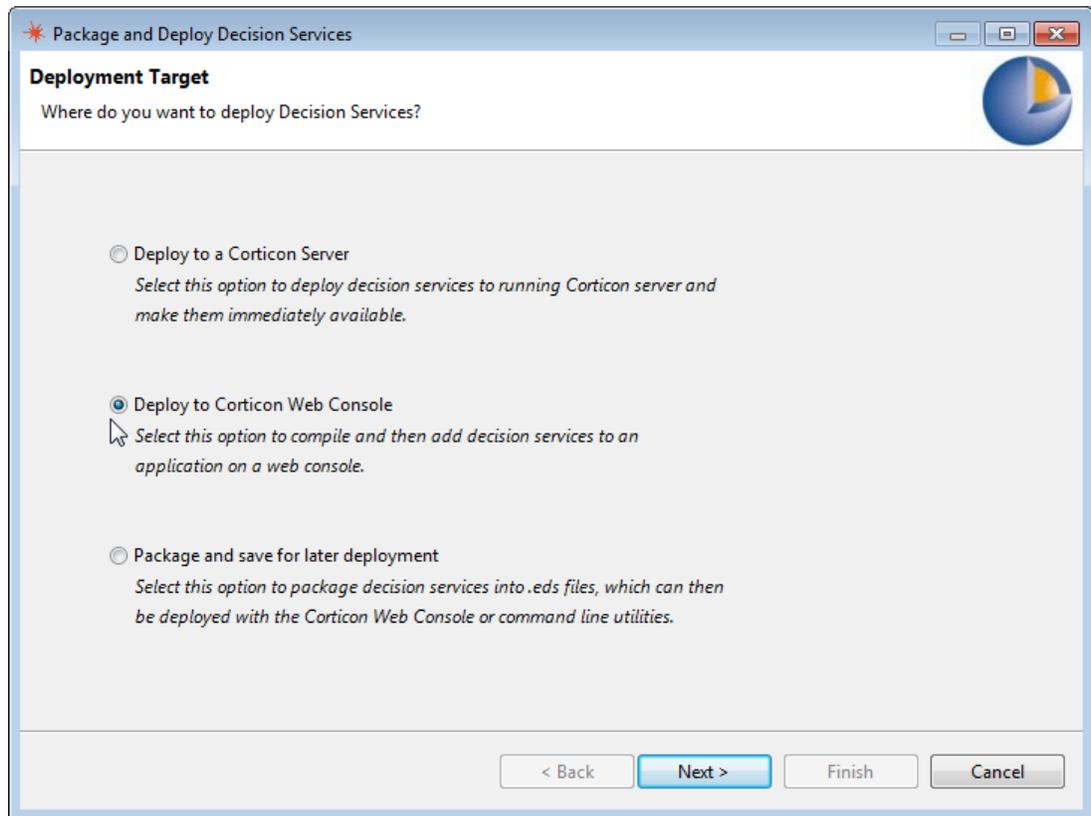
To deploy Ruleflows in Corticon Studio as Decision Services on servers managed by the Web Console:

1. Confirm that the Web Console server you want to use is running. Also confirm that the servers that will run the deployed Decision Services are running.
2. In Corticon Studio, choose **Project > Package and Deploy Decision Services**:

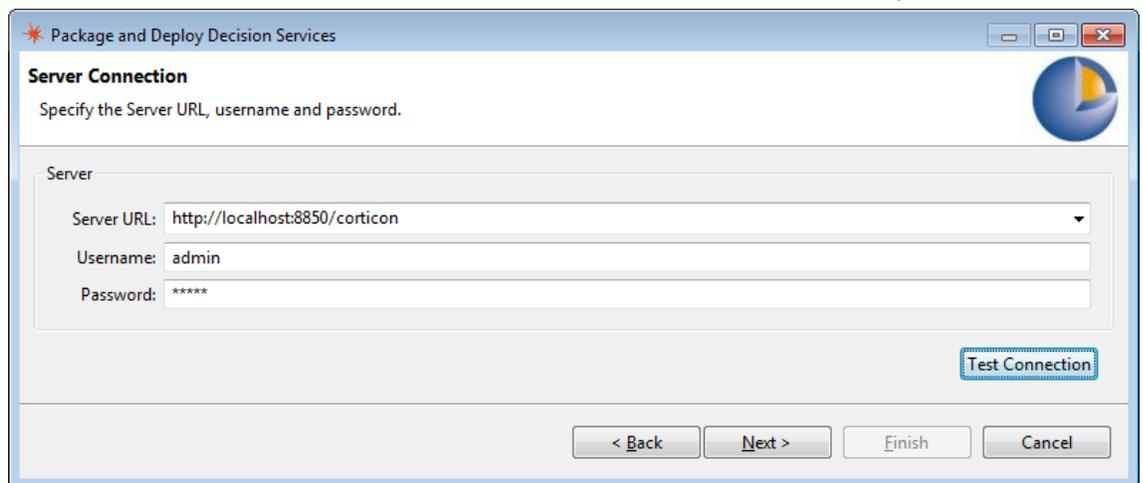


When a project is selected or there is an active file in its editor, the Ruleflows of only that project will be listed. When no projects are selected and no files are in their editor, the Ruleflows of all projects in the workspace will be listed.

3. In the **Package and Deploy Decision Services** dialog, choose the deployment target **Deploy to Corticon Web Console**.

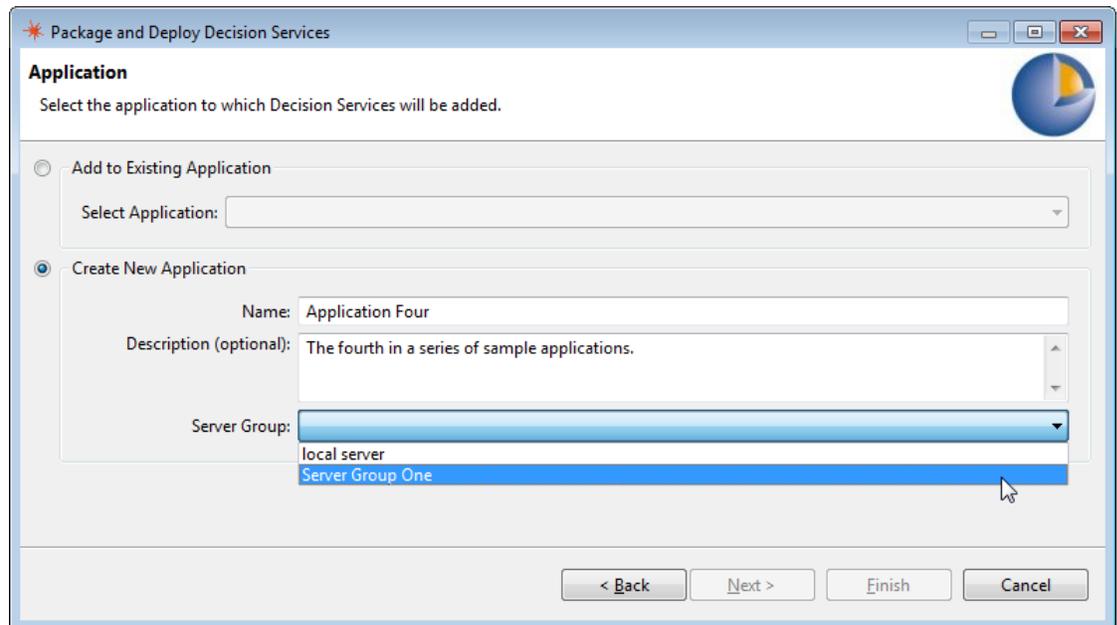


4. Click **Next**.
5. Enter the server connection URL with its port and `/corticon`, then the username and password for that Web Console. The administrative username is `admin` with the initial password `admin`.



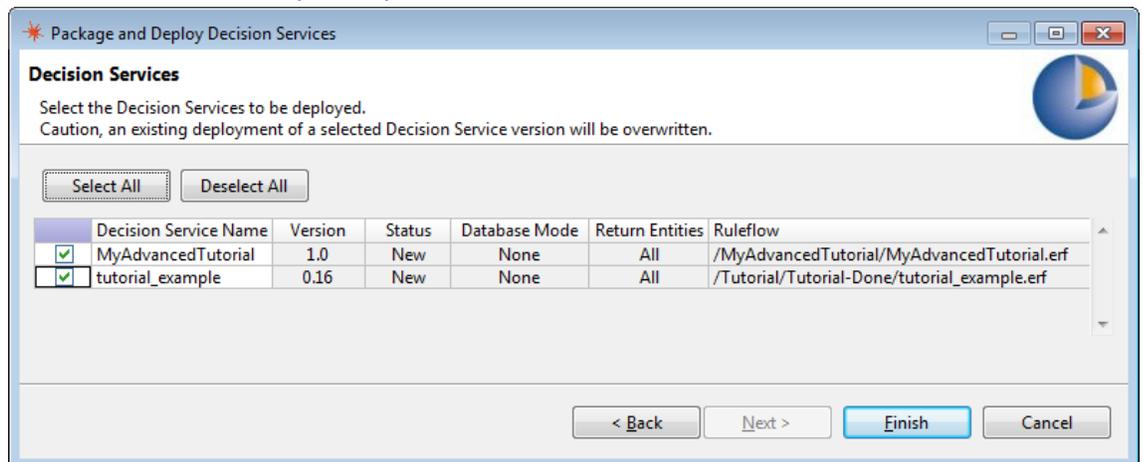
The connection information is persisted locally, so that it can be offered for subsequent publishing to known Web Console locations.

6. Select whether to use an existing Application or to create a new one:
 - To add to an existing Application, choose **Add to Existing Application**, select an Application on the pull-down list, and then click **Next**.
 - To create a new Application, choose **Create New Application**, and then enter a new Application name and its description.



In the **Server Group**'s dropdown menu, choose the server or server group that will host the Application, and then click **Next**.

7. The **Decision Services** panel opens:



Select the Ruleflows to deploy as Decision Services. You can edit the **Decision Service Name** to make it a distinct deployment even though the same Ruleflow Version might already be deployed under another name.

Note: When deploying EDC-enabled Decision Services you must set Database Mode to **Read Only** or **Read/Update** for the Decision Services to access the database once deployed. Your Corticon server must be licensed for EDC.

When your selections are complete, click **Finish**.

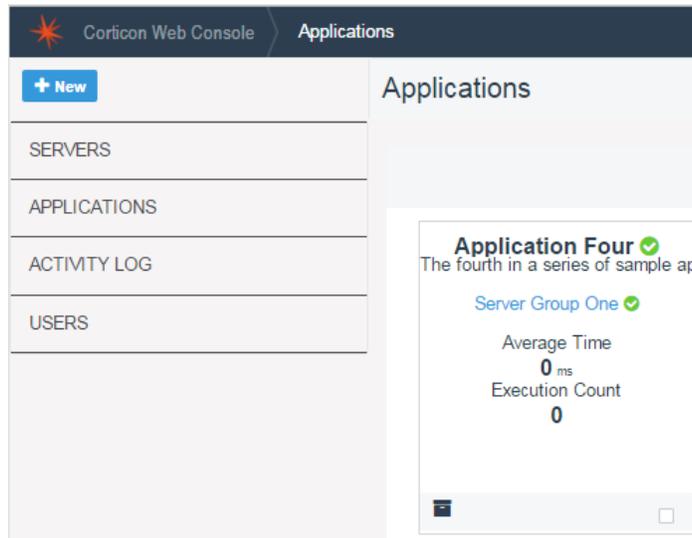
The wizard then compiles the Ruleflows locally, creates a new Application (if required) on the Web Console, and then adds (or updates) the Decision Services in the Application. Then, the Application is updated automatically to deploy/update the Decision Services in Servers and all active server members in Server Groups hosting the Application.

This material has been added as the topic *"Compiling and deploying to a Corticon Web Console"* in the *Integration and Deployment Guide*.

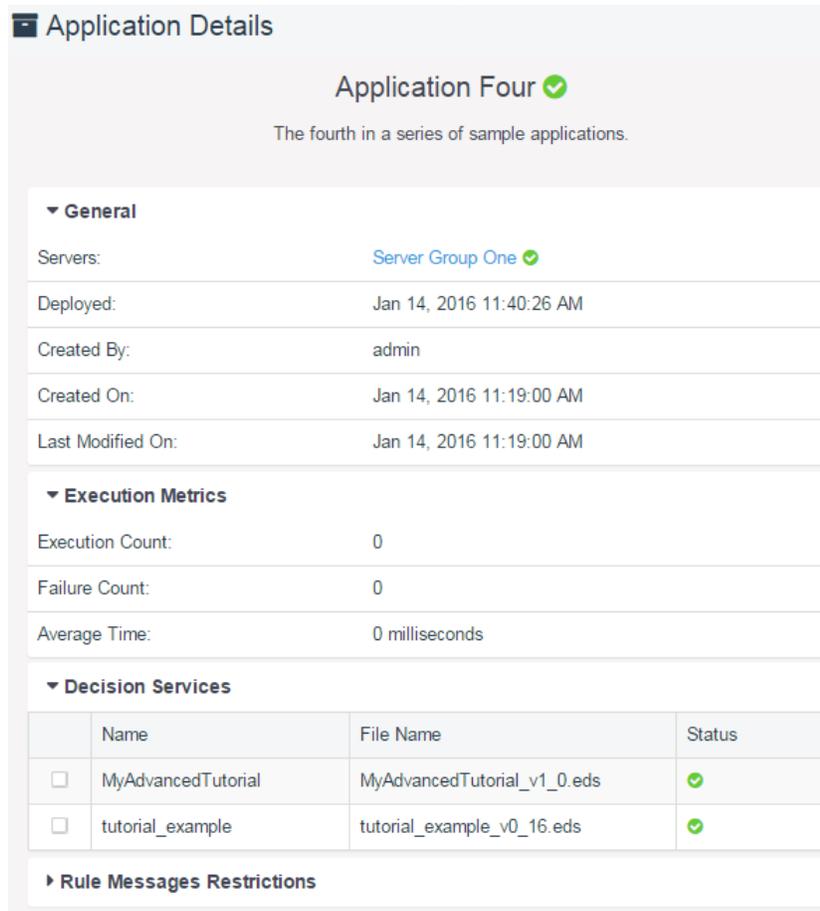
Results of the publishing process in the Web Console

After choosing to create a new application as described, the Web Console has the following information:

- Application Four was created and deployed into Server Group One:



- Application Four includes the two newly created Decision Services:



- Both Decision Services are deployed on all four servers in Server Group One:

The screenshot shows the Corticon Web Console interface. The top navigation bar includes 'Corticon Web Console', 'Servers', and 'Server Group: Server...'. The main content area is titled 'Server Group Details' and shows 'Server Group One' with a green checkmark. Below this, there is a section for 'Servers' with a table listing four servers, all with a status of 'Connected'. The table has columns for IP Address, Port, Context URL, and Status.

	IP Address	Port	Context URL	Status
<input type="checkbox"/>	nb1	8850	axis	✔ Connected
<input type="checkbox"/>	nb1	80	axis	✔ Connected
<input type="checkbox"/>	nb5	8850	Java_UAT	✔ Connected
<input type="checkbox"/>	nb5	8850	axis	✔ Connected

Below the servers table, there is a section for 'Execution Metrics' and a section for 'Decision Services (2)'. The 'Decision Services' section contains a table with columns for Name, Version, Execution Count, Average Time, and Deployed on.

Name	Version	Execution Count	Average Time	Deployed on
MyAdvancedTutorial	1.0	0	0 ms	nb1:80/axis nb5:8850/Java_UAT nb1:8850/axis nb5:8850/axis
tutorial_example	0.16	0	0 ms	nb1:80/axis nb5:8850/Java_UAT nb1:8850/axis nb5:8850/axis

Server management utility for deploying Decision Services through the Web Console

This release introduces the ability to make scripted calls to the Web Console on Windows and Linux to deploy Decision Services. This combined with ability to build and test Decision Services from a script allow you to automate the deployment of your Decision Services. The scripting is a command line utility that makes REST calls to the Web Console to perform actions, and then returns codes that identify success or failure.

The utility, `corticonWebConsole.bat`, is included on all server installations, and is located at `[CORTICON_HOME]/Server/bin`. The general format of this utility's commands is:

```
corticonWebConsole {command} {command options}
```

Commands supported in this utility are as follows, many parameters showing both their short form and long form:

corticonWebConsole -help

```
usage: corticonWebConsole
-application,--application    Perform actions on an application
-ds,--decisionservice        Perform actions on a decision service
-h,--help                    print this message
-login,--login               initiate login to web console
-logout,--logout             initiate logout from web console
```

Lists the syntax of the commands.

corticonWebConsole -login

You must first login to the Web Console before the other commands can be applied.

```
usage: login
-h,--help          print this message
-n,--name <username> Login username for the web console
-p,--password <password> Login password for the web console, if omitted
                    password will be requested through standard
                    input.
-u,--url <url>     Url leading to the web console e.g
                    http://localhost:8850/corticon
-v,--verbose       Include debugging output
```

Authenticates the user on the specified Web Console server. No other commands have any effect until this command executes successfully. Choosing to omit the password will prompt for its entry through standard input.

Note: The login command stores an encrypted login token in your work directory so that, when you later perform commands, you can do so without logging in. When using a batch process to perform deployment, you will need to have this login token available in the work directory of the Corticon install used by the batch process.

corticonWebConsole -logout

```
usage: logout
-h,--help          print this message
-v,--verbose       Include debugging output
```

The logout command closes the connection to the Web Console.

corticonWebConsole -ds

```
usage: decisionservice
-add,--add         Add a decision service to the web console
-delete,--delete   Remove a decision service from the web console
-h,--help         print this message
```

corticonWebConsole -ds -add

```

usage: add
  -application,--application <application name>  Name of application that
                                                    the decision service
                                                    will be added to
  -dbaccessmode,--dbaccessmode <mode>           Database access mode to
                                                    use [optional]
  -dbproperties,--dbproperties <properties file> Path to database
                                                    properties file
                                                    [optional]
  -dbreturnmode,--dbreturnmode <mode>           Database access return
                                                    entities mode to use
                                                    [optional]
  -deploy,--deploy                                When this flag is set
                                                    the decision service
                                                    will be deployed after
                                                    being added to the
                                                    application [optional]
  -enablecache,--enablecache                    When this flag is
                                                    present, database
                                                    caching will be enabled
                                                    [optional]
  -f,--file <eds file>                          Path to decision service
                                                    eds file to be added
  -h,--help                                       print this message
  -maxpool,--maxpool <max pool size>            Maximum pool size
                                                    [optional]
  -n,--name <name>                               Name given to the
                                                    decision service to be
                                                    added
  -overwrite,--overwrite                        When this flag is
                                                    present, the decision
                                                    service will overwrite
                                                    an existing decision
                                                    service in the
                                                    application with a name
                                                    matching that of the
                                                    value of the name
                                                    argument. When this flag
                                                    is not present, and the
                                                    decision service exists
                                                    the deploy will fail.
                                                    [optional]
  -v,--verbose                                   Include debugging output
                                                    [optional]
  -xmlstyle,--xmlstyle <xmlstyle>              XML message style, ether
                                                    null, FLAT, or HIER

```

Adds the specified Decision Service to the specified application.

The database options (`--dbproperties`, `--dbaccessmode`, and `--dbreturnmode`) are used when the Decision Service is configured for EDC database connectivity.

Adding `--deploy` will deploy the specified Decision Service to each Server or Server Group that includes the specified application.

Adding `--overwrite` to the command will replace a corresponding Decision Service that exists.

corticonWebConsole -ds -delete

```
usage: delete
  -application,--application <application name>  Name of application that
                                                    the decision service will
                                                    be removed from
  -h,--help                                       print this message
  -n,--name <name>                               Name given to the
                                                    decision service to be
                                                    added
  -undeploy,--undeploy                          When this flag is set,
                                                    the decision service will
                                                    be undeployed after being
                                                    removed from the
                                                    application [optional]
  -v,--verbose                                   Include debugging output
                                                    [optional]
```

Removes a specified Decision Service from the Web Console server completely.

Adding `--undeploy` will undeploy the Decision Service from each Server or Server Group that includes the specified application.

corticonWebConsole -application

```
usage: application
  -deploy,--deploy      Deploy the specified application to its associated
                        server/server group
  -h,--help             print this message
  -undeploy,--undeploy Undeploy the specified application to its
                        associated server/server group
```

corticonWebConsole -application -deploy

```
usage: deploy
  -h,--help      print this message
  -n,--name <name>  Name given to the application to be deployed
  -v,--verbose     Include debugging output [optional]
```

Deploys the specified application to its associated servers/server groups.

corticonWebConsole -application -undeploy

```
usage: undeploy
  -h,--help      print this message
  -n,--name <name>  Name given to the application to be undeployed
  -v,--verbose     Include debugging output [optional]
```

Undeploys the specified application from its associated servers/server groups.

This material has been added as the topic *"Deploying a Decision Service"* in the *Integration and Deployment Guide*.

Managing in-process servers in the Web Console

Corticon's Web Console can now manage in-process deployments of Corticon Servers. This allows you to use the Web Console to view performance metrics and manage Decision Services that are running in an in-process Corticon Server. By default, in-process Corticon Servers cannot be managed by the Web Console. To enable this you need to modify your code where you instantiate the Corticon Server as follows:

```
CcServerFactory.getCcServer(true)
```

Specifying `true` instructs Corticon that the CcServer is to allow a Web Console to connect to it.

The default port and logging values are read from the `CcServer` properties file, and can be overridden by adding lines to a Java server's `brms.properties` with your preferred values:

```
com.corticon.server.embed.managementPort=9850
com.corticon.server.embed.managementWebLogDirectory=%CORTICON_WORK_DIR%/logs
com.corticon.server.embed.managementWebLogLevel=FINE
```

or by editing a .NET server's `[CORTICON_HOME]\Server.NET\samples\conf\CcServer.properties` file.

Try it!

The following examples let you experience the management feature as implemented in our samples.

- In a Java server installation, launch the Java in process server test script, `[CORTICON_HOME]/Server/bin/testServer.bat`.
- In a .NET server installation, launch the .NET in process test executable, `[CORTICON_HOME]\Server.NET\samples\bin\Corticon-Api-Inprocess-Test.exe`
- Connect your browser to a Web Console, and then create a Server Group that includes the in-process server on port 9850. When you click refresh, the in-process server shows that it is managed through the port you specified, as shown:

Server Group Details

In-Process Servers ✔

Demonstration of managing a Corticon in-process server.

▼ Servers

	IP Address	Port	Context URL	Status
<input type="checkbox"/>	localhost	9850	axis	✔ Connected

▼ Execution Metrics

Execution Count:	10453
Failure Count:	0
Average Time:	2.38 milliseconds

▼ Decision Services (4)

Name	Version	Execution Count	Average Time	Deployed on
ProcessOrder	1.10	10453	2.38 ms	localhost/axis:9850
Candidates	1.14	0	0 ms	localhost/axis:9850
Cargo	1.0	0	0 ms	localhost/axis:9850
AllocateTrade	1.14	0	0 ms	localhost/axis:9850

▼ Server Group Statistics

● Live Data
Hour
12 hours
1 day
7 days

Servers Response and Execution

■ Average Time
 ■ Max. Time
 ■ Min. Time
 ■ # Executions
 ■ # Failures

This information modified and extended the topic "Server Groups and Servers" in the [Corticon Server: Web Console Guide](#).

Servers can use custom context URLs that are managed in the Web Console

When deploying more than one instance of the Corticon server to an application server, you need to provide a unique name for the Java Web Archive, `axis.war`, or .NET application, `axis`. When adding servers to the Web Console you are no longer limited to the name `axis` - you can now specify a custom name as part of the connection parameters for the server.

In the following example, the first two servers are on machine 1, the Java Server on port 8850 and the .NET (IIS) Server on port 80. The next two servers are colocated on machine 5, both on port 8850, each under its own context URL:

Server Group One ✔

▼ **Servers**

	IP Address	Port	Context URL	Status
<input type="checkbox"/>	nbbedgsaintma1	8850	axis	✔ Connected
<input type="checkbox"/>	nbbedgsaintma1	80	axis	✔ Connected
<input type="checkbox"/>	nbbedgsaintma5	8850	Java_UAT	✔ Connected
<input type="checkbox"/>	nbbedgsaintma5	8850	axis	✔ Connected

For more information, see the topic "*Creating custom context URLs on a web server*" in the *Corticon Installation Guide*.

This information also modified and extended topics in the [Corticon Server: Web Console Guide](#).

Creating a rulesheet by importing an Excel worksheet

This release enables importing of an Excel spreadsheet to create a Rulesheet.

What's new

Often decisions are based on statistical data in tables that are actuarial or real-world data that cannot be derived through a formula. Insurance underwriting, manufacturing tooling, life sciences, and census data have examples of data that is key to decision making yet revised only at regular time intervals.

When used in Corticon, a Rulesheet that provides such data is very useful but required hours of data entry to create and to update what might be a very large number of rules. Until now.

Consider the following excerpt of an Excel worksheet as an example for life insurance underwriting. Column A contains headers that identify the customer's age, and row 1 contains headers that identify the beneficiary's age. The cells are the policy factor that adjusts the policy premium:

	A	B	C	D
1	Dimensions	50	51	52
2	40	0.9680	0.9698	0.9715
3	41	0.9652	0.9670	0.9689
4	42	0.9618	0.9640	0.9660

When this two-dimensional information is imported into Corticon Rulesheet, each of the two dimensions and the data area are assigned to Vocabulary attributes. Then, each of the two dimensions defines a Condition and the data point specifies an Action, as shown:

Conditions		0	1	2	3
a	Customer.age		50	51	52
b	Policy.beneficiaryAge		40	40	40

Actions		'''		
Post Message(s)				
A	Policy.factor	0.968	0.9698	0.9715

As the size of the data range expands, an imported Excel sheet could create thousands of rules in the Rulesheet that were all created by one simple import action, and later updated by performing the same import from the revised worksheet.

The sample data file is located in a Studio installation at `[CORTICON_WORK_DIR]\Samples\ImportExcel\SampleDimensionalData.xlsx`. See the topic *"Creating rules by importing an Excel worksheet"* in the *Rule Modeling Guide* for more information.

What's changed

A new option, **Generate Rulesheet from Excel**, was added to the **File > Import** dialog's **Progress Corticon** section.

Introducing Rule Execution Recording

Corticon Decision Service execution is stateless, where all state is maintained in the message payloads. A single incoming message payload seeds the engine's working memory after which rules processing commences. Once rules processing is complete, the final state of the engine's working memory is returned as a response. The response message payload includes two discrete sets of data: data payload and posted messages that can each include data values as well internal information about the sequence of rules firing. While some data might have been stored in a database through an Enterprise Data Connection, the complete message payload is flushed from memory.

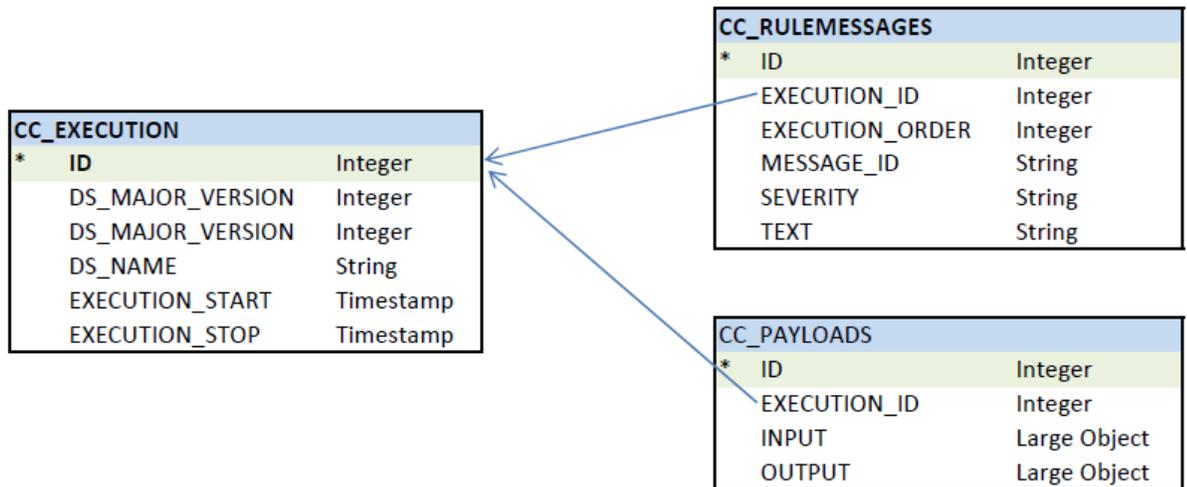
There are several reasons to want to retain payloads and rule messages:

- Auditors might be assigned to determine how certain results were derived from a rule processing. When the exact request and its response payload can be accessed, the auditors can precisely reconstruct a decision.
- Rules always evolve. Developers of rules can replay a 'real' request to see how rule changes impact responses as well as performance.
- Rule modelers can check rule coverage in a large set of 'real' rule messages to see if rules considered obsolete are really not getting any use.
- Rule modelers can also track down common sequences of rule activities to reveal race conditions or re-entrant rules.

Corticon's Rule Execution Recording feature records all input payloads sent to a Decision Service, all the rule messages produced processing the payloads, and the returned payloads. This provides documentation of the activity of a Decision Service. When setup and activated at the server level, each Decision Service deployed on the server must opt in to use the server's rule execution recording mechanism.

Overview of schema for storing execution payloads and rule messages

The general pattern of the schema for the necessary tables in a database you create are as follows:



Within the schema, each Decision Service version execution is timestamped with its time interval, and provided a unique primary key. Payloads and rule messages are in separate tables, each with a unique primary as well as the foreign key to link it to its execution identifier.

Consider:

- The database you use for rule execution recording could be, and typically should be, distinct from the enterprise database of record. Rule execution recording can be used independent of Corticon EDC.
- Multiple Corticon servers could connect to and record execution in the same database.
- Where rule tracing has been implemented, rule messages will be prepended with the metadata of the rule that fired. (Enabled by uncommenting the `brms.properties` line `#com.corticon.reactor.rulestatement.metadata=false` and then setting it to `true`)
- The process is asynchronous "fire-and-forget" from Corticon Server. Data that accrues from this feature in your database is entirely your responsibility. You must provide adequate performance, storage, backup, rollover, and reporting mechanisms. Correspondingly, Corticon Servers and Studio does not read from the execution recording database at any time.
- This feature is not supported on Studio's embedded server. When a remote server is used as the Studio's test subject, that server could be performing execution recording.

Creating the database schema for Rule Execution Recording

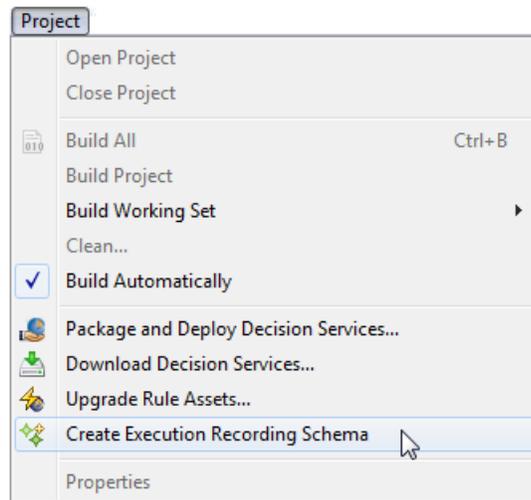
The first step is to setup the schema in your preferred database. Corticon Studio provides a wizard that lets you:

- Define and test the connection to the target database
- Create the default schema for the database brand in the target database
- Define the properties and encrypted credentials that a server will add to its `brms.properties` file to connect to that database

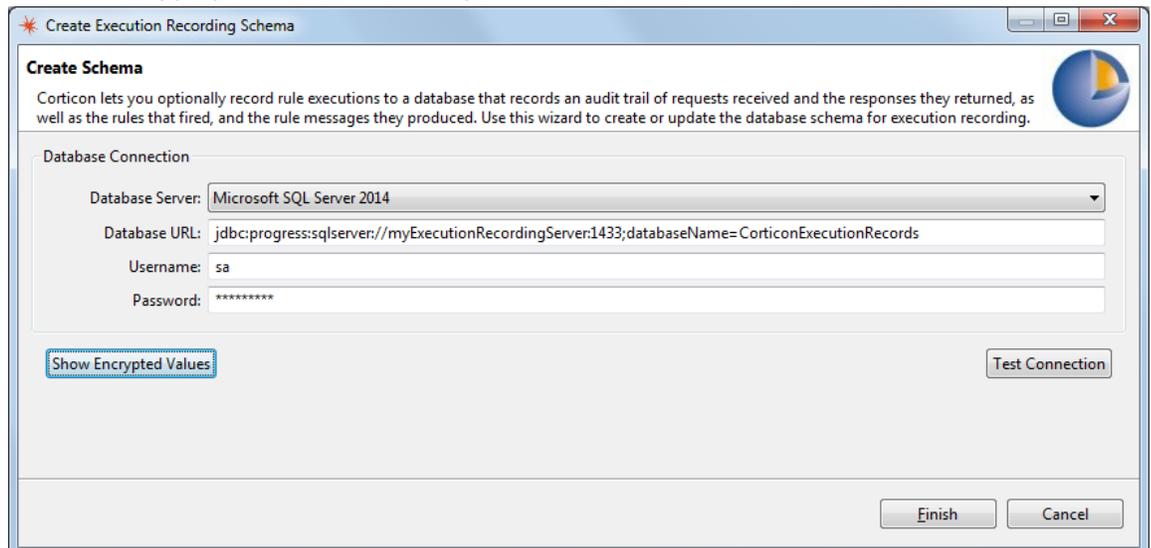
Note: Your database administrator might prefer to create the schema by using (or adapting) the SQL script samples supplied in a Corticon server installation at `[CORTICON_HOME]/Server/src/sql`. The database connection parameters can also be created entirely in a server's `brms.properties` file, although the credentials would have to be in plain text as the decryption algorithm requires that the credentials (either or both username and password) are encrypted by the algorithm applied in Studio.

To test and generate the connection parameters

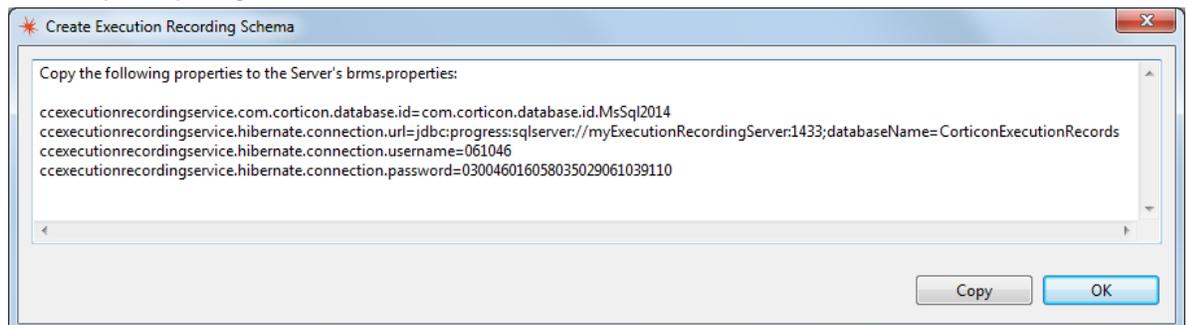
1. In the target database's administrative tool, locate or create the named database instance for recording executions.
2. In Studio, choose the menu command **Project > Create Execution Recording Schema**, as shown:



3. In the **Create Execution Recording Schema** dialog, choose the **Database Server** brand and version brand from the dropdown list. Then edit the **Database URL** that is displayed to change `<server>:nnnn` to your server *host:port*, and `<database_name>` to the name you created. Then enter appropriate username and password credentials for that database.



4. Click **Test Connect** to ensure that the information achieves connection.
5. Click **Show Encrypted Values** to get the connection properties and values that you will move to each participating server.



6. Click **Copy** to put the connection information on the clipboard, click **OK**. Paste the information into a text file that will be transferred to Server locations for inclusion in their `brms.properties`.

Note: This completes the testing of the connection information and creation of encrypted credentials. You can continue to actually create the schema using the default scripts for the specified database.

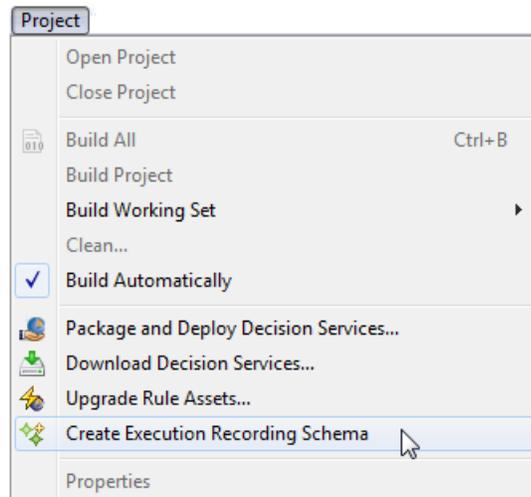
7. Click **Cancel**.

To create the schema from Studio

1. Edit Studio's `brms.properties` to add the line:

```
com.corticon.server.execution.recording.enabled=true
```

2. Restart Studio, and then choose the menu command **Project > Create Execution Recording Schema**, as shown:



3. Enter and test the connection information as described in the previous procedure.
 4. Click **Test Connect** to ensure that the information achieves connection.
 5. **Note:** If the schema exists, its tables and existing data will be deleted, and then the tables recreated.
-

To create the schema, click **Finish**.

6. It is a good idea to shut off the ability to recreate the schema once it has been created. To do that, edit Studio's `brms.properties` to change the line you added to:

```
com.corticon.server.execution.recording.enabled=false
```

Configuring Corticon Servers to store rule messages that they execute

On each server you must set a few properties in the `brms.properties` file to enable recording of rule executions:

1. Enable the Server to instantiate Rule Execution Recording during startup by setting this property to true:

```
com.corticon.server.execution.recording.enabled=true
```

2. Once enabled, both payloads and rule messages will be emitted. You can shut off either one by changing its property setting to false:

```
com.corticon.server.execution.recording.process.payloads=false  
com.corticon.server.execution.recording.process.rulemessages=false
```

3. Paste the text of the connection information then delete (or comment out) the first line.

```
### EXAMPLE: Copy the following properties to the Server's brms.properties:  
  
ccexecutionrecordingservice.com.corticon.database.id=com.corticon.database.id.MsSql2014  
ccexecutionrecordingservice.hibernate.connection.url \\  
    =jdbc:progress:sqlserver://myExecutionRecordingServer:1433; \\  
    databaseName=CorticonExecutionRecords  
ccexecutionrecordingservice.hibernate.connection.username=061046  
ccexecutionrecordingservice.hibernate.connection.password=030046016058035029061039110
```

Note: Either or both the username and password could be entered as plain text instead of encrypted values.

Turning message recording on for individual Decision Services

With the schema set up in the target database, the intent to record enabled on the server, and the database connection tested, all that needs to be done is to set the properties on each Decision Service that will participate in this feature. By default, no Decision Service will use this feature. There are two ways to enable a Decision Service to enable execution recording:

- **Execution properties in a manually edited CDD file** - Add the following line to the `options` in a `decisionService` section of the CDD file: `<option name="PROPERTY_EXECUTION_RECORDING_SERVICE_ENABLED" value="true"`
- **CcServer API methods** - Use `setDecisionServicePropertyValue` for the `DecisionServiceName` to set the static property `PROPERTY_EXECUTION_RECORDING_SERVICE_ENABLED` and the property value `true`.

Once established on a Decision Service, you can turn off execution recording by changing the setting to `false`.

This material was added as the topic *"Implementing Rule Execution Recording in a database" in the Integration and Deployment Guide*.

Specifying Ruletest target date against deployed Decision Services

What's new

When running a Ruletest against a deployed Decision Service, you can now specify a target date to test the effective date and version properties of the Decision Service. From the Studio point of view, such a server is often referred to as a *remote server*. By setting a target date, a test can execute as though it were sent at a specific date and time. By setting a target date, a test can execute as though it were sent at a specific date and time; either in the past or future.

To use a Decision Service on a remote server as the test subject:

1. With the Ruletest you want to run open in its Studio editor, choose the menu command **Ruletest > Testsheet > Change Test Subject**. The **Select Test Subject** dialog box opens:

Select Test Subject

Execute test in Corticon Studio

Execute test against a deployed Decision Service

Server URL:

Decision Services:

Name	Major	Minor	Effective Start Date	Effective Stop Date

Optional Overrides

Major Version:

Minor Version:

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

2. Click **Execute test against deployed Decision Service**.
3. For the **Server URL**, enter a URL; for example, a Corticon Server installed (and running) on the same machine as the Studio: `http://localhost:8850/axis`.
Click **Refresh** to populate the list of deployed Decision Services on that server.

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

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

5. If you don't need to specify a target date or version override, you can click **OK** at this point. Your test will use the selected Decision Service version.
6. When the selected Decision Service was deployed with a date range defined -- active from the effective date through the expiration date -- you can optionally to apply overrides to the test Decision Service's version or to simulate the Ruletest's call as occurring at a specific point in time.

Optional Overrides

Major Version:

Minor Version:

Effective Target Date: Time: : :

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

untitled_1
<http://localhost:8850/axis?name=Cargo,major version=1,effective target date=01/29/16 12:00:01 AM>

8. Run the Ruletest.

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

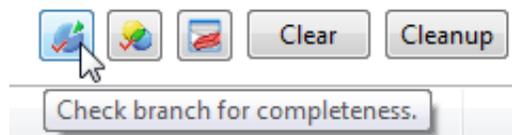
For more information, see "Choosing a test subject that is a deployed Decision Service" in the *Corticon Studio: Quick Reference Guide*.

Performing logical analysis on Ruleflow branch containers

A Ruleflow branch container is subject to two significant types of logical errors: *completeness* and *conflicts*.

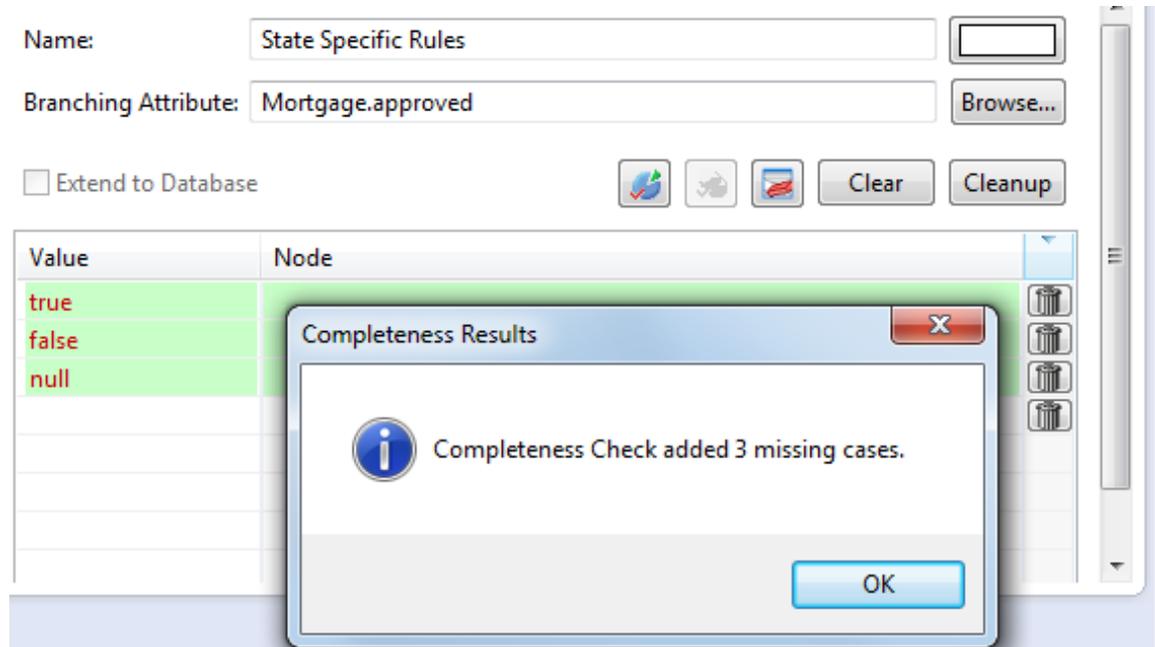
Completeness in a branch

A branch is complete when all of its possible values have been accounted for in branch nodes. When first defining branch activity, instead of selecting each possible value on each line, you can click **Check branch for completeness**, as shown:

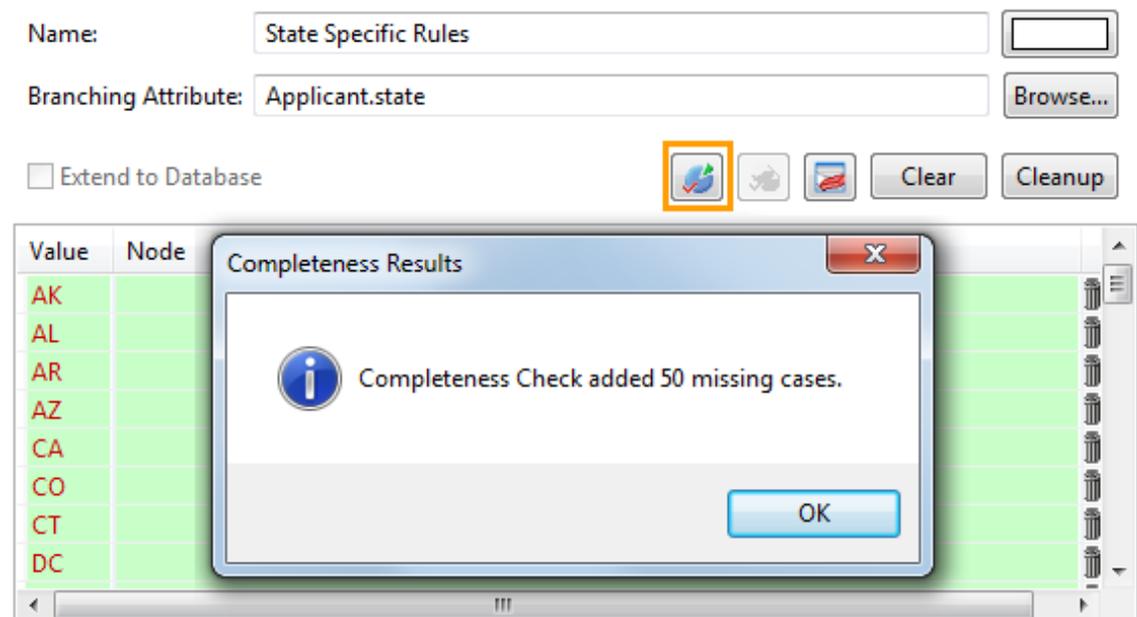


This will add all missing values as branch targets.

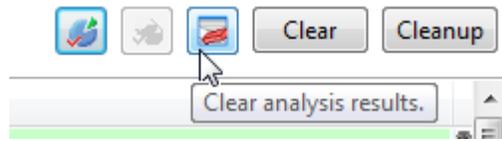
When branching by a Boolean attribute, three values are added, as shown:



When branching by an enumerated Custom Data Type attribute, each label in the enumeration is added, as illustrated:



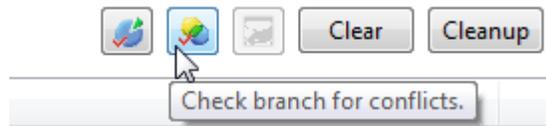
If the completeness check adds additional branch values, these will be highlighted in green. Clicking **Clear analysis results** removes color highlighting:



Assign nodes in the branch to appropriate listed value values. When you are done, click **Cleanup** to remove any branch values which do not have corresponding branch nodes. Unless you specify the keyword `other` as a branch value and assign it a branch node, your branch would be incomplete; you have not accounted for some of the possible branch values.

Conflicts in a branch

When branch nodes include logic that creates conflicts or ambiguities, those conflicts are difficult to identify. You can evaluate whether there are logical conflicts in a branch by clicking **Check branch for conflicts**, as shown:



Conflict or ambiguity in a Ruleflow branch container might be:

- **Different branches modify a shared entity** - You are informed of the attribute/association being modified.
- **A branch accesses the branch entity through an association that is not being filtered by the branch** - For example, the branch is on `Policy.type` while some rules act on `Customer.policy.type`. That creates a conflicting branch node, each of which is highlighted in red, as illustrated:

The screenshot displays the Corticon Rule Modeler interface. At the top, two rule editors are shown: 'DirectAccounts.ers' and 'PartnerAccounts.ers'. In 'DirectAccounts.ers', the 'Conditions' section has 'Policy.type' highlighted in orange. In 'PartnerAccounts.ers', the 'Conditions' section has 'Customer.policy.type' highlighted in orange. Below these is a diagram of an 'Accounts' container, which contains two sub-containers: 'DirectAccounts (Elite, Preferred)' and 'PartnerAccounts (Standard)'. At the bottom, the 'Properties' pane is open, showing 'Branch Activity' for 'Accounts'. The 'Branching Attribute' is set to 'Policy.type'. Below this, there is a table with the following data:

Value	Node
Elite	DirectAccounts
Preferred	DirectAccounts
Standard	PartnerAccounts

The 'Standard' and 'PartnerAccounts' row is highlighted in red, indicating a conflict.

Note: For more about this type of conflict, see the topic, *"How branches in a Ruleflow are processed"*.

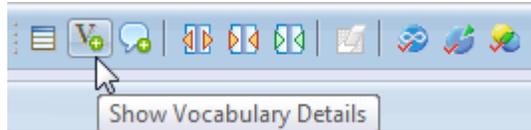
Click the **Clear analysis results** button to remove the highlights.

This information was added as the topics *"Logical analysis of a branch container"* in the *Rule Modeling Guide* and *"Ruleflow logical analysis checks"* in the *Quick Reference Guide*.

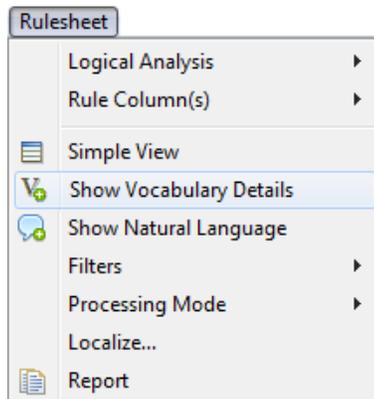
Rulesheet option can show or hide vocabulary details

This release introduces a new way that Vocabulary Details can be shown or hidden.

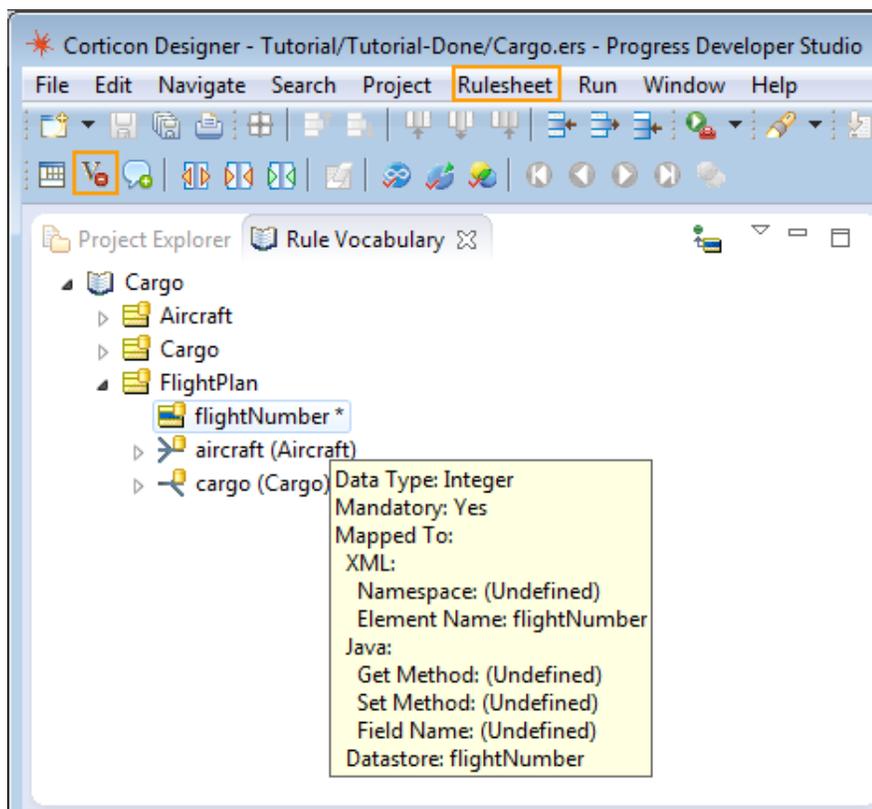
When working on a Rulesheet with the Vocabulary view active, the details of the Vocabulary hover help information might provide too much or too little information. While you could open the Vocabulary in its editor just to toggle the information to show or hide details, Rulesheets now provide an additional toggle to change the hover help view in the Vocabulary view, as shown in the toolbar:



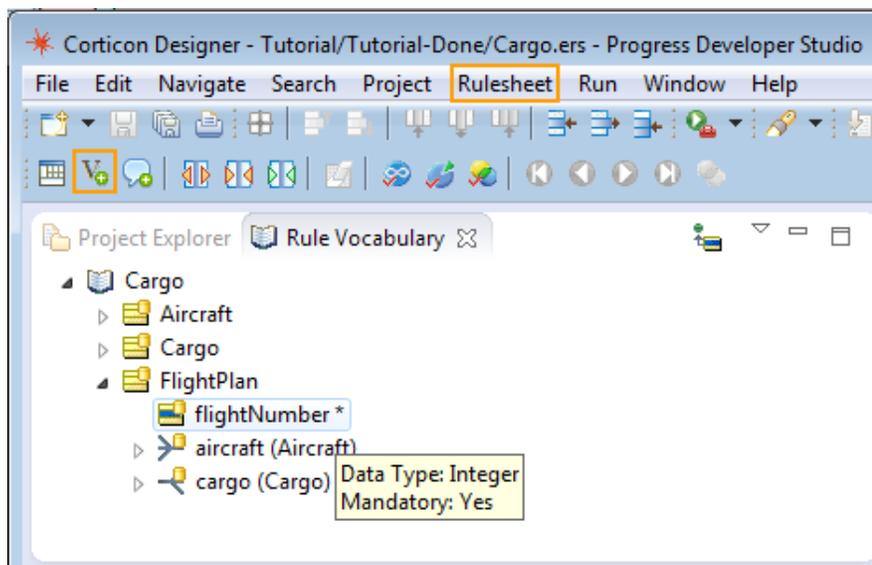
The menu items also offer the opportunity to toggle to the alternate view:



The following illustration shows Vocabulary information is showing its details. and the Rulesheet is the currently active menu and toolbar with the Vocabulary details button displaying its function to toggle to hiding the details:



When switched to hiding details, the Vocabulary details button displays its function to toggle to showing the details:

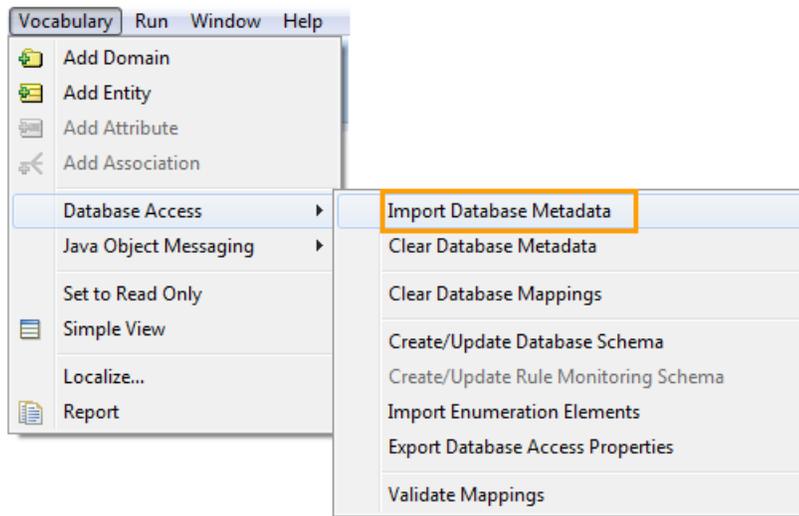


This material was added as the topic "Showing vocabulary details in a Rulesheet" in the *Corticon Studio: Rule Modeling Guide*, as well as menu and toolbar additions in the Vocabulary and Rulesheet topics in the *Quick Reference Guide*.

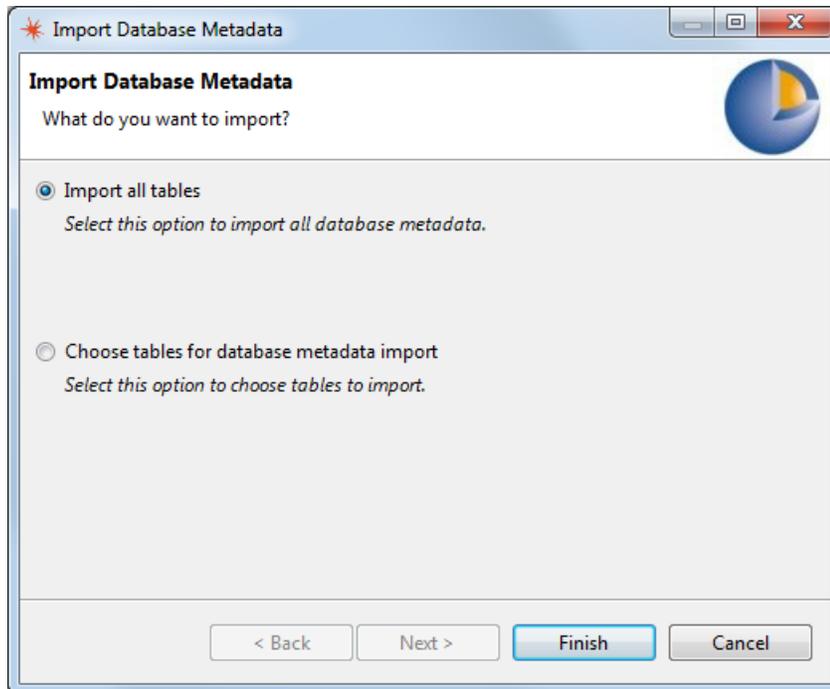
Limiting tables in database metadata import

When using EDC, you can now control the tables that are accessed to transfer metadata to the Vocabulary. When only a small subset of tables will supply the metadata that is needed, the time and space overhead of the process is reduced by choosing only the required tables.

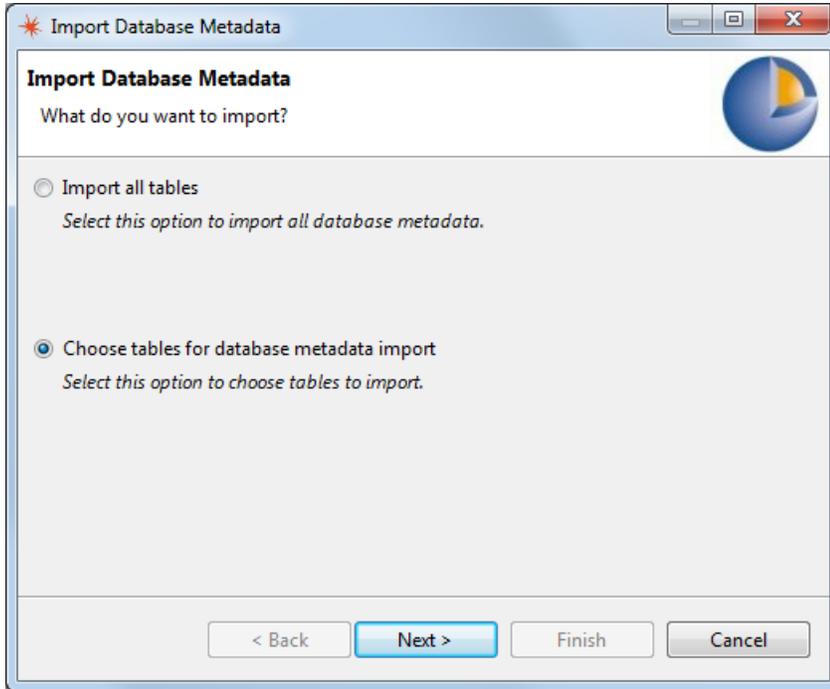
In the Vocabulary editor with a database connection established, select the menu command **Vocabulary > Database Access > Import Database Metadata**, as shown:



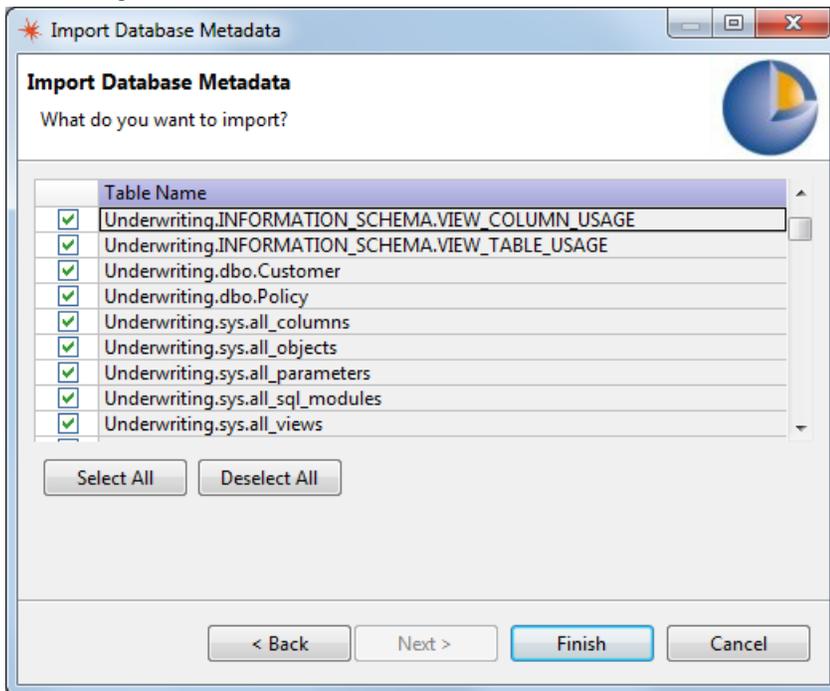
In the dialog, accept **Import all tables**, and click **Finish**, or...



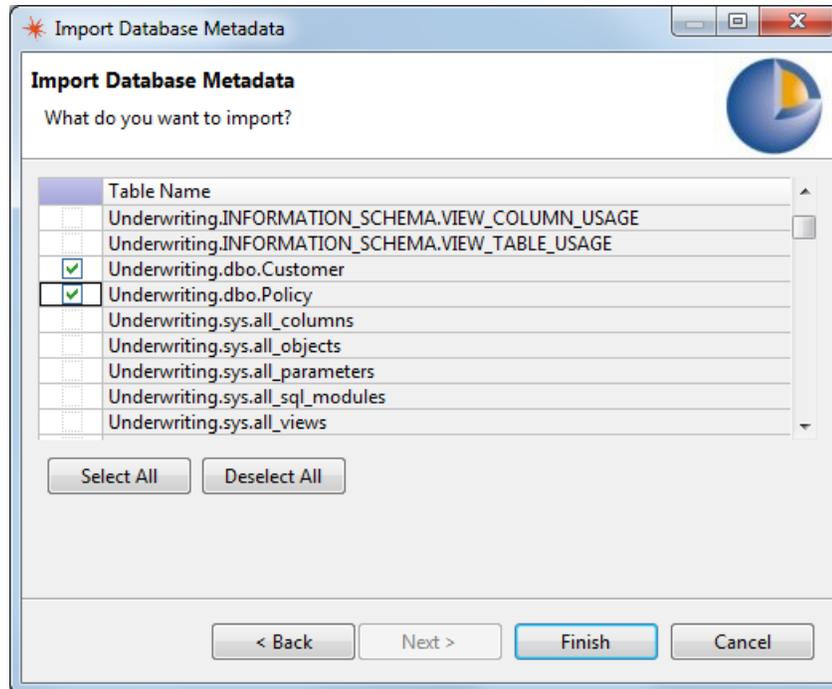
... click **Choose tables for database metadata import**, and click **Next**.



The dialog lists all the tables in the connected database.



Use **Select All**, **Deselect All**, and individual selection boxes to refine the preferred tables.



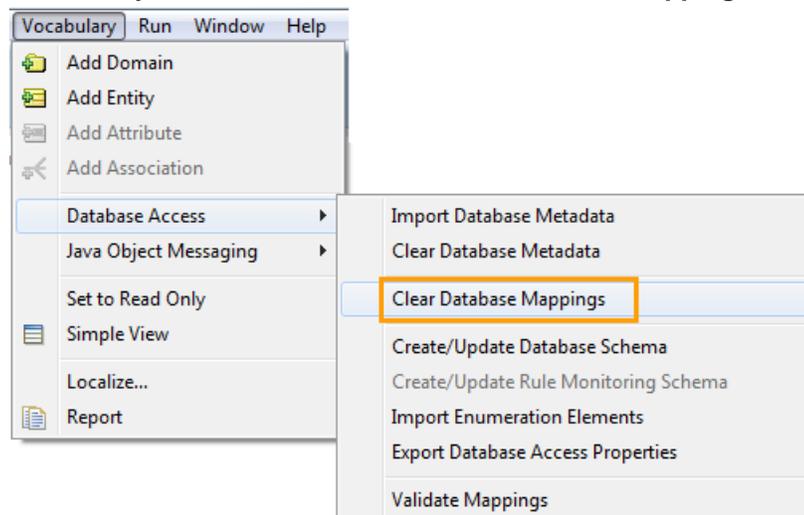
In this example, just two tables are selected. Click **Finish** to perform the task.

This material updated the topic *"Importing database metadata into a Vocabulary"* in the *Corticon Server: Integration and Deployment Guide*.

Clearing database mappings

After importing database metadata and manually adjusting the entries as required, clearing that data when appropriate can be a significant chore. The feature to clear database mappings performs this task in one step.

With the Vocabulary you want to change open in the Vocabulary editor, select the menu command **Vocabulary > Database Access > Clear Database Mappings**, as shown:



The highlighted Entity properties, as shown in the Cargo example, are cleared:

Property Name	Property Value
Entity Name	FlightPlan
Entity Identity	flightNumber
Inherits From	
XML Namespace	
XML Element Name	FlightPlan
Java Package	
Java Class Name	
Datastore Persistent	Yes
Table Name	Freight.dbo.FlightPlan
Datastore Caching	
Identity Strategy	
Identity Column Name	
Identity Sequence	
Identity Table Name	
Identity Table Name Colu...	
Identity Table Value Colu...	
Version Strategy	
Version Column Name	

The highlighted Attribute properties are cleared:

Property Name	Property Value
Attribute Name	flightNumber
Data Type	Integer
Mandatory	Yes
Mode	Base
XML Namespace	
XML Element Name	flightNumber
Java Object Get Method	
Java Object Set Method	
Java Object Field Name	
Column Name	flightNumber
Value Strategy	
Value Sequence	
Value Table Name	
Value Table Name Colum...	
Value Table Value Colum...	

The highlighted Association properties are cleared:

Property Name	Property Value
Association Role Name	cargo
Source Entity Name	FlightPlan
Target Entity Name	Cargo
Cardinalities	1->*
Navigability	Bidirectional
Mandatory	Yes
XML Property Name	cargo
Java Object Get Method	
Java Object Set Method	
Java Object Field Name	
Join Expression	Freight.dbo.FlightPlan.flightNumber=Freight.dbo.Cargo.RflightP...

Note: This material was added as the topic "Clearing database mappings in the vocabulary" in the Rule Modeling Guide.

EDC support for MySQL

This release introduces Enterprise Data Connection support for an additional database. See the topic *"MySQL"* in the *Integration and Deployment Guide* for more information.

REST implementation supports JSONObject.NULL

Corticon's REST implementation now supports `JSONObject.NULL`.

What's new

JSON processing now supports `JSONObject.NULL`. The following excerpt is an example of its usage in the `OrderProcessingPayload.json` sample request:

```
{ "Objects": [ {
  "total": null,
  "myItems": [
    {
      "product": "Ball",
      "price": "10.000000",
      "quantity": "20",
      "subtotal": null,
```

What's changed

In supporting `JSONObject.NULL`, the syntax that was valid in prior releases, `"#null"`, is no longer valid. While the sample has been updated, any user-created requests that use `NULL` must change all instances of `"#null"` values to just `null`.

When passing a null inside a JSON object for an attribute of any type, either omit the JSON attribute inside the JSON object, or include the attribute name in the JSON Object with the value `JSONObject.NULL`.

The `CcServer` property `com.corticon.server.execution.json.null=[JSON_NULL | JAVA_NULL]` has been dropped, and, if specified, is ignored.

See the topic *"Passing null values in messages"* in the *Integration and Deployment Guide* for more information.

New EDC Tutorials and revised EDC documentation

This release revises the learning material for the Enterprise Data Connector (EDC) and restructures EDC documentation.

What's new

This release introduces two new Corticon tutorials that use Microsoft SQL Server 2014 as its EDC database:

- [Modeling Progress Corticon Rules to Access a Database using EDC](#) is aimed at rule modelers. It covers how to model and test rules that read/write to a relational database
- [Connecting a Progress Corticon Decision Service to a Database using EDC](#) is aimed at integration developers. It covers how to set up the Vocabulary to connect and map to a database, configure EDC settings in the Web Console, and other deployment-related tasks.

What's changed

The guide *Corticon EDC: Using Enterprise Data Connector* is no longer in the documentation set. Substantial portions of its material has been abstracted from any specific database and then melded into the restructured section "*Implementing EDC*" in the *Integration and Deployment Guide*.

Joining the Web Console Customer Experience Improvement Program

Corticon has implemented Progress Telerik Analytics to gather data that will help Progress Software determine product usage trends and improve product quality.

When you first launch Web Console, the **Customer Experience Improvement Program** dialog box opens. Read through the information in the dialog. The option to sign up for the Customer Experience Improvement Program is pre-selected.

See the topic "Participating in the Web Console Customer Experience Improvement Program" in the [Corticon Server: Web Console Guide](#).

What was new and changed in Corticon 5.5.1

This section summarizes the new, enhanced, and changed features in Progress® Corticon® 5.5.1, and provides links to those sections of the documentation.

For details, see the following topics:

- [Improved wizard to Package and Deploy Decision Services](#)
- [Server management utilities for building and testing Decision Services](#)
- [Introducing the ability to set deployment properties in a CDD file](#)
- [Using database caching for Enterprise Data Connections](#)
- [Finding a series of items in a collection sequence](#)
- [Enhancements to the Corticon Web Console](#)
- [Server logs record Decision Service metadata](#)
- [User Role preference replaced with Simple/Advanced Vocabulary view](#)
- [EDC support for Postgres, and Amazon Redshift](#)
- [Automatic refactoring of file names and file paths within Studio](#)
- [Joining the Studio Customer Experience Improvement Program](#)

Improved wizard to Package and Deploy Decision Services

What's new

The **Package and Deploy Decision Services** wizard performs compilations on the Studio machine, so you can choose to deploy to a server or save the Decision Service files on the Studio machine for later transfer to servers for deployment.

See *"Using Studio to compile and deploy Decision Services" in the Integration and Deployment Guide* for more information on using this feature.

What's changed

This feature, formerly known as the **Publish Wizard**, pushed required rule assets to the server where they were compiled and deployed.

Server management utilities for building and testing Decision Services

What's new

Users wanting to automate the building and testing of Decision Services will appreciate the addition of command line tools to compile Ruleflows into Decision Services ready for deployment, to create XSD & WSDL files for clients who will call the Decision Services, and to run Ruletests to validate that the Decision Services perform as expected. This can be used to script these processes and integrate them with other automated processes such as the "build" procedure for a larger project. To make this integration easier, a set of ANT macros are provided which make it easy to perform the build and testing of Decision Services within a custom ANT build script.

To run the command line utilities, just launch the **Corticon Command Prompt**, type `corticonManagement`, and then review the usage. The options are:

- `compile` - Compiles a Ruleflow into a Decision Service (.eds) file that can then be deployed to a Corticon server through the Web Console, .cdd file, or other supported tools.
- `test` - Executes one or more test sheets in a Corticon Ruletest (.ert) file, and produces an output file with the test results.
- `schema` - Generates XSD and WSDL schema files from either a Ruleflow (.erf) or Vocabulary (.ecore) file.
- `multicompile` - Compiles one or more Decision Services using directives defined in the specified XML file.

Note: When the target for deployment is the Corticon .NET server, you can use the `corticonManagement` utilities and ANT scripts to build .eds files and run tests. Then, running the IKVM utilities against the .eds file will generate the .NET bytecode.

See *"Using command line utilities to compile Decision Services" in the Integration and Deployment Guide* for more information. That section discusses all the `corticonManagement` utility options except `extractDiagnostics`. See *"Diagnosing runtime performance of server and Decision Services" in the Integration and Deployment Guide* for complete information on processing **diagnostic** data.

What's changed

This feature extends existing functionality. It does not require changes to any existing scripts or processes.

The `multiCompile.bat` utility is being replaced with the `multicompile` option of `corticonManagement.bat` and will be dropped in an upcoming release.

Introducing the ability to set deployment properties in a CDD file

What's new

When deploying with Corticon Deployment Descriptor (CDD) files, you might want to set deployment properties, such as controlling rule messages, in the CDD file so that the CDD fully describes the deployment configuration. In previous releases, you were constrained to properties that were set by the Deployment Console. Even if you edited the files or tried using the APIs, you could not get some properties specified in the CDD.

In this release, the format of the CDD file has been revised so that you can add all the available properties, and even add new ones as they are introduced in upcoming releases.

The new format adds an options section to a CDD where you list the name-value pairs of the available options you want and their respective values.

For more information, see the topics *"Setting deployment properties in a CDD file in a text editor"* and *"Setting deployment properties in a CDD file through APIs" in the Integration and Deployment Guide*.

What's changed

- The deployment console now writes `.cdd` files in the new format. To upgrade existing `.cdd` files that were created in a previous release, open an existing `.cdd` file in the Deployment Console, then choose **Save Deployment File**.
- Deployed `.cdd` files in the old format continue to be supported by Corticon Server.
- Deploying `.cdd` files in the old format will be supported by Corticon Server.

Using database caching for Enterprise Data Connections

What's new

As you move Decision Services that use a database connection toward production, you might want to get the performance benefits of caching entities and queries used in Decision Services. Even though the first cache use takes some performance overhead, subsequent references that use that cached content will see excellent performance. There are two ways caching is used:

An *Entity cache* is appropriate when common database-enabled entities are used in the input messages sent to a Decision Service.

- Case for using entity caching: Consider a Decision Service that expedites shipping requests. The Decision Service might receive a Order entity that has a one-to-many association with Supplier entities. When the Decision Service receives an Order entity it would query the database to get the associated Supplier entities. When using an entity cache, the Suppliers could be queried once and used in expediting other Order entities.

A *Query cache* optimizes lookup (query) of database data, such as when a cached entity is extended to the database in a Rulesheet. A Query Cache is read-only -- a lock is put on the records (returned from the query) associated with that decision service. If an update is attempted on a entity contained in a query cache an exception occurs. A Query Cache is **optimistic** -- that means that updates from outside of the Decision Service will not modify or invalidate the cache contents.

- Case for using query caching: Consider a Decision Service that prices online orders. With a query cache, it could query state sales tax rates once and use these when calculating the price of all orders it receives.
- Case for using query caching: Consider a Decision Service that prices insurance policies. With a query cache, it could query an actuarial table once and use the results in pricing multiple insurance policies.

First, you specify the caching settings in the Vocabulary and Rulesheets, and then enable caching in tests and deployed Decision Services.

Specifying Entity caching

To use entity caching:

1. Identify the Vocabulary entities that you want cached.
2. Edit the Vocabulary and then choose its **Advanced** view.
3. Confirm (or set) the entity's **Datastore Persistent** property to *Yes*
4. Choose the preferred **Datastore Caching** value:
 - *No Cache* or blank (default) - Disable caching.
 - *Read Only* - Caches data that is never updated.
 - *Read/Write* - Caches data that is sometimes updated while maintaining the semantics of "read committed" isolation level. If the database is set to "repeatable read," this concurrency strategy almost maintains the semantics. Repeatable read isolation is compromised in the case of concurrent writes.

- `Nonstrict Read/Write` - Caches data that is sometimes updated without ever locking the cache. If concurrent access to an item is possible, this concurrency strategy makes no guarantee that the item returned from the cache is the latest version available in the database.

Property Name	Property Value
Entity Name	Cargo
Entity Identity	
Inherits From	
XML Namespace	
XML Element Name	Cargo
Java Package	
Java Class Name	
Datastore Persistent	Yes
Table Name	Cargo.dbo.Cargo
Datastore Caching	Nonstrict Read/Write
Identity Strategy	
Identity Column Name	
Identity Sequence	
Identity Table Name	
Identity Table Name Column Name	
Identity Table Value Column Name	
Version Strategy	
Version Column Name	

Specifying Query caching

There are two ways to use query caching: on an entity in a Rulesheet, and on a database filter in a Rulesheet. You can use either or both as needed once you have enabled the Vocabulary.

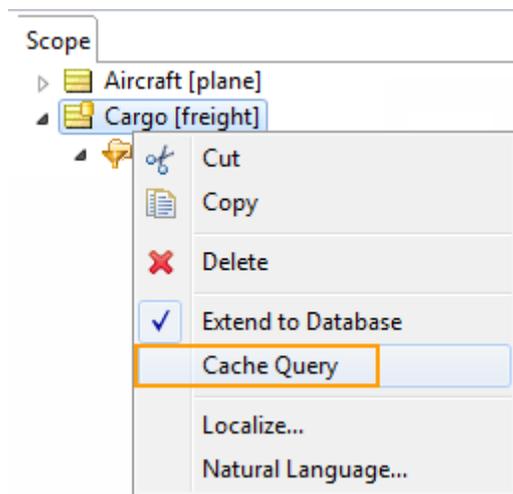
To enable query caching:

1. Identify the Vocabulary entities that you want cached.
2. Edit the Vocabulary and then choose its **Advanced** view.
3. Confirm (or set) the entity's **Datastore Persistent** property to `Yes`

Note: Query caching is independent of entity caching so the Datastore Caching setting is not relevant.

To use query caching on an entity in a Rulesheet:

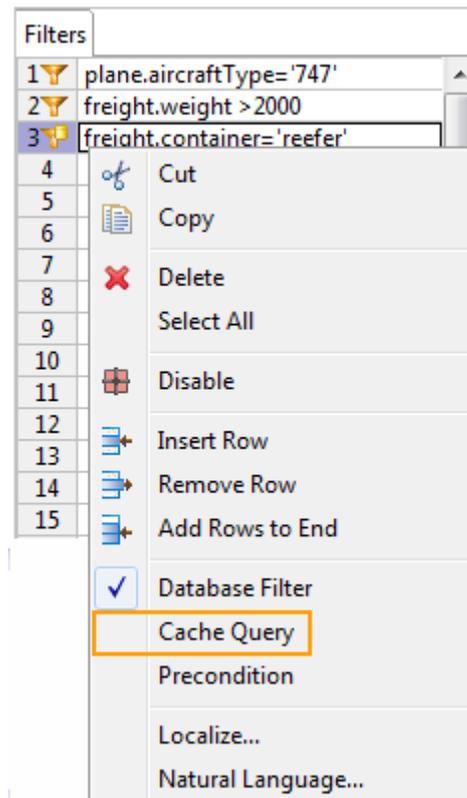
1. In a Rulesheet, choose **Advanced** view to show the **Scope** tab.
2. On the **Scope** tab, datastore-persistent entities show a database decoration, and each one offers the **Extend to Database** option in its dropdown menu. Choose that option to select it.
3. Once extended to database, the option to Cache Query is offered on the dropdown menu, as shown:



4. Choose the option to **Cache Query** on this entity in this Rulesheet.

To use query caching on a database filter in a Rulesheet:

1. In a Rulesheet, choose **Advanced** view to show the **Scope** tab.
2. On the **Scope** section, datastore-persistent entities show a database decoration, and each one offers the **Extend to Database** option in its dropdown menu. Choose that option to select it.
3. On the **Filters** tab, choose a filter that references an entity extended to database, the right-click to display its dropdown menu.
4. Choose **Database Filter**. The filter is decorated with a database symbol.
5. Dropdown the menu on that filter again. The option to **Cache Query** is now available, as shown:



6. Choose the option to **Cache Query** on this filter in this Rulesheet.

Enabling Caching on a Decision Service

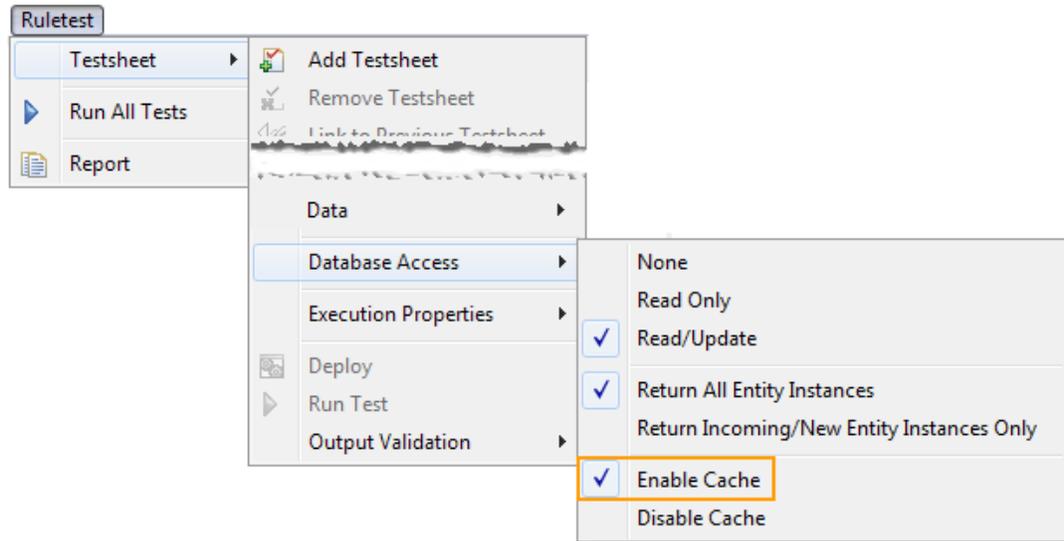
Once Rulesheets and the Vocabulary have enabled database caching, each Decision Service has to enable the caching. You can enable caching in Studio through options in Ruletests, and then set parameters that will enable caching on the Decision Services in deployment.

Enabling caching in Ruletests

Studio Ruletests are effectively a compiled Decision Service running on an embedded Corticon Server. As such, they can use the database connection and exercise caching behaviors that emulate deployment to remote servers.

To enable caching in Ruletests:

1. In Studio, use a Ruletest that specifies a test subject that is (or includes) a Rulesheet that specifies entity or query caching.
2. On the **Ruletest** menu, choose **Testsheet > Database Access > Enable Caching**, as shown:



Note: This material was added as the topic *"Defining database caching in Studio"* in the *Rule Modeling Guide* as well as in related menu settings in the *Quick Reference Guide*.

Enabling caching in Decision Service deployment

Each Decision Service can be enabled at deployment time to implement its cache settings. Corticon Servers, by default, do not enable database caching. Each Decision Service will have its own cache, and cached data is never shared between Decision Services. When the cache size reaches memory limits, it is written to disk on the server. Undeploying a Decision Service immediately clears its cache in memory and on disk.

To set the option to enable caching on each Decision Service during its deployment process, as available in these various deployment techniques:

- **Using Corticon Web Console** - Adding or editing a Decision Service in an application, as described in the topic *"Decision Service and Applications" in the Web Console Guide*.
- **Using Corticon Java Server Console** - Adding or editing a Decision Service, as described in the topic *"Deploy Decision Service page" in the Deploying Web Services with Java Guide*.
- **Using scripts testServerAxis and testServer** - Command 103 includes a parameter for enabling caching. Those commands use the underlying **Using the Corticon Server APIs**. For more information, see the topic *"Using Server API to compile and deploy Decision Services" in the Integration and Deployment Guide*.
- **Manually maintaining Deployment Descriptor (CDD) files** - Add the option "PROPERTY_DATABASE_ACCESS_CACHING_ENABLED" with the value "true", as discussed in topics in the section *"Using Deployment Descriptors to deploy Decision Services" in the Integration and Deployment Guide*.

Note: This material was added as the topic *"Working with database caches" in the Integration and Deployment Guide*. That topic was expanded in 5.5.2 to discuss caching behaviors and settings.

What's changed

The **Datastore Caching** setting on an entity, previously set to Yes|No with No as the permanent value, now enables and specifies the datastore caching technique.

Finding a series of items in a collection sequence

What's new

Corticon has always provided sequence operators that let you sort a collection by an attribute, and then access the value of another attribute of the one element in first, last, or a specified position in the sequence. Now you can find a *series* of elements -- such as the 'first three', the 'last three', or the 'second through fourth' -- elements in a collection sequence as a *subsequence*.

Three new operators have been added to Corticon to enable subsequence functionality:

- **subSequence(from,to):** <Sequence> ->subSequence(*integer1*, *integer2*)
Returns a Sequence containing all elements of <Sequence> between and including the positions *integer1* and *integer2*.
- **first(number):** <Sequence> ->first(*integer*)
Returns a subSequence of the first *integer* entities in the collection Sequence. In other words, ->first(*x*) is effectively ->subSequence(1, *x*)
- **last(number):** <Sequence> ->last(*integer*)
Returns a subSequence of the last *integer* entities in the collection Sequence. In other words, in a sequence of *n* elements, ->last(*x*) is effectively >subSequence(*n-x+1*, *n*)

The following table illustrates the results of subSequence operators against an INPUT sequence that was sorted into ascending order:

INPUT	first(3)	last(3)	subSequence(3,3)	subSequence(2,4)	subSequence(1,3)
A,B,C,D,E,F	A,B,C	D,E,F	C	B,C,D	A,B,C
A,B	A,B	A,B		B	A,B
A,B,C	A,B,C	A,B,C	C	B,C	A,B,C
A	A	A			A

For more information as well as examples of Rulesheets and Ruletests that use these operators, see:

- *"SubSequence" in the Rule Language Guide.*
- *"First(number)" in the Rule Language Guide.*
- *"Last(number)" in the Rule Language Guide.*

Note: Another operator, such as `->sortedBy` or `->sortedByDesc`, must be used to transform a Collection into a Sequence before subSequence operators can be used. Sequence must be expressed as a unique alias. See the topic *"Advanced collection sorting syntax" in the Rule Modeling Guide* for more examples of sorting operations.

For more general information on this topic, see the section *"Collections" in the Rule Modeling Guide*.

What's changed

This feature has no impact on existing operators or Rulesheets.

Enhancements to the Corticon Web Console

What's new

Corticon 5.5.1 features the following enhancements to the Web Console:

- **On Screen Tips** - Users who log in to the Web Console for the first time will see *on screen tips* that provide an overview of the Web Console interface.
- **Language preferences and localization** - You can view the Web Console in different languages by setting your preferences in the Web Console. Doing this also localizes the format of dates and numbers.
- **Summary view** - The Server and Application pages provide a preview pane to the right of the page. This preview pane displays a summary of page components and recent activities on the page. For example, on the Server page, the pane displays the total number of Servers available, the number of Server groups, etc. When you select a component on the page, the pane displays information about that component. For example, for a selected Server, the pane displays execution metrics, a list of Decision Services deployed on the Server, etc.
- **Drilldown to Decision Services in Server Summary view** - After you select a Server, you can double-click a Decision Service in the preview pane to navigate to the Decision Service's details page.

- **Ability to download log files** - You can download log files for local as well as remote Servers, choosing from two options--download all log files or download log files that have been modified by the Server in the last 24 hours.
- **Undeploy unmanaged Decision Services** - You can undeploy Decision Services that have been published to a Server from Corticon Studio, from the Server's details page in the Web Console.
- **Enable Database Caching** - You can enable database caching on each Decision service that uses the Enterprise Data Connector.
- **LDAP support** - You can configure the Web Console to authenticate users using an LDAP server.
- **Technique to reset administrator password to default** - If you lose your administrator password, you can reset the password to the default `admin` using a command line utility.
- **Inactive users are logged out automatically** - Administrators can configure a time period beyond which inactive users will get logged out automatically.
- **Audit trail** - The Web Console now has an Activity Log page where you can see logs for user actions and system events in the Web Console. You can also filter log messages by users, components, message types, etc.

See the online [Corticon Server: Web Console Guide](#) for more information about these features.

Server logs record Decision Service metadata

What's new

Corticon log files now add metadata on each Decision Service as it is loaded so that users can confirm consistent loading of a Decision Service instance in different environments. For example, the log header will have lines similar to the following:

```
2015-08-20 14:39:52.784 INFO DIAGNOSTIC [localhost-startStop-1]
Cc com.corticon.eclipse.server.core.impl.CcServerPool -
ADD DECISION SERVICE :: DecisionServiceName=Cargo,
Version=10,CompiledVersionNumber=5.5.1,
CompiledBuildNumber=7300,EDSTimestamp=08/03/15 4:04:03 PM,
RuleCount=2,MaxPoolSize=1,AutoReload=true,
CddPath=C:/_55x_install_dir/work_dir/Server/cdd/Cargo.cdd,
DatabaseAccessMode=null,ReturnEntities=ALL,ContainsServiceCallouts=false,
...
2015-08-20 14:39:52.815 INFO DIAGNOSTIC [localhost-startStop-1]
Cc com.corticon.eclipse.server.core.impl.CcServerPool -
ADD DECISION SERVICE :: DecisionServiceName=ProcessOrder,
Version=1.10,CompiledVersionNumber=5.5.1,
CompiledBuildNumber=7300,EDSTimestamp=08/03/15 4:04:07 PM,
RuleCount=6,MaxPoolSize=1,AutoReload=true,
CddPath=C:/_55x_install_dir/work_dir/Server/cdd/OrderProcessing.cdd,
DatabaseAccessMode=null,ReturnEntities=ALL,ContainsServiceCallouts=false,
```

This information was added to the topic *"Troubleshooting Corticon Server problems" in the Integration and Deployment Guide.*

What's changed

Logs have additional DIAGNOSTIC entries.

User Role preference replaced with Simple/Advanced Vocabulary view

Note: This *What's was New in 5.5.1* topic has been edited to reflect naming changes that occurred in 5.5.2.

The User Role preference setting was replaced with a Vocabulary toggle to make it easier to hide or show advanced vocabulary properties without the need to restart Studio.

What's new

The Vocabulary menu and toolbar provide a toggle between **Show Vocabulary Details** | **Hide Vocabulary Details**. When showing details, the Vocabulary exposes the database access and Java object messaging properties for entities, attributes, and associations. The menu actions on the Vocabulary menu and the **Database Access** tab are still presented and active. Switching between views does not clear or change any data.

See "*Vocabulary menu commands*" in the *Quick Reference Guide* for more information on using this feature.

What's changed

- The Studio's **Window > Preferences > Progress Corticon > Rule Modeling** panel for **User Role** is no longer available.
- The Vocabulary menu sections for **Database Access** and **Java Object Messaging** are not hidden when hiding details.
- The Rulesheet option to select an entity alias to **Extend to Database** is always available, provided that the entity was set to **Datastore Persistent**.

EDC support for Postgres, and Amazon Redshift

This release introduces Enterprise Data Connection support for additional databases.

See the topics "*Postgres*", and "*Amazon Redshift*" in the *Using Enterprise Data Connector Guide* for more information.

Automatic refactoring of file names and file paths within Studio

What's new

Within Studio, you can now rename files or folders and move files to other folders -- all with assurance that the Studio refactors the file names referenced in other files. For example, if you relocate the project's Vocabulary to a new `Vocab` folder, Rulesheets, Ruletests, and Ruleflows that reference the Vocabulary are automatically updated.

IMPORTANT NOTES:

- Close all files that are active in editors before attempting to change or move files in the Project Explorer. You cannot move files that are open. If you try, Studio prompts you to close those files.
- Refactoring support applies **only** when changes are made *within* Corticon Studio's Project Explorer. Other techniques for file name and path changes, such as Windows Explorer or command line interface, are NOT handled by this refactoring functionality and would result in dependent file references becoming invalid.
- This functionality relates to Corticon asset names and paths. Renaming a Vocabulary's elements -- domains, entities, attributes, and associations -- is advisable only before you create assets based on the Vocabulary.

For more information, see the topic "*Renaming and relocating assets in the Project Explorer*" in the *Quick Reference Guide*.

Joining the Studio Customer Experience Improvement Program

Corticon has implemented Progress Telerik Analytics to gather data that will help Progress Software determine product usage trends and improve product quality.

When you first launch Corticon Studio, the **Customer Experience Improvement Program** dialog box opens. Read through the information in the dialog. The option to sign up for the Customer Experience Improvement Program is pre-selected. Clicking **OK** accepts your participation in the program. To opt out, do one of the following:

- Clear the option, and then click **OK**,
- Click **Cancel**, or
- Click the close box

See "*Participating in the Customer Experience Improvement Program*" in the *Installation Guide* for more information.

What was new and changed in Corticon 5.5

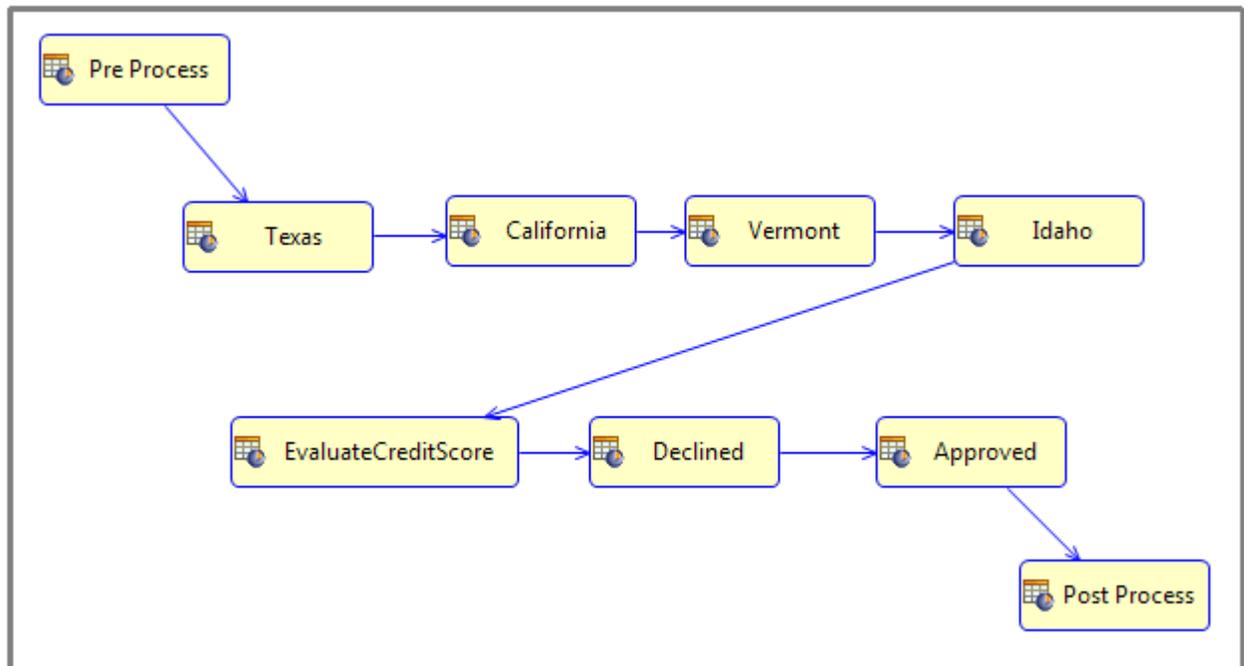
This section summarizes the new, enhanced, and changed features in Progress® Corticon® 5.5, and provides links to those sections of the documentation.

For details, see the following topics:

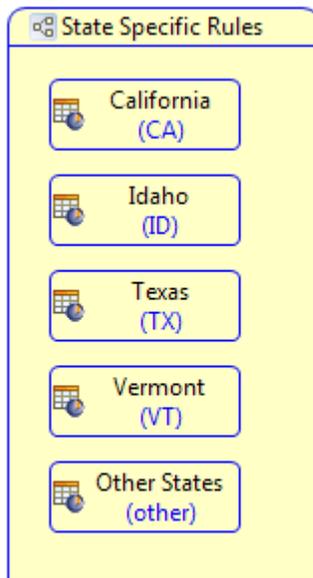
- [Introducing conditional branching in Ruleflows](#)
- [Introducing the ability to use a Ruleflow in another Ruleflow](#)
- [Introducing colors and comments on Ruleflow objects](#)
- [Improved Server Logging](#)
- [Execution thread pooling queue](#)
- [Consolidated Server installer](#)
- [Introducing the Corticon Web Console](#)
- [Introducing the Corticon REST Management API](#)
- [Improved JSON import and export](#)
- [Enhanced REST test server options](#)
- [New operator for set membership test](#)
- [Option to return just rule messages in Ruletests](#)
- [Other additions and changes in 5.5.0](#)

Introducing conditional branching in Ruleflows

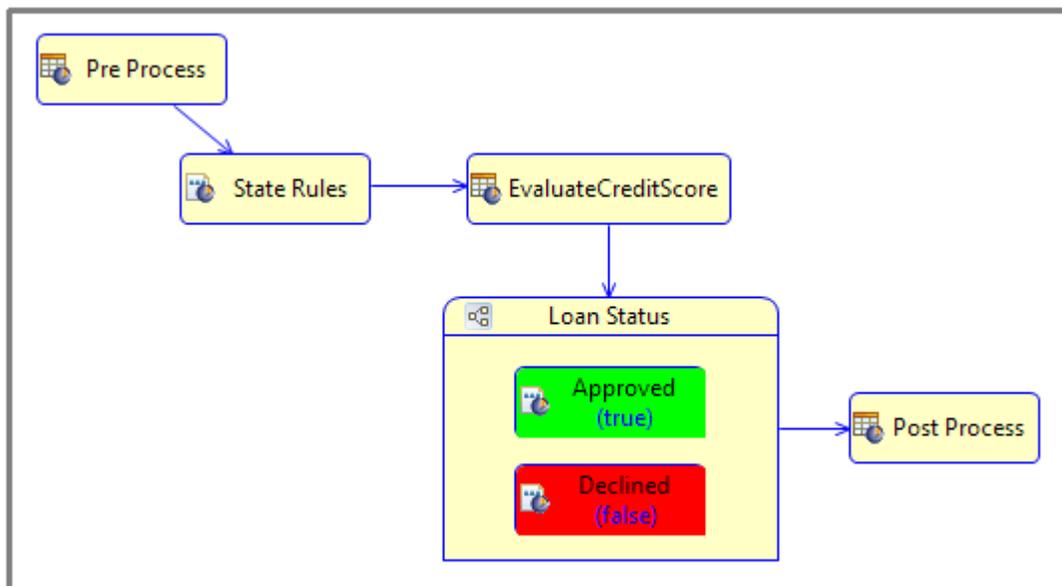
In a Ruleflow, you often have steps which should only process an entity with a specific attribute value. You can accomplish this by using *preconditions* on a Rulesheet, but the resulting logic, or flow, is difficult to perceive when looking at the Ruleflow. The following Ruleflow shows a progression of processing from the upper left to the lower right. But the rules to decide whether a loan is approved or declined are one-or-the-other, and the Rulesheets for the US states do not really represent a progression because the applicant's state is going to trigger only one of these Rulesheets to fire its rules:



Looking at this Ruleflow the real flow is somewhat hidden. If the Rulesheets for Texas, California, Vermont, and Idaho each had a precondition such that only matching states were processed, then they represent a set of mutually exclusive options, not the linear flow depicted in the Ruleflow. We'll see how we can create a branch in a Ruleflow like this:



And then bring that Ruleflow into another Ruleflow where we will also create a branch for the Declined and Approved Rulesheets that also might have needed to use preconditions. The completed Ruleflow looks like this:



A branch node can be Rulesheet, Ruleflow, Service Call Out, Subflow, or another Branch container.

Note: Multiple branches can be assigned to the same target activity. These values are shown as a set in the Ruleflow canvas.

Branching can occur on either enumerated or boolean attribute types. Only these are allowed because they have a set of known possible values. These possible values can be used to identify a branch. Using branches in a Ruleflow lets you clearly identify the set of options, or branches, for processing an entity based on an attribute value. In our example, using branching for the set of state options and whether the loan is approved or declined makes the flow more apparent. It will also be easier to create and maintain.

This material is the first section of the new topic *"Conditional Branching in Ruleflows" in the Rule Modeling Guide*. That topic also provides a refresher on enumerations and Booleans.

See also:

- *"Example of Branching based on a Boolean" in the Rule Modeling Guide.*
- *"Example of Branching based on an Enumeration" in the Rule Modeling Guide.*
- *the "Ruleflows" section of the Quick Reference Guide*
- *"Enumerations" in the Rule Modeling Guide.*

What's changed

These techniques and features do not change any existing behaviors or features of Ruleflows. Existing Ruleflows created in previous releases will update without issues, and can then add branches based on existing attributes that are Boolean or Enumerated data types.

Introducing the ability to use a Ruleflow in another Ruleflow

Until now, a Ruleflow had to contain each individual Rulesheet and Service Callout that made up the entire rule set for the Decision Service. The resulting Ruleflows often became unwieldy canvases where the only mechanisms for simplification were layout and enclosing multiple nodes into Subflows that had minimal success in reducing complexity.

Now, a Ruleflow can be brought onto the canvas of another Ruleflow, simply by dragging the `.erf` file onto the canvas. If you have a Ruleflow that calculates shipping costs and methods, you can use it in multiple other Ruleflows under the same Vocabulary. This ability to construct a Ruleflow from multiple other Ruleflows facilitates modularity, easier unit testing, collaboration, and management of complexity.

This nesting of a Ruleflow in another Ruleflow can be done multiple times in a Ruleflow (A includes B and C) or to arbitrary depth (A includes B which includes C). The imported Ruleflows are compiled into the parent Ruleflow when the Decision Service is generated. If A includes B and `A.eds` is generated, subsequent changes to B are not reflected until `A.eds` is regenerated. Even though B is included in A you can still generate a `B.eds` as needed.

This technique allows any Ruleflow to be included in another Ruleflow that uses the same Vocabulary -- it is not a *type* of Ruleflow.

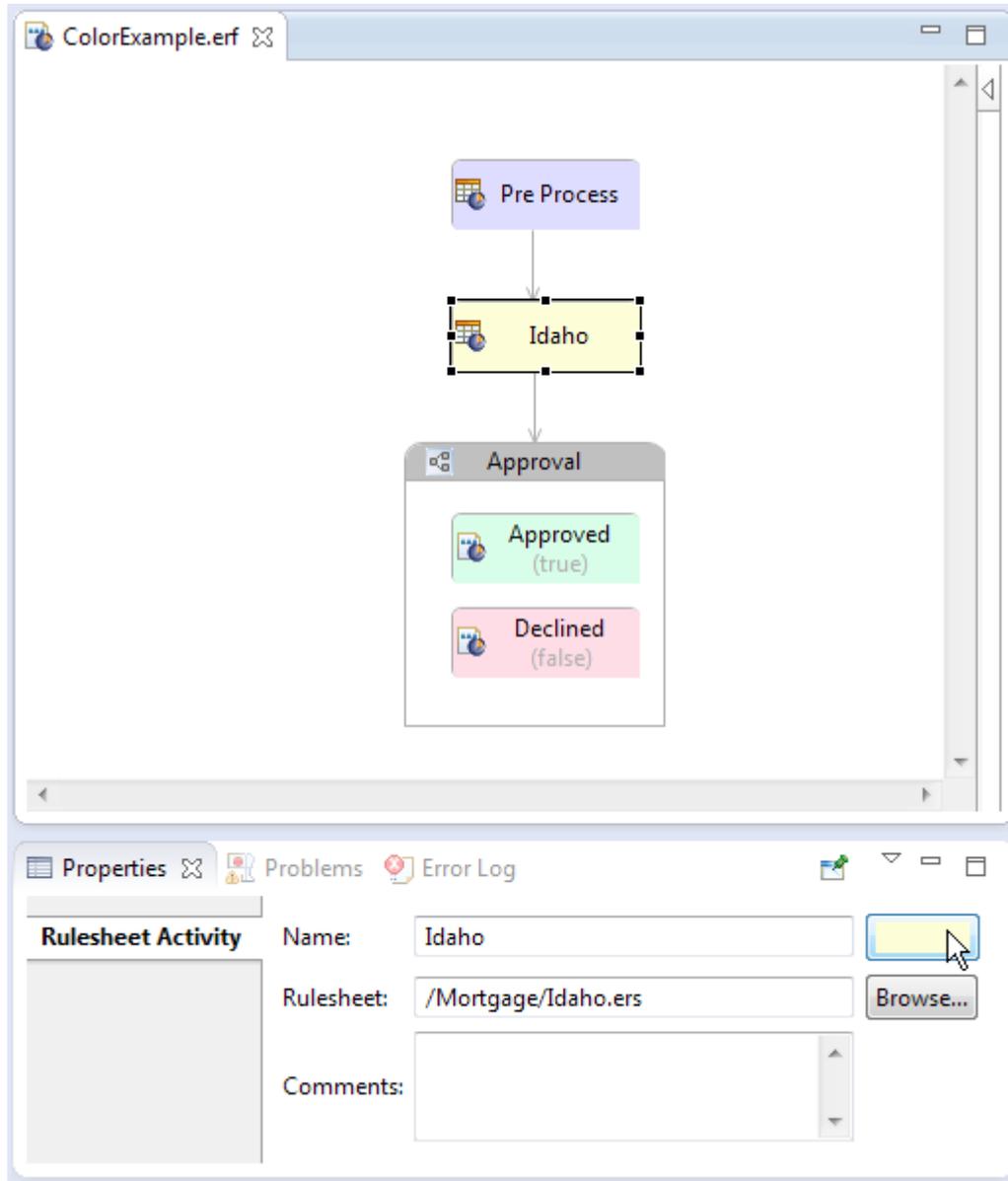
For more information, see *Using a Ruleflow in another Ruleflow in the Corticon Rule Modeling Guide* and the *"Ruleflows" section of the Quick Reference Guide*

What's changed

This technique does not change any existing behaviors or features of Ruleflows. Existing Ruleflows created in previous releases will update without issues, and then can be included in other Ruleflows as well bring other Ruleflows onto their canvas.

Introducing colors and comments on Ruleflow objects

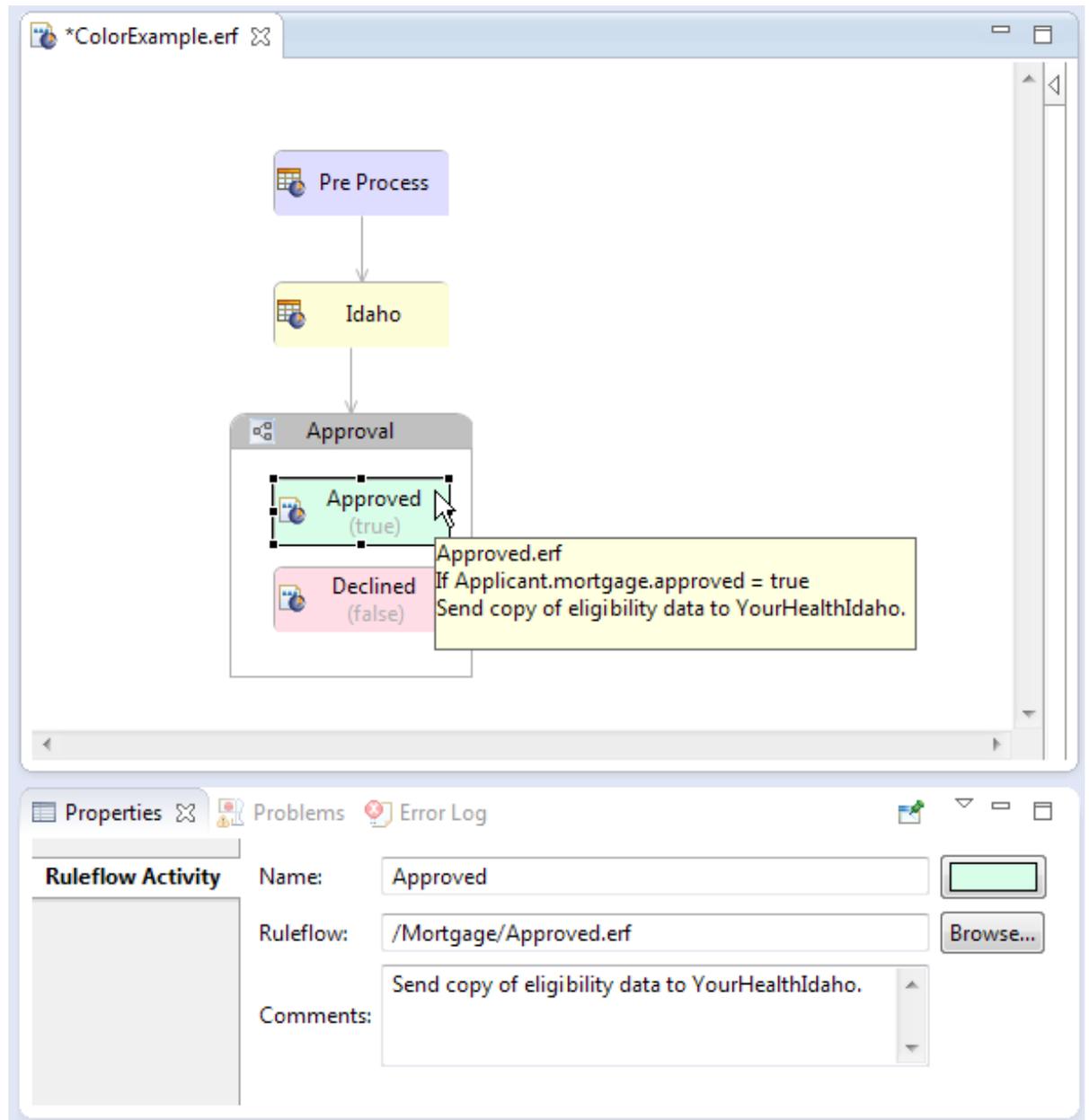
You can now add color to objects on the Ruleflow canvas to enhance the look and feel of the set of objects. Each object on the canvas provides a color button to set that objects color, as illustrated:



The color palette is the standard Windows basic 48-color palette that lets you choose to create custom colors.

Adding comments to Ruleflow objects

Each object on the Ruleflow palette can also have text that you add to its Property tab Comments. The following illustration shows that the text is displayed when you hover over the object.



In this example, the selected item is a Ruleflow that is the node for a branch activity. The info box first lists the asset name, then the branch attribute condition that assigned it to the node, and then the comment text.

This material was added as the topic *"Adding colors and comments to Ruleflow objects"* in the *Corticon Quick Reference Guide*.

Improved Server Logging

What's new

Corticon now implements logging features that provide faster, more flexible logging functionality for production Servers and Studio tests. Features include:

- **Distinct logging of Decision Service actions** - Filters let you mix your preferences for logging rule tracing, diagnostics, timing, invocations, and violations.
- **Simplified management of log files** - Configurations can specify time and size thresholds for log rotation, retention time of automatically compressed logs, and logging of individual Decision Services.
- **Focused logging configuration** - The `brms.properties` file defines overrides for all the exposed logging properties.

See a graphic presentation of the log settings and a look at actual logging runs in the technical video [Corticon Logging](#).

Log settings

The default log settings produce substantially the same output as previously (although much faster), and adds automatic rollover and retention. You can override all the default log settings by editing the `brms.properties` file.

- `loglevel` - The depth of detail in standard system logging. Values range from `OFF` through `ERROR`, `WARN`, to `INFO`, the default level. At this level, the survivors of processing-related log filters are also logged and will be included in higher log levels: `DEBUG`, `TRACE`, and `ALL`, the maximum system detail.
- `logFiltersAccept` - Log filters determine the types of processing log entries that will be recorded. By listing the `logFiltersAccept`, you can pick and choose just the entry types you want logged. The filter values that can be in the comma-separated list are:
 - `RULETRACE` - Records performance statistics on rules
 - `DIAGNOSTIC` - Records service performance diagnostics at a defined interval (default is 30 seconds).
 - `TIMING` - Records timing events.
 - `INVOCATION` - Records invocation events.
 - `VIOLATION` - Records exceptions.
 - `INTERNAL` - Records internal debug events
 - `SYSTEM` - Records low-level errors and fatal events.

The default `logFiltersAccept` setting is: `DIAGNOSTIC,SYSTEM`

- `logpath` - The directory where logs are written. Default value is `%CORTICON_WORK_DIR%/logs`. Note: Use forward slashes as path separator, as in `c:/Users/me/logs`
- `logDailyRollover` - Specifies whether to rollover the logs on a daily basis. Default value is `true`.

- `logRolloverMaxHistory` - Specifies the number of rollover logs to keep. Default value is 5.

The `loglevel` and `logpath` can be changed using following methods, which will override this setting:

```
ICcServer.setLogLevel(String)
ICcServer.setLogPath(String)
```

For more information, see *Using Corticon Server logs in the Integration and Deployment Guide*

Topics in that section include:

- *How users typically use logs* - Links to logging topics that demonstrate use cases.
- *Changing logging configuration* - Discusses the settings and options defined in the logging configuration properties (listed here.)
 - *Configuring log content* - Describes how log levels and info filters control the volume and type of log content.
 - *Configuring log files* - Shows how to configure log locations, archiving, and logging for each Decision Service.
- *Troubleshooting Corticon Server problems* - Describes several common problems that are revealed in logs.

What's changed

The features and details that are logged are substantially unchanged. However, the performance of logging and the way that logging is configured, filtered, stored, and archived have been completely revised. The performance of logging has improved dramatically over previous releases, so that production deployments can use detailed logging without significant impact.

Previously in Corticon logging, the incremental detail level of system log entries were intertwined with processing log levels. If you wanted a high level of system log data, such as `DEBUG`, you would get all the preceding processing log settings -- `INTERNAL`, `RULETRACE`, `DIAGNOSTIC`, `TIMING`, `INVOCATION`, `VIOLATION`, `SYSTEM`. Now you can set the log level to `DEBUG`, but set `logFiltersAccept=` (no value) in `brms.properties` to filter out processing log entries.

Logging per thread is no longer supported.

Some logging properties from prior versions are now ignored:

- `CcCommon.properties`:

```
logverbosity
com.corticon.logging.thirdparty.logger.class
```

- `CcServer.properties`:

```
com.corticon.server.execution.logperthread
```

Execution thread pooling queue

In prior releases, Decision Service instances executed using Decision Service-level thread pools. When multiple Decision Services were executing on a Server, this could overload the server and cause poor performance.

In this release, Server-level thread pooling is implemented by default, and multiple Decision Services place their requests in a queue for processing. That allows the server to determine how many concurrent requests are to be processed at any one time. The Corticon Server uses built-in Java concurrent pooling mechanisms to control the number of concurrent executions.

Implementation of an Execution Queue

Each thread coming into the Server gets an available Reactor from the Decision Service, and then the thread is added to the Server's Execution Thread Pooling Queue, or, simply put, in the **Execution Queue**. The Execution Queue guarantees that threads do not overload the cores of the machine by allowing a specified number of threads in the Execution Queue to start executing, while holding back the other threads. Once an executing thread completes, the next thread in the Execution Queue starts executing.

The Server will discover the number of cores on a machine and, by default, limit the number of concurrent executions to that many cores, but a property can be set to specify the number of concurrent executions. Most use cases will not need to set this property. However, if you have multiple applications running on the same machine as Corticon Server, you might want to set this property lower to limit the system resources Corticon uses. While this tactic might slow down Corticon processing when there is a heavy load of incoming threads, it will help ensure Corticon does not monopolize the system. Conversely, if you have Decision Services which make calls to external services (such as EDC to a database) you may want to set this property higher so that a core is not idle while a thread is waiting for a response.

Ability to Allocate Execution Threads

The Corticon Server takes each thread (regardless of which Decision Service the Thread is executing against), and then adds the thread to the Execution Queue in a first-in-first-out strategy. While that generally satisfies most use cases, you might want more control over which Decision Services get priority over other Decision Services. For that, you first set the Server property `com.corticon.server.decisionservice.allocation.enabled` to `true`, and then set the maximum number of execution threads (`maxPoolSize`) for each specified Decision Service in the Execution Queue. Once you set the allocation on every Decision Service, the Server will try to maintain corresponding allocations of execution threads from the Decision Services inside the Execution Queue.

When you have allocation enabled, you can dynamically change a Decision Service's `maxPoolSize`, depending on how you deployed that Decision Service:

- If deployed using the API method `ICcServer.addDecisionService(..., int aiMaxPoolSize, ...)`, then the `maxPoolSize` can be updated using `ICcServer.modifyDecisionServicePoolSizes(int aiMinPoolSize, int aiMaxPoolSize)`.
- If deployed using a CDD file, then change the value of `max-size` in the CDD. When the `CcServerMaintenanceThread` detects the change, it will update the Decision Service.

Better memory management

In prior releases, you were advised to set the minimum and maximum pool sizes to equal to the number of cores on the machine, in an effort to limit the number of concurrent threads that would be executing. Now, the ability to do allocation means that you could allocate hundreds of execution threads for one Decision Service. But in the old paradigm that would take a lot of memory. By refactoring how Reactors are maintained in each Decision Service, the Server can now re-use cached data across all Reactors for a Decision Service. As a result, runtime performance should reveal only modest differences in memory utilization between a Decision Service that contains just one Reactor and another that contains hundreds of Reactors.

In prior releases, you set minimum and maximum pool sizes that the Server would use based on load. As load increased, the Server would allocate more Reactors to the Decision Service Pool (up to Max Pool Size), then, as load decreased, the Server would remove Reactors (down to Min Pool Size). This mechanism attempted to throttle the Server so that it would not run out of memory. In this release, there is no need to decrease the number of Reactors in the Pool because extra Reactors are not actually sitting in the Pool. A new Reactor is created for every execution thread, and -- when the execution is done -- the Reactor is not put back into the Pool for reuse (as it was in previous versions), it just drops out of scope and garbage collection releases its memory.

With better memory management, large request payloads are more of a concern than the number of concurrent executions or the number of Reactors.

Take a deep dive into this feature in the technical video [Server Level Thread Pooling](#).

New Server properties

- Determines how many concurrent executions can occur across the Server. Ideally, this value is set to the number of Cores on the machine. By default, this value is set to 0, which means the Server will auto-detect the number of Cores on the server.

```
com.corticon.server.execution.processors.available=0
```

- Specifies the timeout setting for how long an execution thread can be inside the Execution Queue. The time starts when the execution thread enters the Execution Queue and ends when it completes executing against a Decision Service. A `CcServerTimeoutException` will be thrown if the execution thread fails to complete in the allotted time. The value is in milliseconds. Default value is 180000 (180000ms = 3 minutes)

```
com.corticon.server.execution.queue.timeout=180000
```

- In some cases, you might want to cause prioritizing of one Decision Service over another, making more resources available to that type. Setting this property's value to `true` enables Decision Service level allocations to control the number of Decision Service instances that can be added to the queue at a particular time. Default value is `false`

```
com.corticon.server.decisionservice.allocation.enabled=false
```

New API methods: `addDecisionService` with `aiQueueAllocation` parameter

```
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle)
```

```
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,
```

```
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    boolean abDeployAsTestDecisionService)  
  
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    String astrDatabaseAccessMode,  
    String astrDatabaseAccessReturnEntities,  
    String astrDatabaseAccessPropertiesPath)  
  
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    String astrDatabaseAccessMode,  
    String astrDatabaseAccessReturnEntities,  
    String astrDatabaseAccessPropertiesPath,  
    boolean abDeployAsTestDecisionService)
```

New API methods: modifyDecisionServicePoolSize

```
void modifyDecisionServiceQueueAllocation(String astrDecisionServiceName,  
    int aiMaxPoolSize,)  
  
void modifyDecisionServiceQueueAllocation(String astrDecisionServiceName,  
    int aiDecisionServiceMajorVersion,  
    int aiMaxPoolSize,)  
  
void modifyDecisionServiceQueueAllocation(String astrDecisionServiceName,  
    int aiDecisionServiceMajorVersion,  
    int aiDecisionServiceMinorVersion,  
    int aiMaxPoolSize,)
```

For more information, see the topic *Multi-threading, concurrency reactors, and server pools in the "Inside Corticon Server" section of the Corticon Integration and Deployment Guide*.

What's changed

The revisions to Corticon Server thread pooling have caused several Server properties and API methods to be dropped.

Dropped Server properties

The following CcServer properties have been removed:

```
com.corticon.server.pool.expansion.percentage  
com.corticon.server.serverpool.timeout  
com.corticon.server.pool.expansion.percentage
```

Deprecated API methods: The addDecisionService methods with minPoolSize and MaxPoolSize parameters in their signature.

```
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMinPoolSize,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle)
```

```
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMinPoolSize,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    boolean abDeployAsTestDecisionService)  
  
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMinPoolSize,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    String astrDatabaseAccessMode,  
    String astrDatabaseAccessReturnEntities,  
    String astrDatabaseAccessPropertiesPath)  
  
void addDecisionService(String astrDecisionServiceName,  
    String astrRuleAssetPath,  
    boolean abAutoReload,  
    int aiMinPoolSize,  
    int aiMaxPoolSize,  
    String astrMsgStructStyle,  
    String astrDatabaseAccessMode,  
    String astrDatabaseAccessReturnEntities,  
    String astrDatabaseAccessPropertiesPath,  
    boolean abDeployAsTestDecisionService)
```

Dropped API methods: All `modifyDecisionServicePoolSizes` methods

```
void modifyDecisionServicePoolSizes(String astrDecisionServiceName,  
    int aiNewMinPoolSize,  
    int aiNewMaxPoolSize)  
  
void modifyDecisionServicePoolSizes(String astrDecisionServiceName,  
    int aiDecisionServiceMajorVersion,  
    int aiNewMinPoolSize,  
    int aiNewMaxPoolSize)  
  
void modifyDecisionServicePoolSizes(String astrDecisionServiceName,  
    int aiDecisionServiceMajorVersion,  
    int aiDecisionServiceMinorVersion,  
    int aiNewMinPoolSize,  
    int aiNewMaxPoolSize)
```

Consolidated Server installer

Corticon now provides one download package to install Corticon Server for Java, Corticon Server for .NET, and the Corticon Web Console - any one of these components or even all three. The installer colocates the selected components into the specified target home and work directory.

These server installations can gather installation information so that the installer can perform silent installations on other machines.

For more information, see *Installing Corticon Servers and the Web Console in the Corticon Installation Guide*

What's changed

In prior releases, the EAR/WAR containers for deployment on other platforms and application servers were bundled into the Java Server installer. Those containers are available as a separate download. This impacts users in two ways:

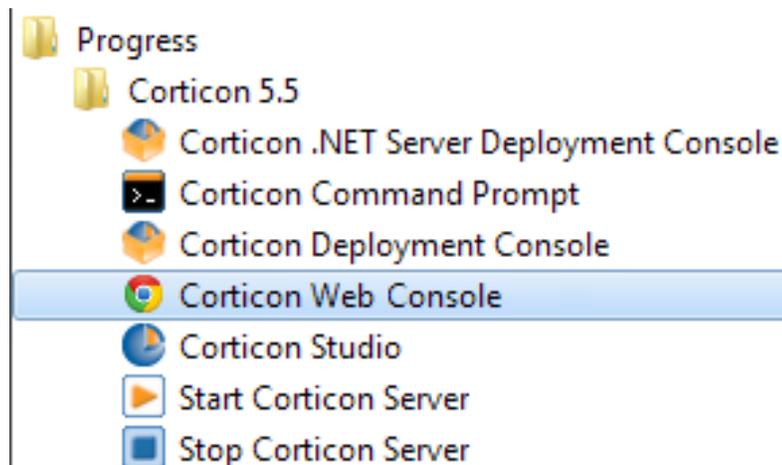
- If you want to deploy to JBoss or as an EAR file, download the server archive, and then extract the appropriate files.
- If you want to deploy `axis.war` to your own appserver, copy it from `[CORTICON_WORK]/pas/server/webapps`. (Note that this path was `[CORTICON_HOME]/pas/webapps` in 5.4 releases.)

Introducing the Corticon Web Console

What's new

The Web Console provides for a central point of managing and monitoring Corticon Java and .NET Servers. It supports deploying of Decision Services and provides a variety of metrics on both the performance of Decision Services and Servers. The Web Console can be installed in the same application server as the Corticon Java Server or installed, by itself, in a separate application server and used to manage and monitor multiple Corticon servers.

On the installation machine, a **Start** menu command (as shown, where all 5.5 components and products were also installed) launches the Web Console in your default browser:



For more information, see *Installing Corticon Servers and the Web Console in the Corticon Installation Guide* and the online [Corticon Server: Web Console Guide](#).

What's changed

The previous console implementation, Java Server Console, that was enabled in a Java Server installation is still available. This tool will be phased out in upcoming releases as the new Web Console replaces the Java Server Console's functionality.

Introducing the Corticon REST Management API

The introduction of the REST management API in the *Integration and Deployment Guide* provides information about this API's common request/response types, metrics, status codes, and error handling. See the section "*REST Management API*" in the *Integration and Deployment Guide*.

Method	Type	Syntax and function
API ListDecisionServices	HTTP GET	<code><base>/decisionService/list</code> Returns a list of Decision Services deployed on the server. The resulting payload will be enclosed in the response's body in JSON object form.
API Deploy Decision Service	HTTP POST	<code><base>/decisionService/deploy</code> Attempts to add a Decision Service to the server. Request objects will be sent as the content.
API Undeploy Decision Service	HTTP POST	<code><base>/decisionService/undeploy</code> Attempts to remove the Decision Service from the server and delete the EDS file associated with it. The request will take HTTP headers that provides the Decision Service name, and, optionally the Major and Minor version Number.
API Get Decision Service Properties	HTTP GET	<code><base>/decisionService/getProperties</code> Returns the properties pertaining to the Decision Service. The request gets the properties for the Decision Service passed in its header. The request will take HTTP headers that provide the Decision Service name, and, optionally the Major and Minor version Number.
API Set Decision Service Properties	HTTP POST	<code><base>/decisionService/setProperties</code> Attempts to modify the properties of a specified Decision Service.
API Ping Server	HTTP GET	<code><base>/server/ping</code> Returns an object containing the current uptime of the server, which confirms that the server is reachable and running.

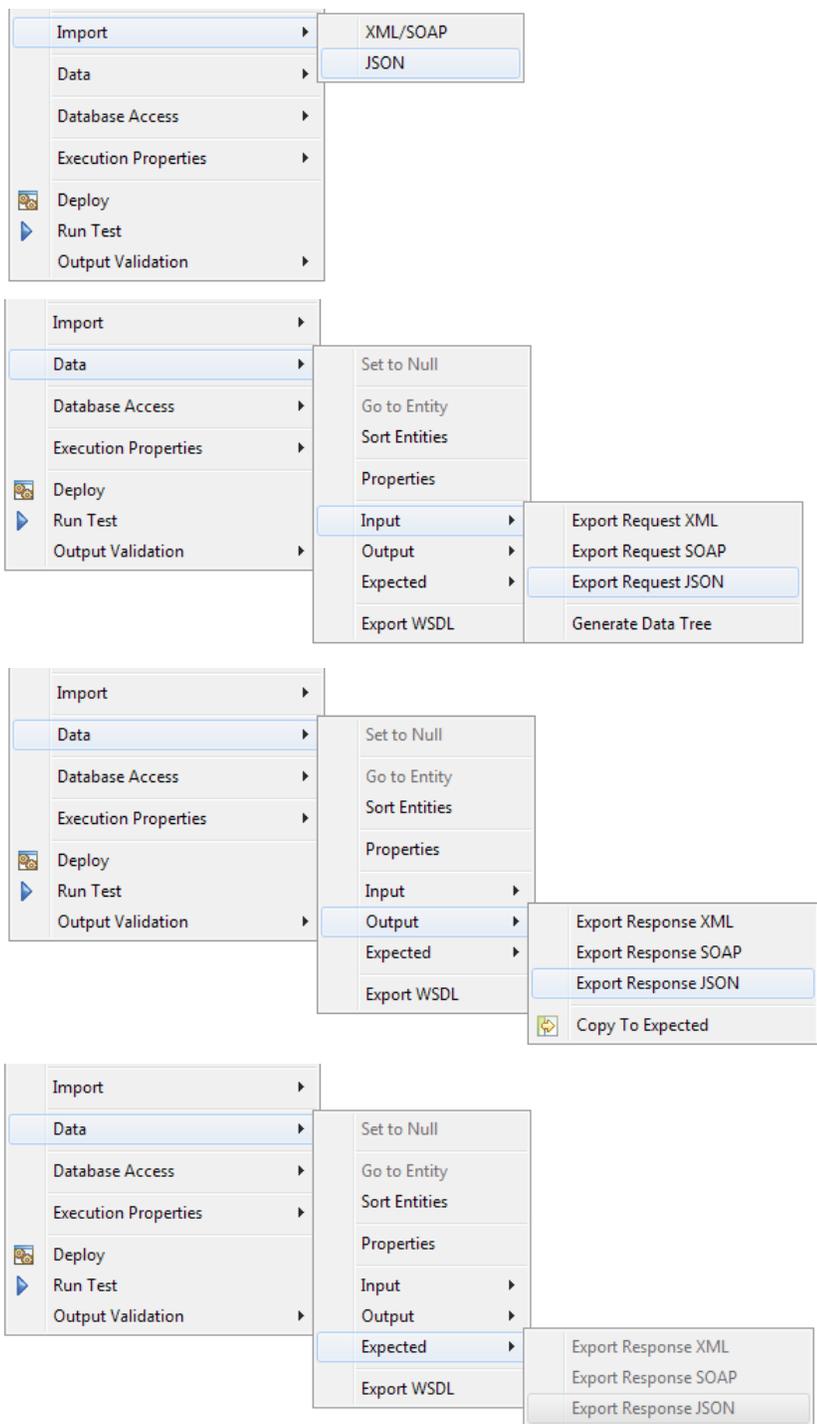
Method	Type	Syntax and function
API Retrieve Metrics	HTTP POST	<p><code><base>/server/metrics</code></p> <p>Retrieves metrics for the server. The request can pass in a Decision Service (or a list of Decision Services as an array), and a timestamp showing when the windowed metrics will begin. If the JSON object does not contain any Decision Service, metrics are returned for all the Decision Services in the server along with the server metrics.</p>
API Get Server Info	HTTP GET	<p><code><base>/server/info</code></p> <p>Returns information about the server. The returned object will contain information about both the server and the Decision Services deployed on the server.</p>
API Get Server Log	HTTP GET	<p><code><base>/server/log</code></p> <p>Returns the server log entries after the specified timestamp.</p>
API Get Server Properties	HTTP GET	<p><code><base>/server/getProperties</code></p> <p>Returns the properties pertaining to the server. The returned object will be a list of key/value pairs consisting of key: <i>propertyName</i> value: <i>value</i> with information about the property.</p>
API Set Server Properties	HTTP POST	<p><code><base>/server/setProperties</code></p> <p>Sets properties on the server using a JSON object. The object will contain a key/value format system: <i>propertyName</i> value: <i>value</i> intended to be set for this property.</p>
API Set Server License	HTTP POST	<p><code><base>/server/setLicense</code></p> <p>Sets the license file for the server. This can be used to add a license in the event the current one is not usable.</p>

For more information, see *REST management API in the Corticon Integration and Deployment Guide*.

Improved JSON import and export

Corticon Studio's Ruletest menus now enable easy import and export of JSON formatted requests and responses.

The Ruletest's Testsheet menu offers options specific to JSON while maintaining the familiar SOAP and XML options, as shown:



For more information, see *Ruletest menu commands*, *Importing a JSON document to a testsheet*, and *Exporting a testsheet to a JSON document* in the *Quick Reference Guide*.

Enhanced REST test server options

What's new

The `testServerREST` script has new test options:

```
142 - Execute JSON REST request
143 - Execute JSON REST request (by specific Decision Service Major Version)
144 - Execute JSON REST request (by specific Decision Service Major and Minor
Version)
145 - Execute JSON REST request (by specific execution Date)
146 - Execute JSON REST request (by specific execution Date and Decision Service
Major Version)
```

For more information, see the topic *Running the Sample JSON Request in the guide Corticon Server: Deploying Web Services on .NET* and *Running the Sample JSON Request in the guide Corticon Server: Deploying Web Services on Java*

What's changed

The `testServerREST` script on both servers previously had only one test:

```
132 - Execute JSON REST request
```

That test number is no longer available for RESTful JSON formatted requests.

New operator for set membership test

A new operator has been added to the Corticon Studio to simplify the qualification of set membership in a test. When the values, particularly ones in an enumerated list, are not structured for range value tests, the workaround has been to use a long list of logical OR operations, such as:

```
candidate.State='MA' or candidate.State='NH' or candidate.State='VT' or
candidate.State='RI' or candidate.State='ME' or candidate.State='CT'
```

Now, that can be expressed as:

```
candidate.State in {'MA','NH','VT','RI','ME','CT'}
```

When you are using enumerations, such as...

Data Type Na...	Base Data Type	Enumerati...	...	Label	Value
state	String	Yes		MA	'Massachusetts'
				NH	'New Hampshire'
				VT	'Vermont'
				RI	'Rhode Island'
				ME	'Maine'
				CT	'Connecticut'

... the String labels do not require quotes in the list. For example:

```
candidate.State in {MA,NH,VT,RI,ME,CT}
```

For more information, see the section *Qualifying rules with ranges and lists in the Corticon Studio: Modeling Guide* and *the operator topic "In {List}" in Corticon Studio: Rule Language Guide*

What's changed

The `in` operator is now listed in the **Rule Operators** view.

Option to return just rule messages in Ruletests

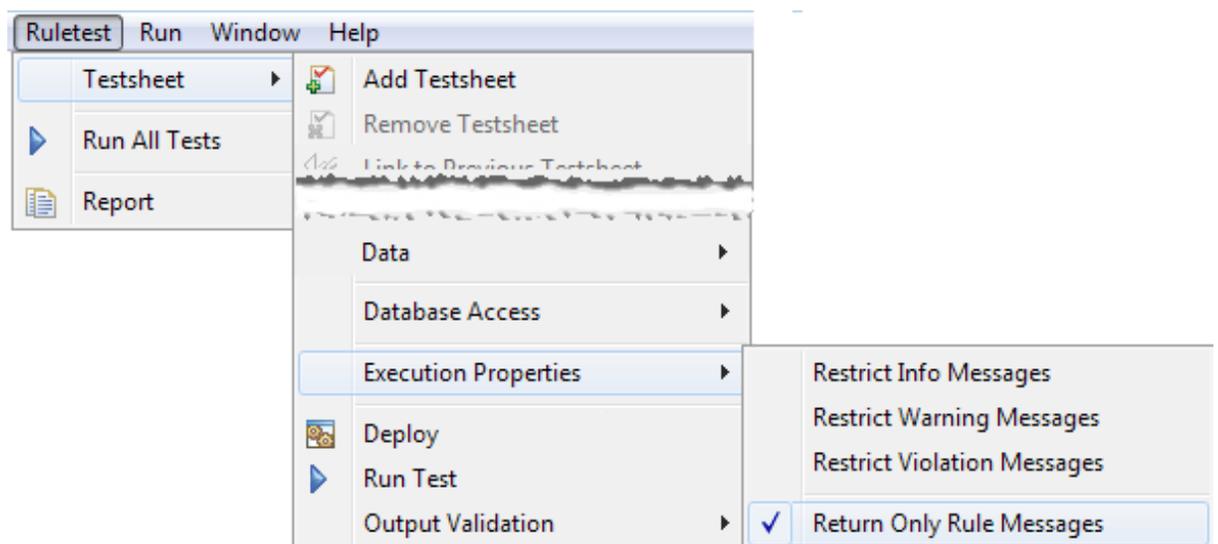
In some rule model designs, the rules create many new entities that generate database entries through the Enterprise Data Connection (EDC), and corresponding rule messages in the Corticon output. The generation of output in the Ruletest slows down the test process. You can choose to suppress the work document output in server output and logs, and you can test that same behavior in your Ruletests first. This feature lets you tell each Decision Service or each execution whether to return the Entities under the WorkDocuments section in the CorticonResponse, which results in much smaller -- and faster -- CorticonResponse.

The new execution property that suppresses the creation of the output pane, and displays just rule messages is:

```
com.corticon.server.restrict.response.rulemessages.only
```

Note: You can also suppress any of the three types of Rule Messages (info, warning, and violation) from being posted to the output of an execution.

You can see the impact of these restrictions in testsheets by engaging the option toggle under **Execution Properties** on the **Ruletest > Testsheet** menu. When the option is checked, output presentation is suppressed:



Clicking the option again clears it, and that output is presented in the next test run.

Importing and exporting SOAP/XML messages will carry these settings:

- On import, a payload that includes (for example)

```
<ExecutionProperties>
  <ExecutionProperty
    value="true"
    name="PROPERTY_EXECUTION_RESTRICT_RESPONSE_TO_RULEMESSAGES_ONLY"/>
</ExecutionProperties>
```

will select the testsheet option **Return Only Rule Messages**.

- On export, the checked **Execution Property** option generates an `ExecutionProperty` line set to `true`.

This information was added to *"Ruletest menu commands" in the Quick Reference Guide*.

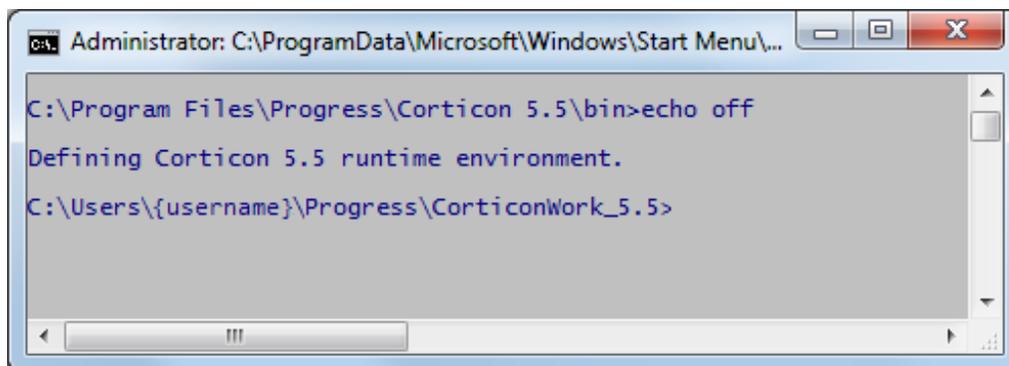
Other additions and changes in 5.5.0

What's new

- **Corticon Command Prompt** - The **Start** menu on a Server installation now offers a command to launch a Windows Command Prompt:



The command opens a Command Prompt window, calls `corticon_env.bat`, and then locates the prompt at the root of the Corticon work directory:



It adds several `[CORTICON_HOME]` script paths to the `PATH` so that you can launch scripts by name from the following Server installation locations:

- `\bin`
- `\Server\bin`
- `\Server\pas\bin`
- **New server property for names on complexTypes** - A server property lets you choose whether the XSD and WSDL generators append the word "Type" at the end of each `complexType` in the related XSD or WSDL file. This was the standard in earlier versions of the generators. Default value is `false`.

```
com.corticon.servicecontracts.append.typeLabel=false
```

What's changed

- **Savvion material dropped** - The topics "Building a Vocabulary based on Savvion Dataslots", "Generating the Vocabulary file", and "Dataslot to Vocabulary Mapping" have been dropped.
- **Basic and Advanced Tutorials are revised and available in the cloud** - The two introductory modeling tutorials have been reformatted into a new style and access point. The content of these tutorials is no longer in the online help or accessible in ESD downloads.

Progress Corticon documentation - Where and What

Corticon provides its tutorials documentation in various online and installed components.

Access to Corticon tutorials and documentation

Corticon Online Tutorials	
Tutorial: Basic Rule Modeling in Corticon Studio	An introduction to the Corticon Business Rules Modeling Studio. Learn how to capture rules from business specifications, model the rules, analyze them for logical errors, and test the execution of your rules -- all without any programming. <i>Online only. Uses samples packaged in the Corticon Studio.</i>
Tutorial: Advanced Rule Modeling in Corticon Studio	Learn about the concepts underlying some of Studio's more complex and powerful functions such as ruleflows, scope and defining aliases in rules, understanding collections, using String/DateTime/Collection operators, modeling formulas and equations in rules, and using filters. <i>Online only.</i>
Modeling Progress Corticon Rules to Access a Database using EDC	Shows rule modelers how to model and test rules that read/write to a relational database. <i>Online only. Uses samples packaged in the Corticon Studio, and Microsoft SQL Server 2014 as its EDC database.</i>

Connecting a Progress Corticon Decision Service to a Database using EDC	Shows integration developers how to set up the Vocabulary to connect and map to a database, configure EDC settings in the Web Console, and other deployment-related tasks. <i>Online only. Uses samples packaged in the Corticon Studio, and Microsoft SQL Server 2014 as its EDC database.</i>
Corticon Online Documentation	
Progress Corticon User Assistance	Updated online help for the current release.
Introducing the Progress® Application Server	The Progress Application Server (PAS) is the Web application server based on Apache Tomcat installed as the default Corticon Server. TCMAN, the command-line utility, manages and administers the Progress Application Server.
Progress Corticon Documentation site	Access to all guides in the Corticon documentation set in PDF format and JavaDocs.
Corticon Documentation on the Progress download site	
Documentation	Package of all guides in PDF format.
What's New Guide	PDF format.
Installation Guide	PDF format.
Corticon Studio Installers	Include Eclipse help for all guides except Web Console.

Components of the Corticon documentation set

The components of the Progress Corticon documentation set are the following guides:

Release and Installation Information	
<i>What's New in Corticon</i>	Describes the enhancements and changes to the product since its last point release.
<i>Corticon Installation Guide</i>	Step-by-step procedures for installing all Corticon products in this release.
Corticon Studio Documentation: Defining and Modeling Business Rules	

<i>Corticon Studio: Rule Modeling Guide</i>	Presents the concepts and purposes the Corticon Vocabulary, then shows how to work with it in Rulesheets by using scope, filters, conditions, collections, and calculations. Discusses chaining, looping, dependencies, filters and preconditions in rules. Presents the Enterprise Data Connector from a rules viewpoint, and then shows how database queries work. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting of Rulesheets and Ruleflows. Includes <i>Test Yourself</i> exercises and answers.
<i>Corticon Studio: Quick Reference Guide</i>	Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.
<i>Corticon Studio: Rule Language Guide</i>	Reference information for all operators available in the Corticon Studio Vocabulary. Rulesheet and Ruletest examples are provided for many of the operators.
<i>Corticon Studio: Extensions Guide</i>	Detailed technical information about the Corticon extension framework for extended operators and service call-outs. Describes several types of operator extensions, and how to create a custom extension plug-in.
Corticon Server Documentation: Deploying Rules as Decision Services	
<i>Corticon Server: Integration and Deployment Guide</i>	An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the Deployment Console tool. Describes JSON request syntax and REST calls. Discusses relational database concepts and implementation of the Enterprise Data Connector. Goes deep into the server to discuss state, persistence, and invocations by version or effective date. Includes troubleshooting servers through logs, server monitoring techniques, performance diagnostics, and recommendations for performance tuning.
<i>Corticon Server: Deploying Web Services with Java</i>	Details setting up an installed Corticon Server as a Web Services Server, and then deploying and exposing Decision Services as Web Services on the Progress Application Server (PAS) and other Java-based servers. Includes samples of XML and JSON requests.

<i>Corticon Server: Deploying Web Services with .NET</i>	Details setting up an installed Corticon Server as a Web Services Server, and then deploying and exposing decisions as Web Services with .NET. Includes samples of XML and JSON requests.
Corticon Server: Web Console Guide	Presents the features and functions of browser connection to a Web Console installation to manage Java and .NET servers in groups, manage Decision Services as applications, and monitor performance metrics of managed servers.