PROGRESS Corticon

# What's New in Corticon

PROGRESS

# Notices

# 1

# 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. 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* to access, prepare, and install Corticon Studio.

- **Corticon Servers** implement web services 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).

## 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.

# 2

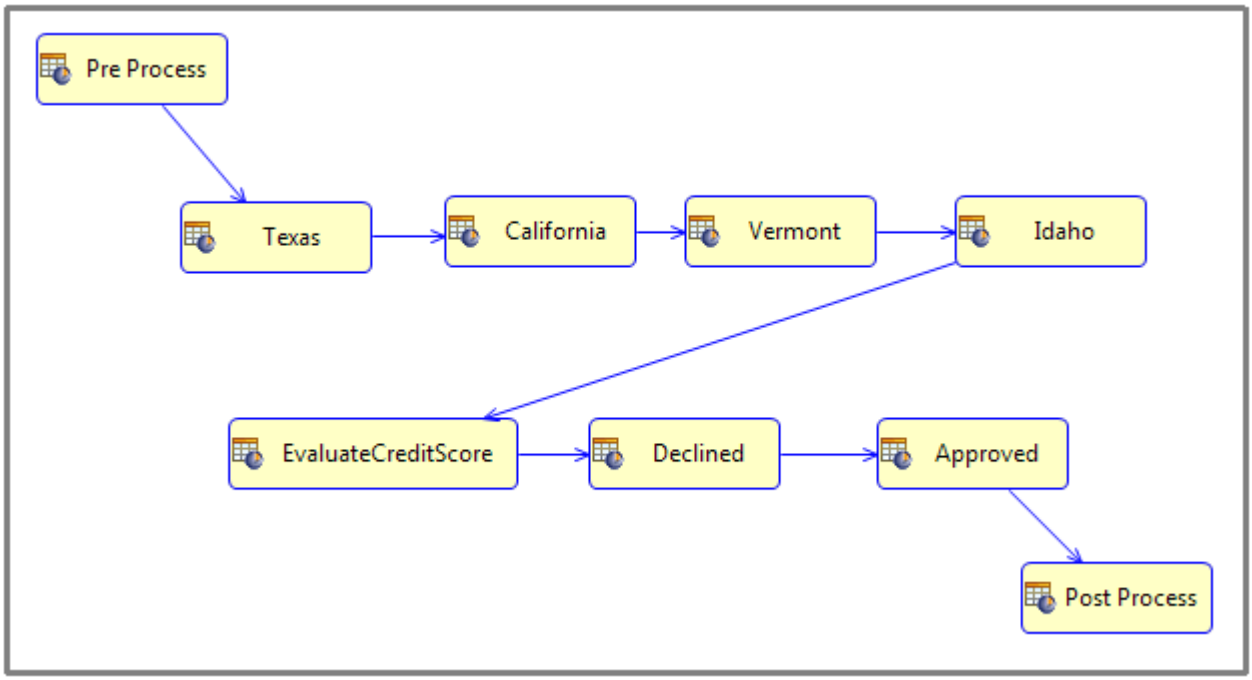
# What's new and changed in Corticon 5.5

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:

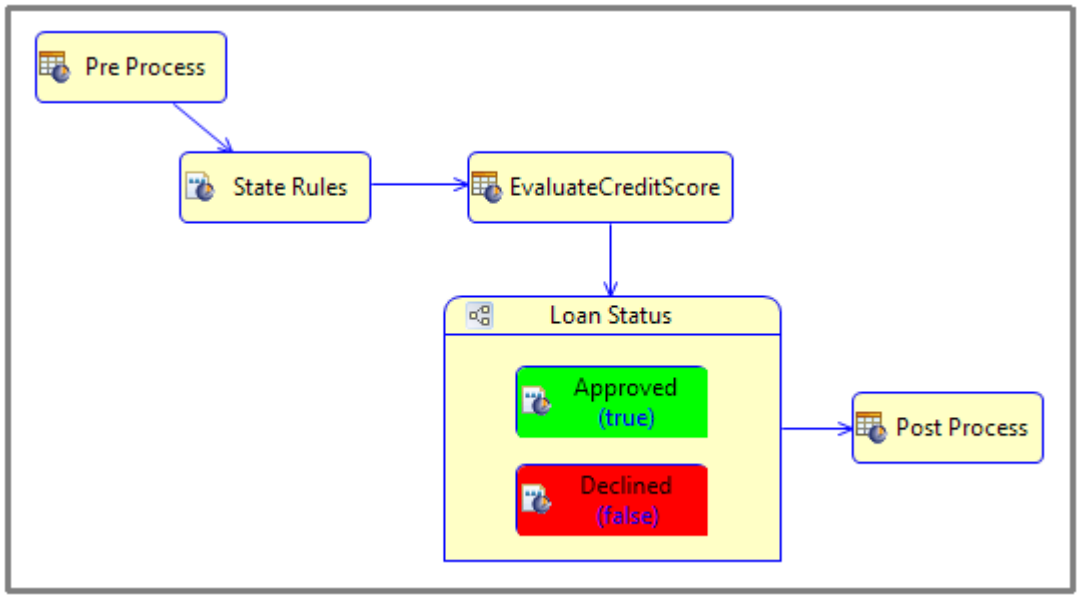
# Introducing conditional branching in Ruleflows

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.

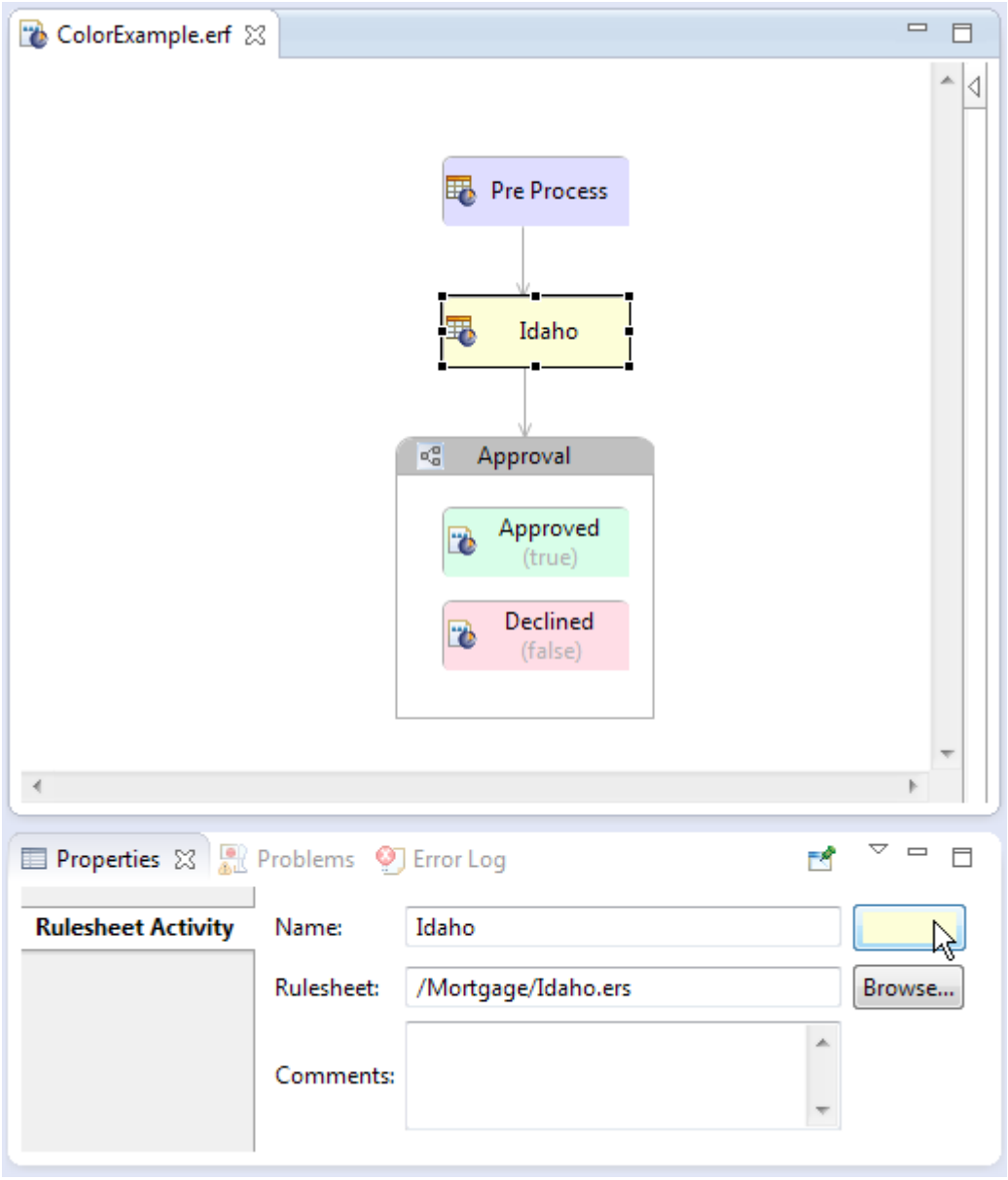
For more information, see *Using a Ruleflow in another Ruleflow in the Corticon Rule Modeling Guide* and *the "Ruleflows" section of the Quick Reference Guide*

### 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

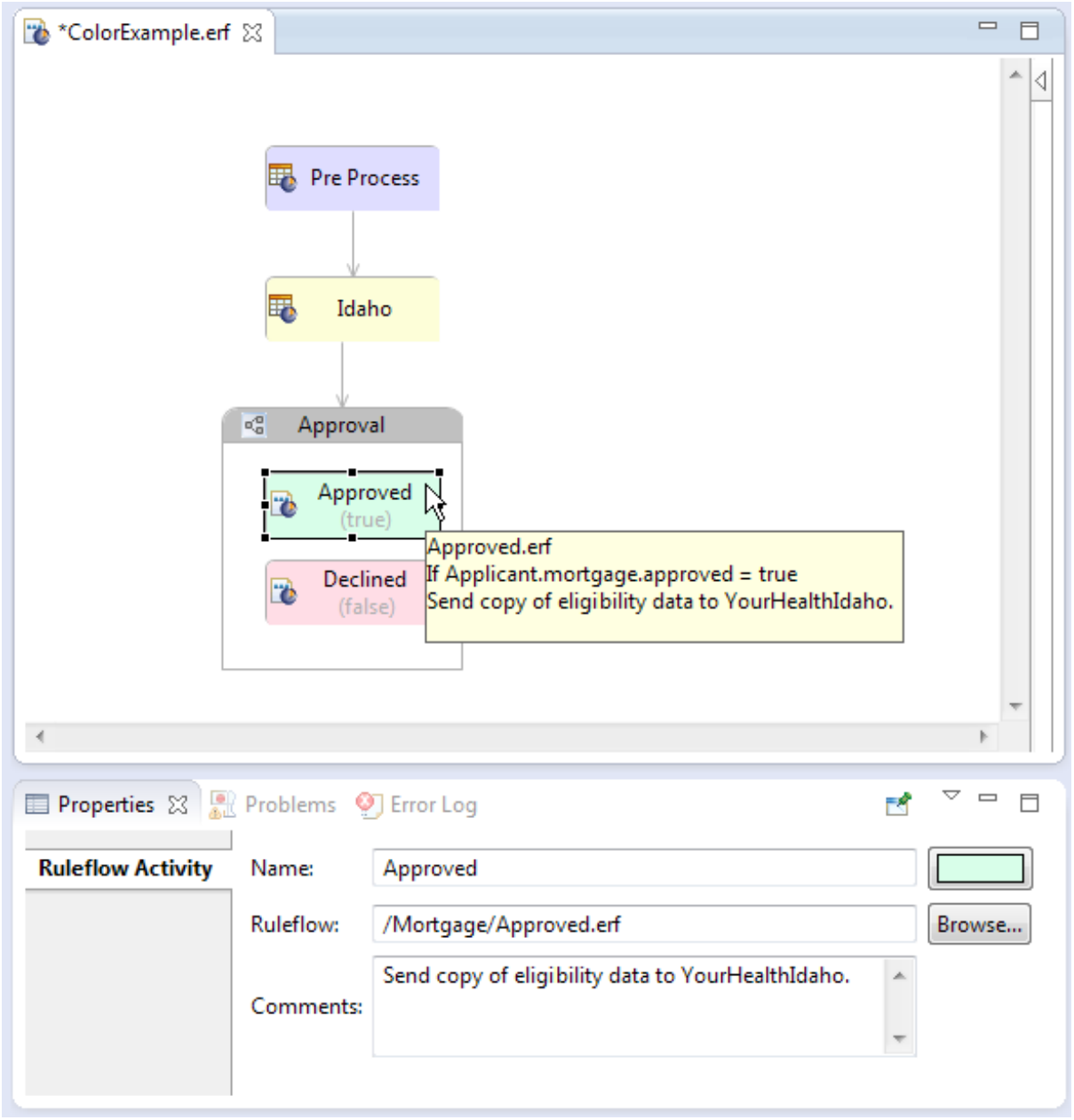
You can now add color to objects on the Ruleflow canvas to enhance the look and feel of the set of objects. Each object on the canvas provides a color button to set that objects color, as illustrated:




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 installler

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:

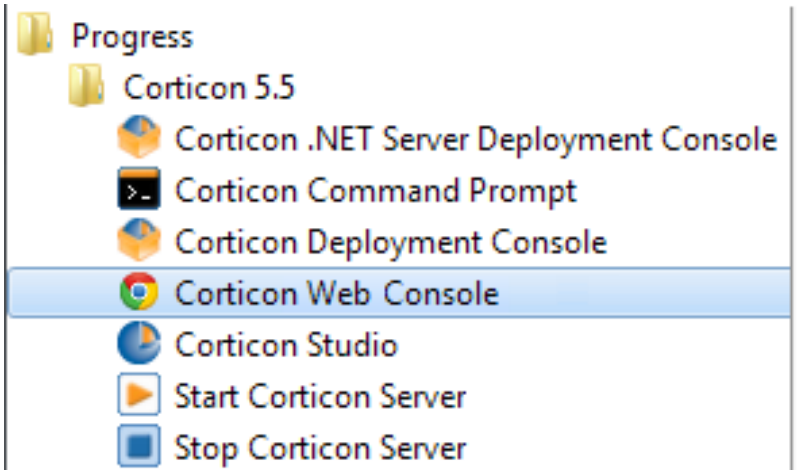
- If you want to deploy to JBoss or as an EAR file, download the server archive, and then extract the appropriate files.
- If you want to deploy `axis.war` to your own appserver, copy it from `[CORTICON_WORK]/pas/server/webapps`. (Note that this path was `[CORTICON_HOME]/pas/webapps` in 5.4 releases.)

# 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` | `<base>/decisionService/list`<br>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` | `<base>/decisionService/deploy`<br>Attempts to add a Decision Service to the server. Request objects will be sent as the content. |
| ***API Undeploy Decision Service*** | `HTTP DELETE` | `<base>/decisionService/undeploy`<br>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` | `<base>/decisionService/getProperties`<br>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` | `<base>/decisionService/setProperties`<br>Attempts to modify the properties of a specified Decision Service. |
| ***API Ping Server*** | `HTTP GET` | `<base>/server/ping`<br>Returns an object containing the current uptime of the server, which confirms that the server is reachable and running. |

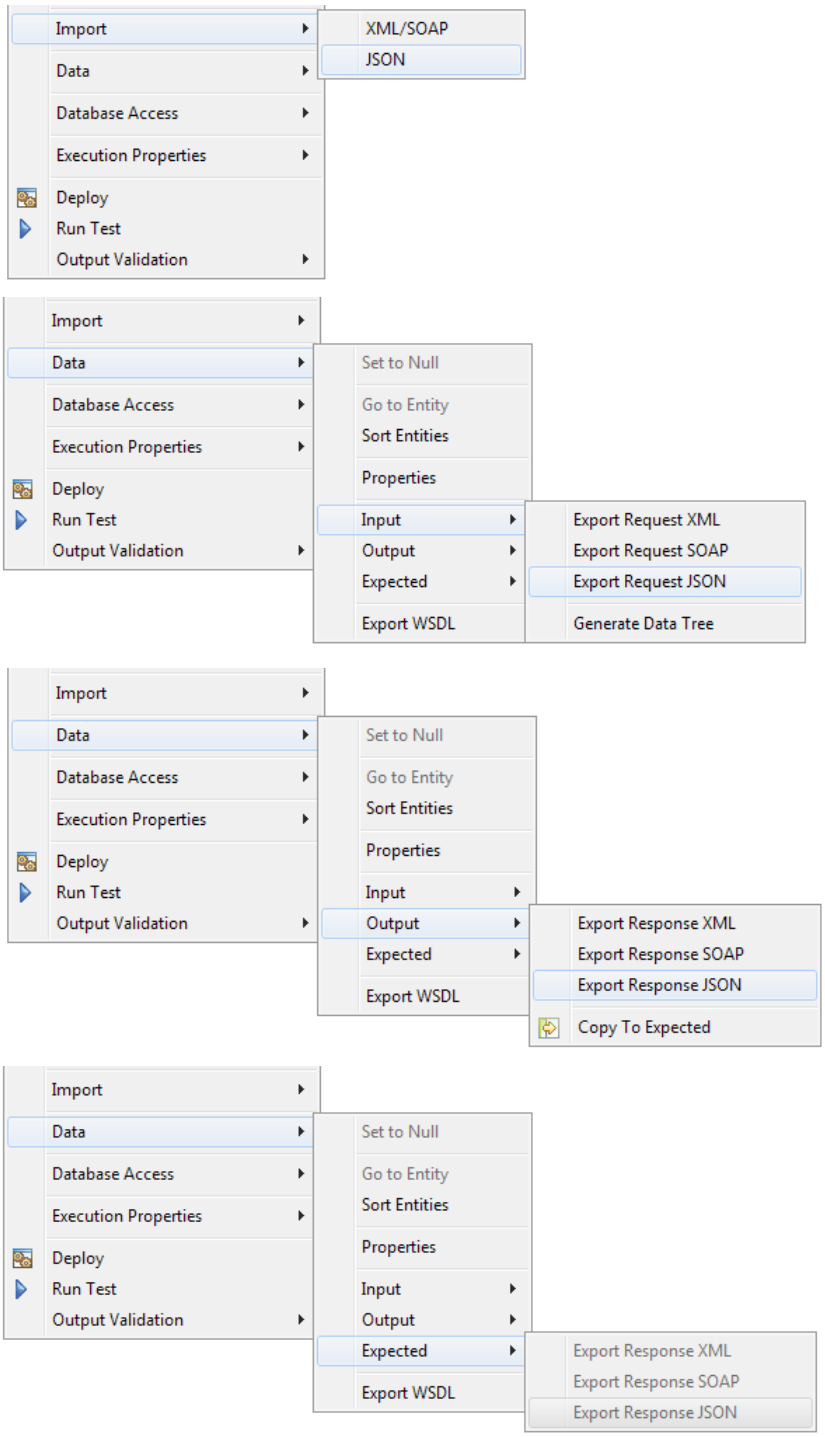
| Method | Type | Syntax and function |
|---|---|---|
| ***API Retrieve Metrics*** | `HTTP POST` | *<base>*`/server/metrics`<br>Retrieves metrics for the server. The request can pass in a Decision Service (or a list of Decision Services as an array), and a timestamp showing when the windowed metrics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

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... | ... |
|---|---|---|---|
| state | String | Yes | |

| Label | Value |
|---|---|
| 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 though 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:

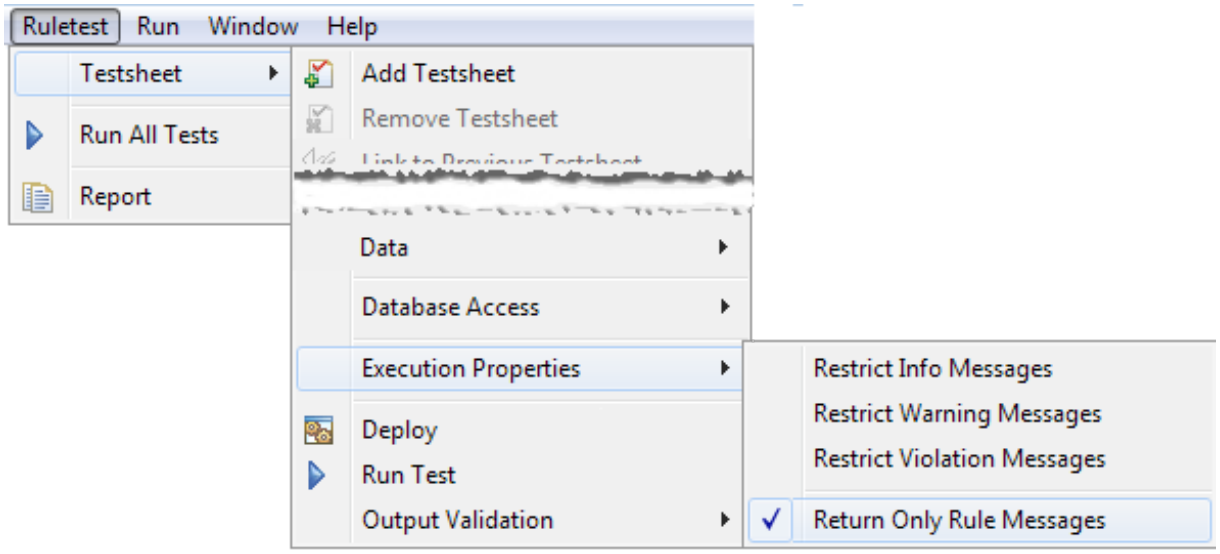
```
com.corticon.server.restrict.response.rulemessages.only
```

---

**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**.

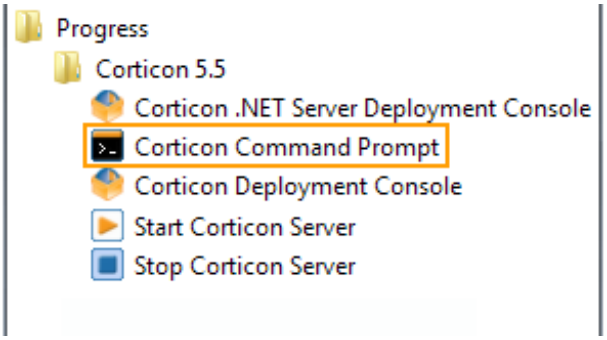
- On export, the checked **Execution Property** option generates an `ExecutionProperty` line set to `true`.

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

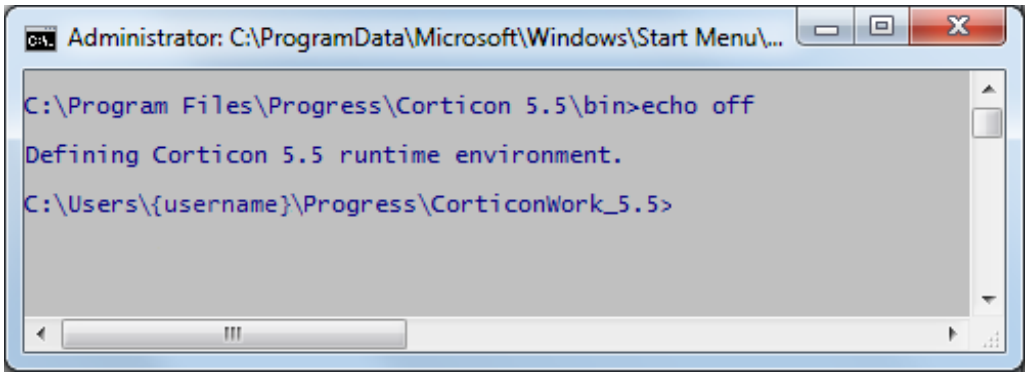
# Other additions and changes

## 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.

# A

# Progress Corticon documentation - Where and What

Corticon provides its documentation in various online and installed components.

**Access to Corticon tutorials and documentation**

| | |
|---|---|
| **Corticon Online Tutorials** | |
| Tutorial: Basic Rule Modeling in Corticon Studio | Online only. Uses samples packaged in the Corticon Studio. |
| Tutorial: Advanced Rule Modeling in Corticon Studio | Online only. |
| **Corticon Online Documentation** | |
| Progress Corticon User Assistance | Updated online help for the current release. |
| Corticon Server: Web Console Guide | Included in User Assistance. Not available in Studio help. |
| Progress Corticon Documentation site | Individual PDFs (including Web Console guide) 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 tutorials and documentation set**

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

| | |
|---|---|
| **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. |
| Tutorial: Advanced Rule Modeling in Corticon Studio | An introduction to complex and powerful functions in Corticon Business Rules Modeling Studio. Learn 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. |
| **Release and Installation Information** | |
| *What's New in Corticon* | Describes the enhancements and changes to the product since its last point release. |
| *Corticon Installation Guide* | Step-by-step procedures for installing all Corticon products in this release. |
| **Corticon Studio Documentation: Defining and Modeling Business Rules** | |
| *Corticon Studio: Rule Modeling Guide* | Presents the concepts and purposes the Corticon Vocabulary, then shows how to work with it in Rulesheets by using scope, filters, conditions, collections, and calculations. Discusses chaining, looping, dependencies, filters and preconditions in rules. Presents the Enterprise Data Connector from a rules viewpoint, and then shows how database queries work. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting of Rulesheets and Ruleflows. Includes *Test Yourself* exercises and answers. |
| *Corticon Studio: Quick Reference Guide* | Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions. |

| | |
|---|---|
| *Corticon Studio: Rule Language Guide* | Reference information for all operators available in the Corticon Studio Vocabulary. Rulesheet and Ruletest examples are provided for many of the operators. |
| *Corticon Studio: Extensions Guide* | Detailed technical information about the Corticon extension framework for extended operators and service call-outs. Describes several types of operator extensions, and how to create a custom extension plug-in. |
| **Corticon Enterprise Data Connector (EDC)** | |
| *Corticon Tutorial: Using Enterprise Data Connector (EDC)* | Introduces Corticon's direct database access with a detailed walkthrough from development in Studio to deployment on Server. Uses Microsoft SQL Server to demonstrate database read-only and read-update functions. |
| **Corticon Server Documentation: Deploying Rules as Decision Services** | |
| *Corticon Server: Integration and Deployment Guide* | An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the Deployment Console tool. 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 though logs, server monitoring techniques, performance diagnostics, and recommendations for performance tuning. |
| *Corticon Server: Deploying Web Services with Java* | Details setting up an installed Corticon Server as a Web Services Server, and then deploying and exposing Decision Services as Web Services on the Pacific Application Server (PAS) and other Java-based servers. Includes samples of XML and JSON requests. |
| *Corticon Server: Deploying Web Services with .NET* | 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 remote connection to a Web Console installation to enable manage Java and .NET servers in groups, manage Decision Services as applications, and monitor performance metrics of managed servers. |
| *Pacific™ Application Server for Corticon®: TCMAN Reference Guide* | Provides reference information about TCMAN, the command-line utility for managing and administering the Pacific Application Server. |