

# **Guide to Creating Corticon Extensions**



# Copyright

---

© 2017 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.

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 Developers Network, Rollbase, 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. Analytics360, AppServer, BusinessEdge, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustCode, JustDecompile, JustMock, JustTrace, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient 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.

**Updated: 2017/05/15**



# Table of Contents

<b>Chapter 1: Overview of Corticon extensions.....</b>	<b>7</b>
<b>Chapter 2: Using extensions when creating Decision Services.....</b>	<b>9</b>
<b>Chapter 3: Reviewing what is in the sample extensions.....</b>	<b>13</b>
What's in the Extended Operators Sample Projects.....	14
What's in the Service Callout Sample Projects.....	15
Service Callout Java and Rule Projects.....	15
Weather Callout Java and Rule Projects.....	17
<b>Chapter 4: Code conventions.....</b>	<b>21</b>
Using annotations.....	21
Imports and interfaces used in extensions.....	22
<b>Chapter 5: Using DataDirect drivers.....</b>	<b>25</b>
<b>Chapter 6: Creating custom extended operators.....</b>	<b>27</b>
<b>Chapter 7: Creating custom service callouts.....</b>	<b>31</b>
Specifying properties on a service callout instance.....	35
Access to Vocabulary Metadata .....	37
Using Advanced Data Callouts (ADC).....	38
<b>Chapter 8: Building the Java classes and JARs.....</b>	<b>69</b>
<b>Chapter 9: Deploying Decision Services with extensions.....</b>	<b>71</b>
<b>Appendix A: Access to Corticon knowledge resources.....</b>	<b>73</b>



---

# Overview of Corticon extensions

---

When you are creating business rules, you sometimes need to perform operations that are not built natively into Corticon. For example, you may need to apply a complex mathematical formula or to retrieve data from an external web service. Corticon provides the ability to add custom extensions for just such purposes.

Extensions are written as custom Java code that you package into one or more JAR files. You simply add extension jar files to your rule project to have them bundled into the EDS file for your Decision Service. This ease of adding extensions makes it easier to develop extensions yourself or to use open source extensions that you download from the Corticon community. By bundling extensions with EDS files, the EDS file becomes *self-contained*. You can deploy it to a Corticon Server without modifying the server's classpath. It also allows you to have different Decision Services, or versions of Decisions Services, running that use different versions of an extension.

---

**Note:** In releases prior to 5.6, you had to modify Corticon's Java classpath and perform other configuration steps to use an extension in your Decision Services. While the API has been enhanced, it is still compatible with extensions created in earlier releases.

---

When developing an extension, it needs to implement one or more Java interfaces that Corticon has defined. The [Corticon Extensions API](#) provides for Java annotations to describe the extension, thereby eliminating the need for additional configuration files.

When developing a project in Corticon Studio you can add extensions to your project through the project's **Properties** dialog box, so that they are available for development and running rule tests. The **Package and Deploy** wizard in Corticon Studio will include any extensions used by the project into the EDS file it generates or deploys. If you want to script the building of your EDS files, you can also use the Corticon ANT scripts to package extensions into EDS files.

For compatibility with previous releases you can still place extension JAR files on the Corticon classpath so that they are available to all Decision Services.

Extensions can be created in the Java development environment included in Corticon Studio, or you can use another IDE.

There are two types of Corticon extensions:

- **Extended operators** - Operators are used when defining conditions and actions in a Rulesheet. While Corticon has a large built-in set of operators, you can expand this set by adding custom operators. Operators can operate on individual attributes, collections or sequences. Examples include:
  - Financial functions, such as net present value, and loan amortization
  - Statistical functions, such as standard deviation, and permutations
  - Engineering functions, such as pi, sine, and cosine
- **Service callouts** - Callouts can be used in a ruleflow to retrieve, modify, or store data that is being processed by the rules. The most common use is to access data in a database or external web service. For example, if your Ruleflow needs to look up an applicant's credit rating, the service callout can have a step in the Ruleflow processing that calls out to a trusted realtime ratings provider, and then adds the response back into the decision processing.



---

## Using extensions when creating Decision Services

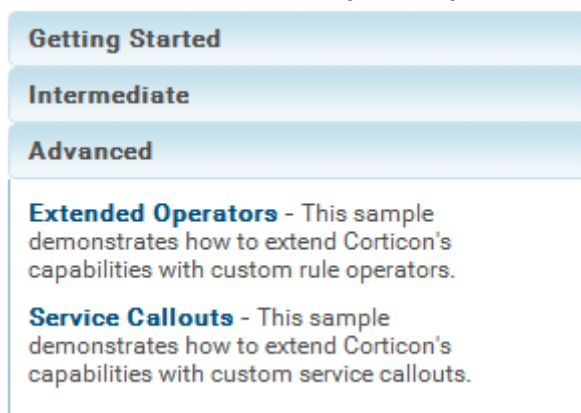
---

You might want your project to include extensions that are already packaged and ready to use. The sample extensions bundled with Corticon Studio provide sample Rule projects with their samples already packaged into JARs.

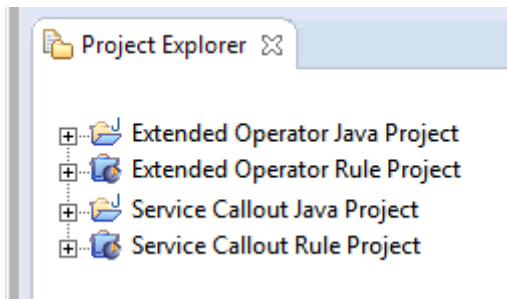
For now, assume that you just want to use the functionality in these extensions.

**To use the packaged extension samples in my project:**

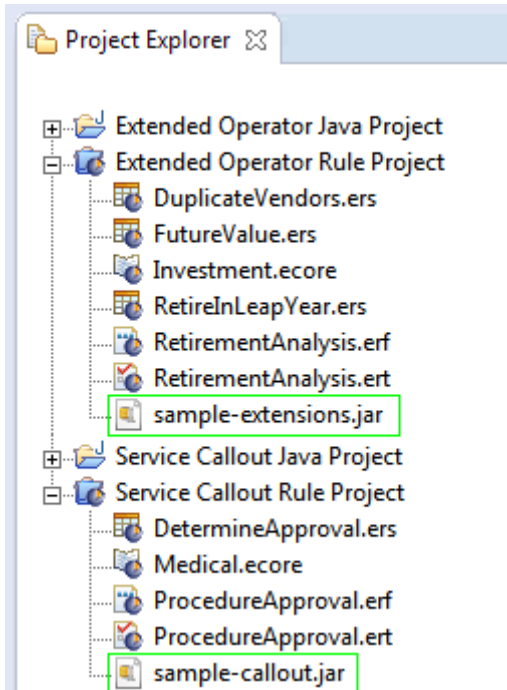
1. In Corticon Studio, choose **Help > Samples**. Locate the **Advanced** samples:



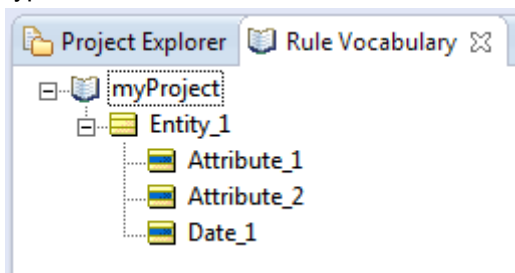
2. Select **Extended Operators**, click **Open**, and then click **OK**.
3. Then do the same to open the **Service Callouts** sample.
4. Your Studio's **Project Explorer** lists the two samples, each with its Java project and its Rules project:



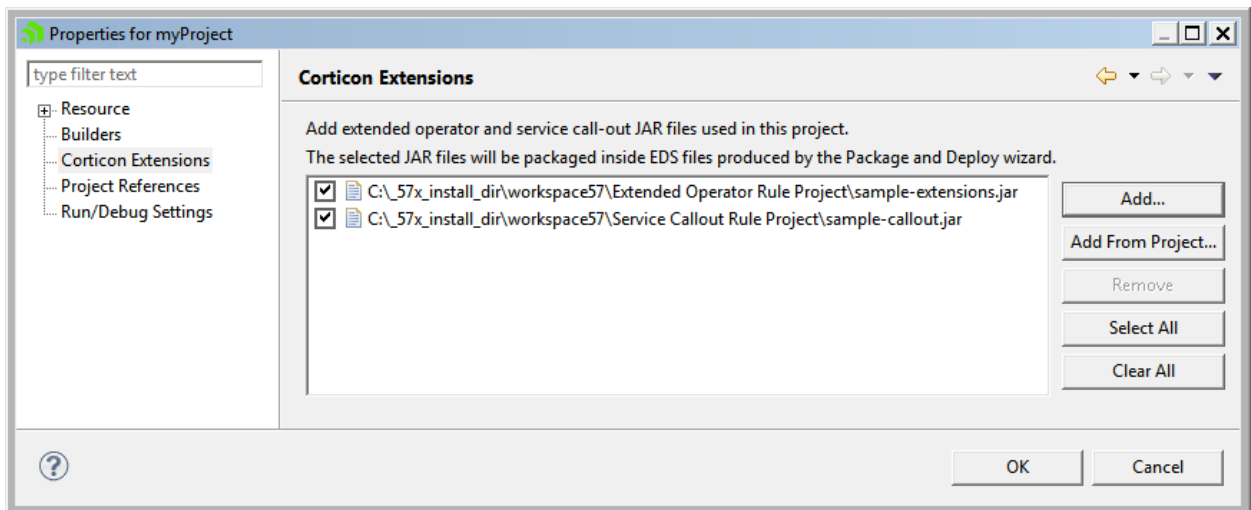
5. Expanding the two Rule Projects, you can see that each has a related samples JAR.



6. Create a new Rule Project named `myProject` and create a very simple Vocabulary that includes a date type:



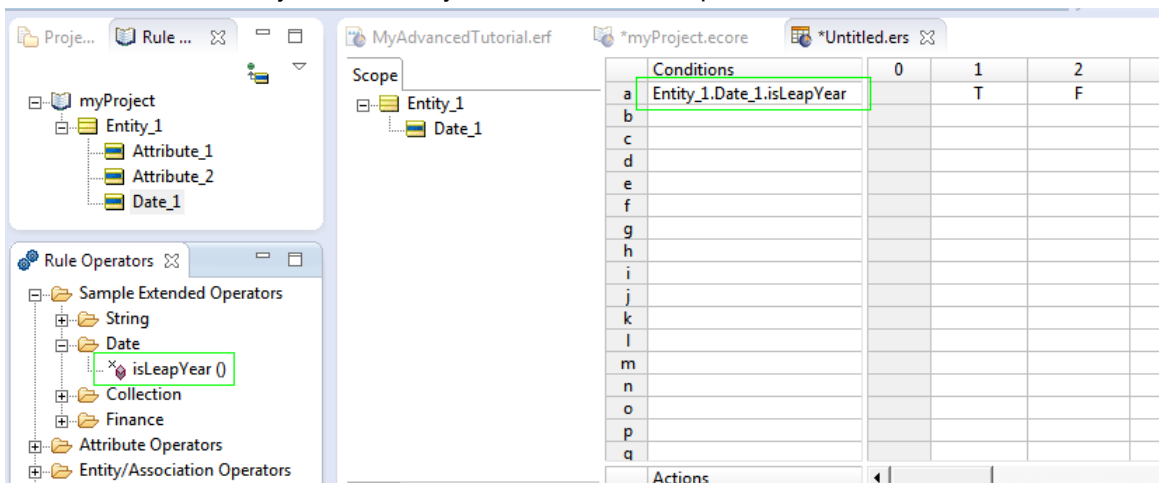
7. In the Project Explorer, click on `myProject`, and then select **Properties**. Select **Corticon Extensions**. Click **Add** then navigate to each of the sample Rule Projects to select its extensions JAR, as shown:



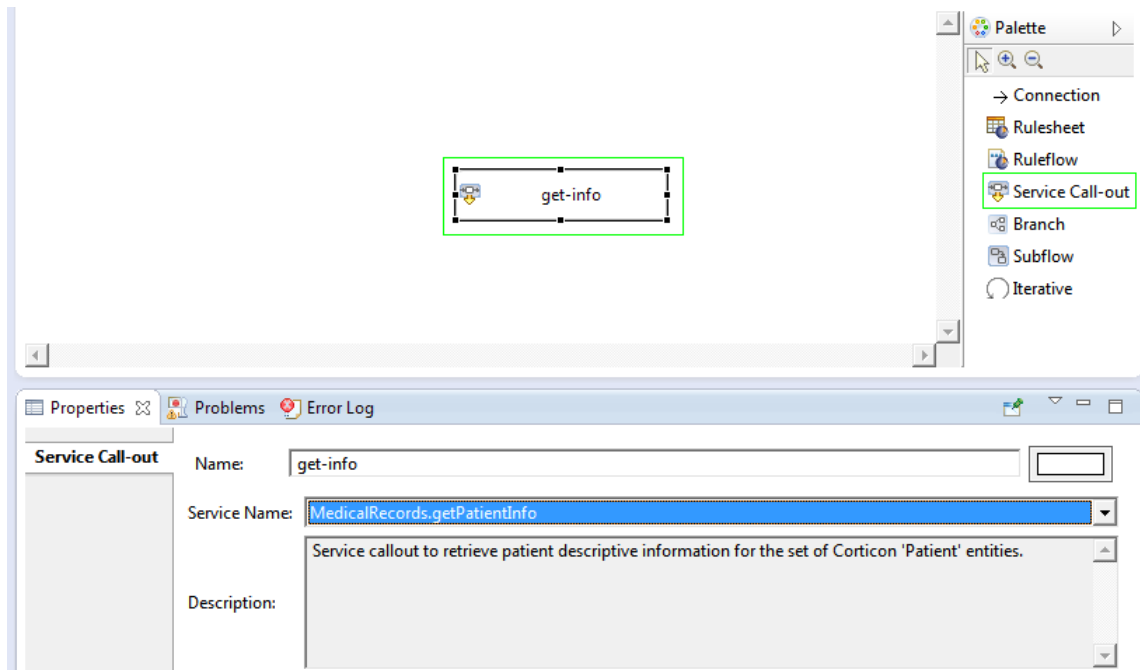
While it is good to copy extension JARs you want to use into your project, you can navigate to other locations to add them to your project

**Note:** This feature does not support JARs nested within a JAR that you add to a project.

8. Because the JARs were referenced in their original locations, they are not listed in `myProject` folder. If your extensions requires other JARs, you can add them here.
9. Create a Rulesheet in `myProject`. Note that the Rule Operators tab adds in the extended operators that are in the JAR, so that you can readily use one as a valid operator in the Rulesheet, as shown:



10. Create a Ruleflow in `myProject`. Click **Service Call-out** on the palette, click on the canvas, and then name it. On the **Properties** tab, click the Service Name dropdown to see the service callouts that are packaged in the sample JAR you added to the project, as shown when `getPatientInfo` was selected:



Because the two extension JARs are properties of the project, they are embedded in Decision Services that you package and deploy from Studio.

The next section looks at the source code in each of their Java projects to gain insight into how extensions are created and prepared for use.

## Reviewing what is in the sample extensions

---

Both the Extended Operator and Service Callout samples contain Java projects that show how you can create the sample extension JAR yourself, and a Corticon Rules project that demonstrates those extensions.

There are two sets of extension samples:

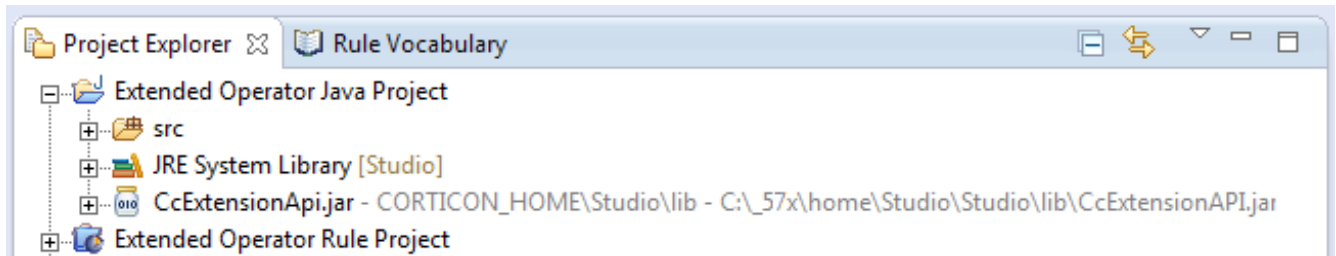
- **Extended Operator Sample Projects** - The Extended Operators sample contains the source code for several extended operators and a rule project that uses them. The rule project uses extended operators for determining the future value of an investment, if a collection contains duplicate strings, and if a given date is a leap year.
- **Service Callout Sample Projects** - The Service Callout sample contains the source code for several service callouts and a rule project that uses them. The rule project uses service callouts to retrieve and update patient medical data in a pseudo external service. Accessing web services or other external datasources is a common use case for service callouts.

For details, see the following topics:

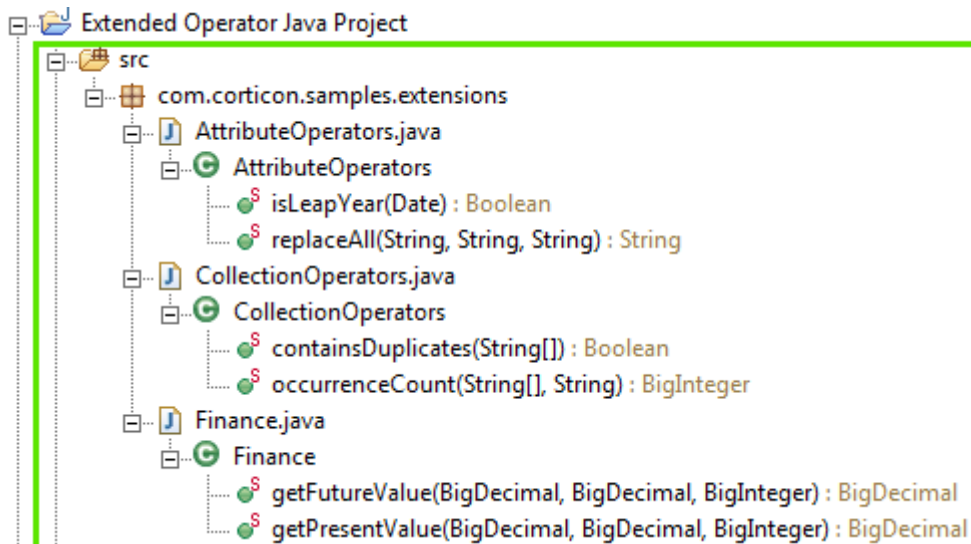
- [What's in the Extended Operators Sample Projects](#)
- [What's in the Service Callout Sample Projects](#)

## What's in the Extended Operators Sample Projects

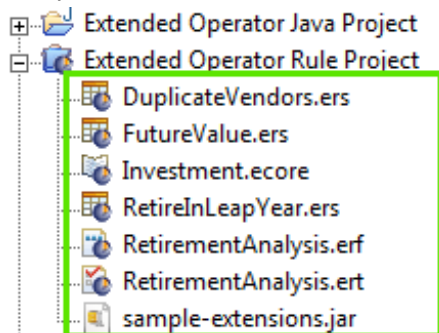
The **Extended Operator Java project** contains a `src` folder with the source code for the extended operators and has on its build path the standard Java system library and `CcExtensionApi.jar` with the Corticon extension API:



The `src` folder contains the source code for three Java classes that implement extended operators:



The **Extended Operator Rule Project** is a set of rule assets that demonstrate the extended operator Java samples:

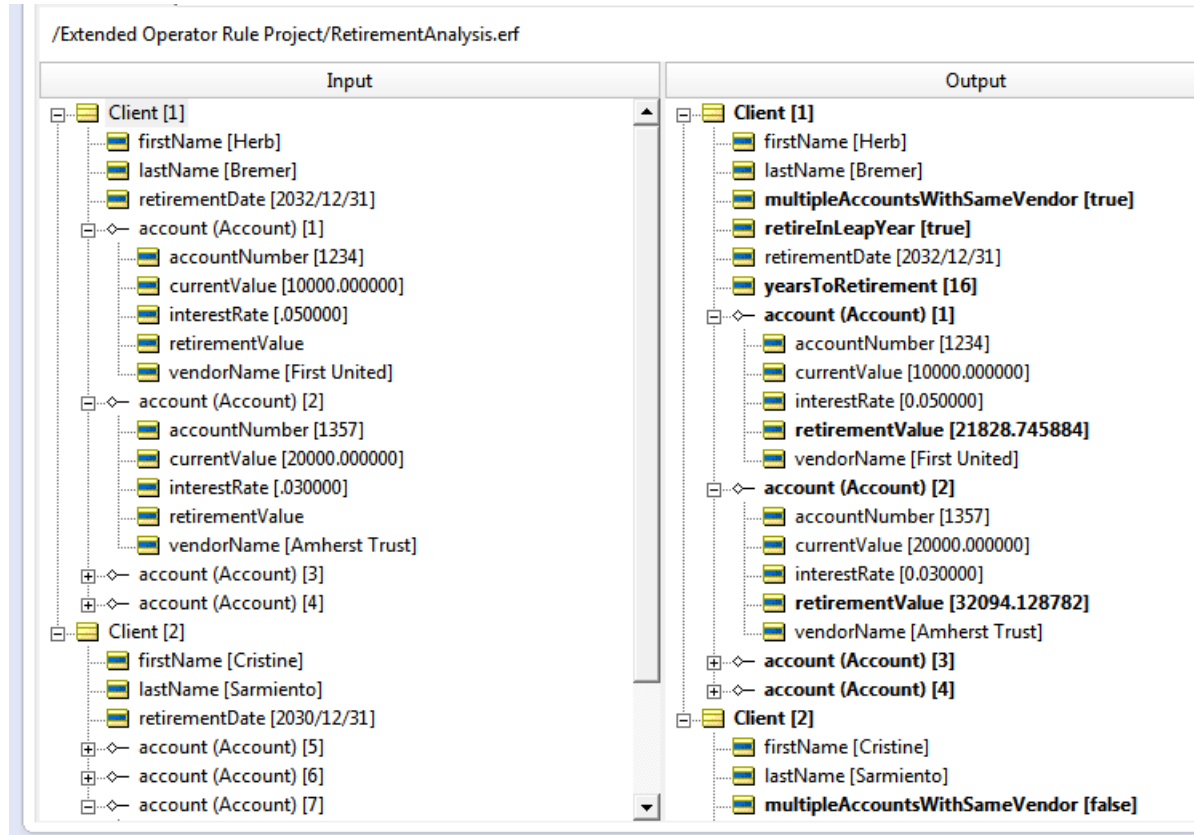


The **Extended Operator Rule project** uses the extended operators and the supporting vocabulary, Ruleflows, and Ruletests for the project. It also contains the JAR file of the extensions built from the Java project, `sample-extensions.jar`.

The sample project contains a Vocabulary, and three Rulesheets that demonstrate each of the new extended operators:

1. FutureValue - Determines years to retirement and the future value of the current investment with a constant interest rate.
2. RetireInLeapYear - Checks whether the retirement year is a leap year
3. DuplicateVendors - Checks to determine that there are not multiple accounts with the same vendor

The Ruleflow chains the three Rulesheets together for the Ruletest to see that the extended operators behave as expected.

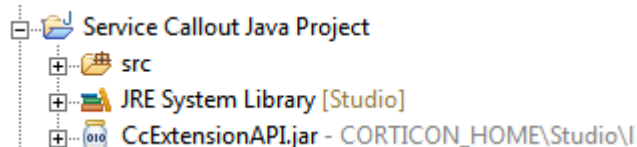


## What's in the Service Callout Sample Projects

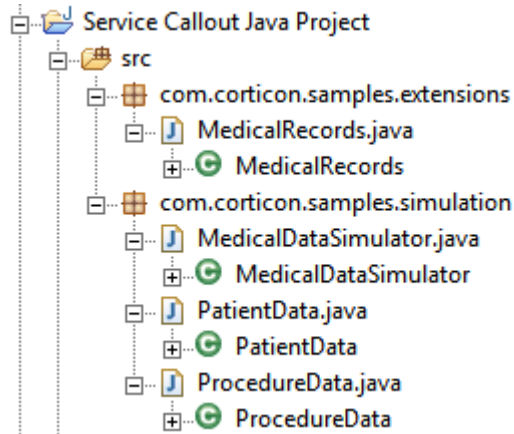
A Corticon Studio installation includes two Service Callout sample projects that you can immediately bring in to your workspace, and then run.

### Service Callout Java and Rule Projects

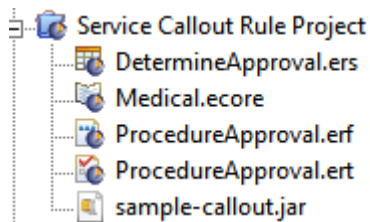
The **Service Callout Java Project** contains a `src` folder with the source code for the service callouts and has on its build path the standard Java system library and `CcExtensionApi.jar` with the Corticon extension API:



The `src` folder contains the source code for four Java classes that implement service callouts:

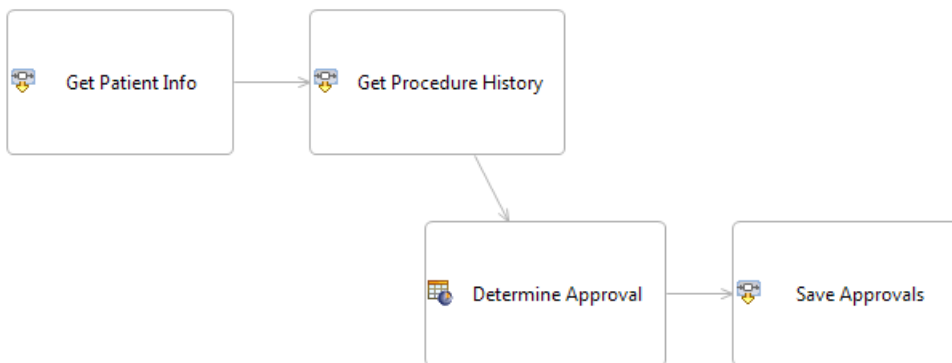


The **Service Callout Rule Project** is a set of rule assets that demonstrate the Service Callout Java samples:



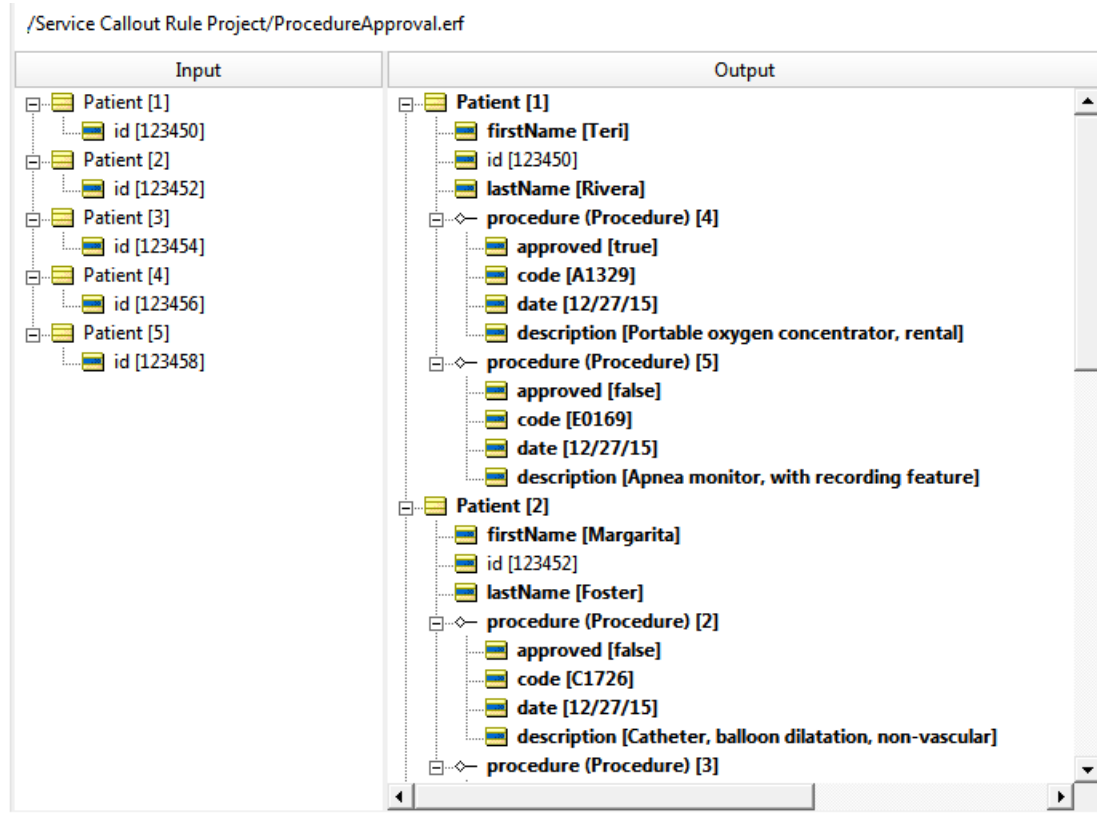
The project uses the service callouts and the supporting vocabulary, Rulesheet, Ruleflow, and Ruletest for the project. It also contains the JAR file of the extensions built from the Java project, `sample-callouts.jar`.

The Ruleflow uses three service callouts and a Rulesheet to perform its functions:



The Ruletest shows that the service callouts behave as expected:





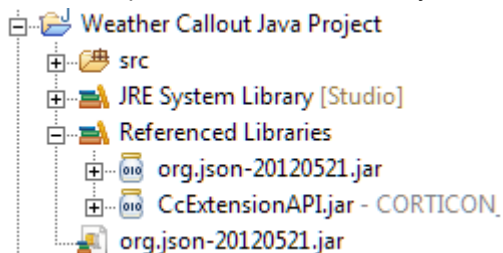
The **Service Callouts** sample is accessed from the Studio's **Help > Samples** in the **Advanced** section.

## Weather Callout Java and Rule Projects

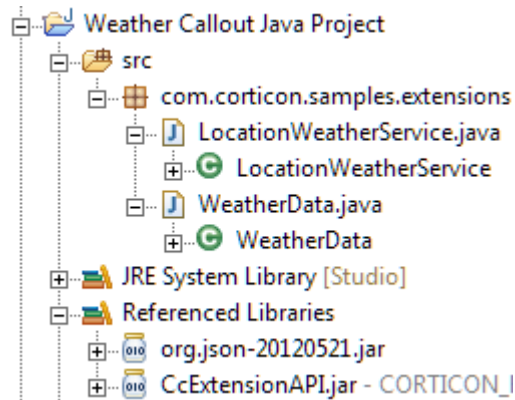
Corticon Studio includes a sample project that shows how service callouts can be reused when they can be parameterized so that each instance can have a different configuration.

The **Weather Callout Java Project** is a Java project that includes the source code for a service callout to call the REST API on [OpenWeatherMap.org](http://OpenWeatherMap.org) to retrieve weather data for individual cities. The **Weather Callout Rule Project** uses the callout in the Java project to retrieve data for cities specified in `Location` entities. The service callout is specific to the vocabulary in that it looks for `Location` entities with specific attributes. To retrieve live weather data for a city you need to create an account on <http://openweathermap.org/>, and then generate an API key that you provide as a property on the callout. By default, the keyword 'demo' is used, and sample weather data is generated.

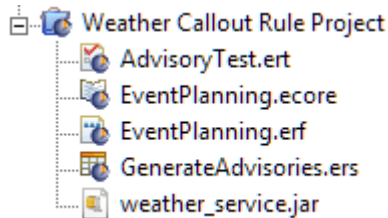
The **Weather Callout Java project** contains a `src` folder with the source code for the weather callout and has on its build path the standard Java system library, `org.json-20120521.jar`, and `CcExtensionApi.jar`.



The `src` folder contains the source code for two Java classes.

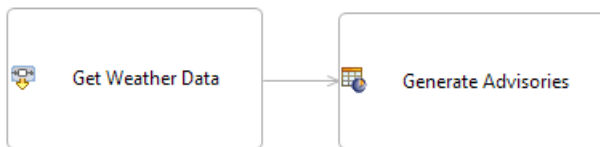


The **Weather Callout Rule Project** is a set of rule assets that demonstrate the Weather Callout Java samples.

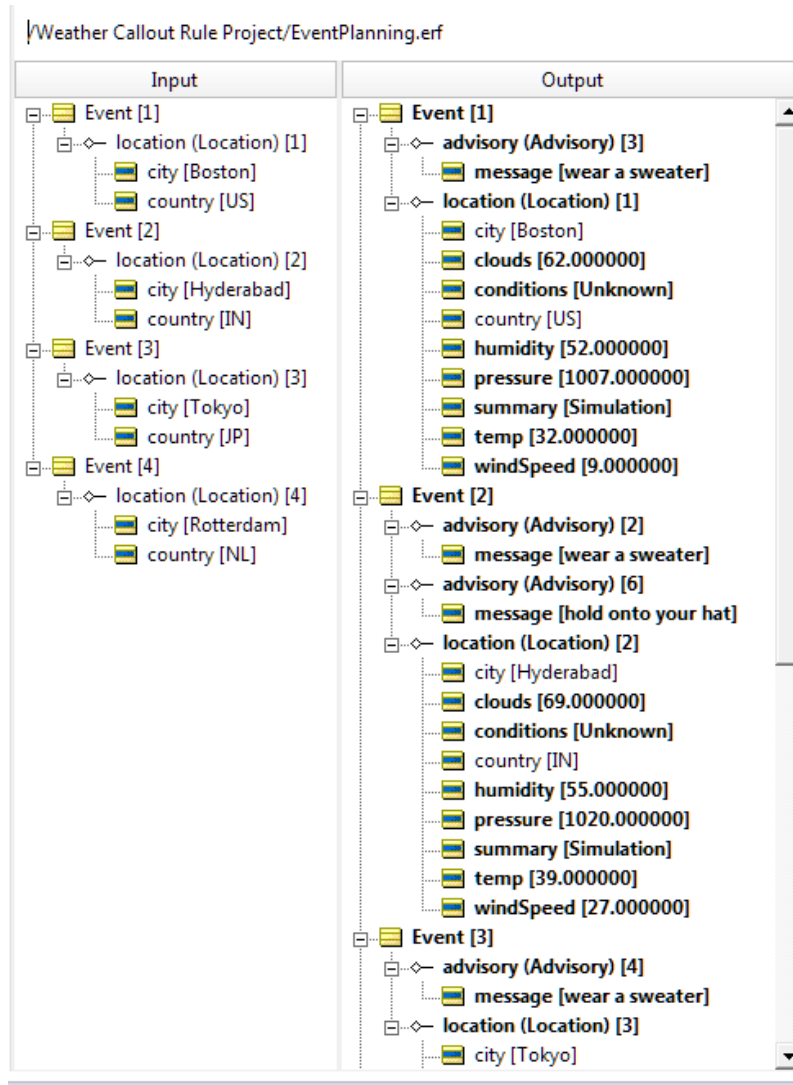


The project uses the service callout and the supporting vocabulary, Rulesheet, Ruleflow, and Ruletest for the project. It also contains the JAR file of the extensions built from the Java project, `weather_service.jar`.

The Ruleflow uses a service callout and a Rulesheet to perform its functions.



The Ruletest shows that the weather callout behaves as expected.



The **Weather Callout** sample is accessed from the Studio's **Help > Samples** in the **Advanced** section.



---

## Code conventions

---

For details, see the following topics:

- [Using annotations](#)
- [Imports and interfaces used in extensions](#)

## Using annotations

Corticon extensions use Java annotations to get information about extensions. Corticon supports four types of annotations:

- **Class annotations:**
  - `@TopLevelFolder` - The name of the top level folder that will contain the extended operator. The operator tree supports two levels of folders; a top level folder and an operator folder. All operators defined in the class will be under a subfolder of the top level folder defined for the class..
- **Method annotations**
  - `@Description` - The text that describes the operator in the Rule operator tooltip or the description of the service callout in the Ruleflow properties. Note that this is typically the only annotation type used withh service callouts.
  - `@OperatorFolder` - The name of the subfolder, under top level folder, for the operator defined by the method.
- **Parameter annotations**

- `@ArgumentName` - The name of the argument shown in the function signature part of the tool tip.

### Localizing Annotations

Annotations offer a format that allows localization. In its basic format, you can just enter

```
@Annotation("text").
```

You can choose to add a list of one or more locales with corresponding strings for each locale. For example:

```
@Annotation(lang={"en","fr"}, values={"text","texte"})
```

### Creating multi-line annotations

Some annotations provide the help that the user sees when they hover over an operator in the **Rule Operator** tab. Often the description can be improved by adding line returns

You can embed line returns in your description by using the `"\n text" +` convention, as shown:

```
@Description(lang = { "en" }, values = { "  
  \n Replace all occurrences of a substring" +  
  \n within a string" +  
  \n with another string."  
})
```

The following excerpt from `AttributeOperators.java` highlights usage of Corticon extension annotations:

```
...  
@TopLevelFolder("Sample Extended Operators")  
public class AttributeOperators implements ICcDecimalExtension,  
    ICcStringExtension, ICcDateTimeExtension {  
  
    @OperatorFolder(lang = { "en" }, values = { "String" })  
    @Description(lang = { "en" }, values = { "Replace all occurrences of a substring with a  
string with another string." })  
    public static String replaceAll(String s,  
        @ArgumentName(lang = { "en" }, values = { "searchString" }) String searchString,  
        @ArgumentName(lang = { "en" }, values = { "replacement" }) String replacement) {  
        ...  
    }  
}
```

## Imports and interfaces used in extensions

### Annotations

**Extended Operators** - Extended operators load four annotation types:

```
import com.corticon.services.extensions.ArgumentName;  
import com.corticon.services.extensions.Description;  
import com.corticon.services.extensions.OperatorFolder;  
import com.corticon.services.extensions.TopLevelFolder;
```

**Service Callouts** – Service callouts load one annotation type:

```
import com.corticon.services.extensions.Description;
```

## Interfaces

**Extended Operators** – The interfaces added are those required for the data types used in the extended operator class, selected from the following:

```
import com.corticon.services.extensions.ICcCollectionExtension;
import com.corticon.services.extensions.ICcDateTimeExtension;
import com.corticon.services.extensions.ICcDecimalExtension;
import com.corticon.services.extensions.ICcIntegerExtension;
import com.corticon.services.extensions.ICcSequenceExtension;
import com.corticon.services.extensions.ICcStandAloneExtension;
import com.corticon.services.extensions.ICcStringExtension;
```

**Note:** While extended operators are limited to returning a value, the standalone extended operator type can access the interfaces for `ICcDataObject`:

```
import com.corticon.services.extensions.ICcStandAloneExtension;
import com.corticon.services.dataobject.ICcDataObject;
import com.corticon.services.dataobject.ICcDataObjectManager;
```

When these interfaces are loaded, extended standalone operator methods can define `ICcDataObjectManger` as their first parameter. This provides flexibility in integrating Corticon with databases and other external services to perform complex actions, such as retrieving a set of records from a web service, and then adding them as associations on an entity. The `ICcDataObjectManager` parameter is not allowed in rule syntax.

**Service Callouts** – A service callout adds the following interface:

```
import com.corticon.services.extensions.ICcServiceCalloutExtension;
```

## Extended operator data type mappings

The mapping of parameter and return types for Extended Operators are as follows:

Java Type	Corticon Type
<code>java.math.BigInteger</code>	Integer
<code>java.math.BigDecimal</code>	Decimal
<code>java.lang.Boolean</code>	Boolean
<code>java.util.Date</code>	Date, Time or DateTime
<code>java.lang.String</code>	String





---

## Using DataDirect drivers

---

If you need custom database access beyond that provided by Corticon's Enterprise Data Connector (EDC) or Advanced Data Callout (ADC) you can now use the DataDirect® drivers bundled with Corticon in your extensions and wrappers. Corticon provides a new factory method for getting a connection to a database. It connects to a database using a DataDirect driver and returns a standard `java.sql.Connection` object. You work with this `Connection` object the same as you would any `Connection` in Java.

The `ICcDataObjectManager` class now provides a method to retrieve an instance of `IDatabaseDriverManager`. This new class provides the `getConnection(...)` method that can be used to create a database connection using a bundled DataDirect driver. See the Corticon JavaDoc for details on these classes and methods.

---

**Note: EDC License** - A valid EDC license is required, otherwise the factory method returns an error.

---

### Get a connection

The class `CcDatabaseConnectionFactory` opens a connection to the database and processes queries. It contains a method that returns a `java.sql.Connection` interface to open a connection using the DataDirect driver and returning it. The `getConnection()` method returns a `java.sql.Connection` from these signatures:

```
getConnection(String dataSourceId, String driverId, String connectionString, String
username, String password)
```

- `String: dataSourceId` - The JNDI `dataSource` Id. This value is used to lookup an existing connection.
- `String: driverID` - The Id (as outlined in `DatabaseDefinitions.xml`) for the DataDirect driver.
- `String: ConnectionString` - Driver connection string, in the same format as EDC connection in Corticon Studio. For example, `jdbc:progress:openedge://hostname:5566;databaseName=corticon`
- `String: username` - username for logging into the database.

- String: `password` - password for logging into the database.

```
getConnection(String dataSourceId, String driverId, String host, int Port, String
databaseName, String username, String password)
```

- String: `dataSourceId` - The JNDI `dataSource` Id. This value is used to lookup an existing connection.
- String: `driverID` - The Id (as outlined in `DatabaseDefinitions.xml`) for the DataDirect driver.
- String: `host` - The hostname of the database server for connection.
- Int : `Port` - Port number of the database server for connection.
- String: `databaseName` - the name of the database for connection on the server .
- String: `username` - username for logging into the database.
- String: `password` - password for logging into the database.

```
getConnection(String dataSourceId)
```

- String: `dataSourceId` - The JNDI `dataSource` Id. This value is used to lookup an existing connection.

```
getConnection(Properties connectionProperties)
```

- Properties: *properties* - Object passed with each of the above items as fields. This also lets you specify additional connection parameters. Constants for the properties fields will be supplied.

### Close a connection

When you have completed processing requests, you can either:

- Close the connection using the `close()` method in the `Connection` interface. In some cases, you want to immediately release a connection's database and JDBC resources instead of waiting for them to be automatically released; the `close()` method provides this immediate release.
- Leave the connection up when connection pooling is being used.
- Leave the connection open until the JVM exits or the class gets garbage collected.

---

## Creating custom extended operators

---

---

**Note:** Corticon Studio is built on Eclipse which provides a Java development environment you can use for creating Corticon extensions that you can use in current and future versions of Corticon. If you want to create extensions in a separate IDE, you must use Java 1.7.0\_05 or higher.

---

---

**Note:** You might want to simply copy the sample project **Extended Operator Java Project**, and then tweak a sample such as `AttributeOperators.java` by renaming the `TopLevelFolder` to `mySampleExtendedOperators` for your first run. You can then go ahead to [Building the Java classes and JARs](#) on page 69.

---

For many developers, the quickest way to learn is by example. You might want to compare the three Java source files in the **Extended Operator Java Project** to see what is common and what changes. In this example, the `AttributeOperators.java` is presented.

### 1. Specify the imports and interfaces you will need.

```
package com.corticon.samples.extensions;

import java.util.Calendar;
import java.util.Date;

import com.corticon.services.extensions.ArgumentName;
import com.corticon.services.extensions.Description;
import com.corticon.services.extensions.ICcDateTimeExtension;
import com.corticon.services.extensions.ICcStringExtension;
import com.corticon.services.extensions.OperatorFolder;
import com.corticon.services.extensions.TopLevelFolder;
```

This class imports the Corticon `ICcStringExtension` and `ICcDateTimeExtension` interfaces because it will implement extended operators for `String` and `DateTime` attributes. The other Corticon imports are for the annotations which will be used to describe the extensions.

2. Enter your comments that describe the class.

```
/**
 * This class provides sample Corticon stand-alone extended operators.
 * Extended operators are a means to add custom features to Corticon for
 * use in Corticon rules.
 *
 * The samples in this class provide simple operators for calculating the
 * present and future value of an investment for a number of years at a given
 * interest rate.
 */
```

A general description of this source file is always good coding practice. It has no use outside of the source file.

3. Specify the `TopLevelFolder` name.

```
@TopLevelFolder("Sample Extended Operators")
```

The `TopLevelFolder` annotation identifies the folder that will group the extended operators on the **Rule Operators** tab in Corticon Studio. You can name the folder to fit your needs, such as "My Operators", or "Financial Operators".

4. Specify the class and its implementations.

```
public class AttributeOperators implements ICcStringExtension, ICcDateTimeExtension {
```

5. Name your operator folder, and use the locale parameters if appropriate.

```
@OperatorFolder(lang = { "en" }, values = { "Date" })
```

The `OperatorFolder` defines the subfolder that will list an individual operator within the `TopLevelFolder` on the **Rule Operators** tab in Corticon Studio. You can organize and name folders to fit your needs.

6. Add your description of the operator, and use the locale parameters if appropriate..

```
@Description(lang = { "en" }, values = { "Returns true if the date is in a leap year."
})
```

The `Description` annotation describes the specific operator. The hover help reveals what is passed, what is returned, and description text for the locale.

7. Write your actual implementation of the extended operator. It is always `public static`.

```
public static Boolean isLeapYear(Date d) {
    if (d == null)
        return null;

    Calendar c = Calendar.getInstance();
    c.setTime(d);

    int year = c.get(Calendar.YEAR);
    if ((year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0)))
        return true;
    else
        return false;
}
```

The example takes a date and returns a boolean. It is up to you to determine that this produces the result you want, and that you can verify it across a range of values and error conditions.

- 
8. The sample includes another operator definition using the same structure, this one for the **String** **OperatorFolder**. You can similarly change this section, or just cut the whole section out.

```
/**
 * Replaces all occurrences of a substring in a string with another string.
 *
 * @param s A string.
 * @param searchString The substring to look for in s.
 * @param replacement The string to replace it with.
 * @return The original string with all instances of searchString replace by
 * replacement.
 */
@OperatorFolder(lang = { "en" }, values = { "String" })
@Description(lang = { "en" }, values = { "Replace all occurrences of a substring within
a string with another string." })
public static String replaceAll(String s,
    @ArgumentName(lang = { "en" }, values = { "searchString" }) String searchString,
    @ArgumentName(lang = { "en" }, values = { "replacement" }) String replacement) {
    if (s == null)
        return null;

    if (searchString == null)
        return s;

    String r = s.replaceAll(searchString, replacement);
    return r;
}
}
```

9. Save your work, and then go ahead to [Building the Java classes and JARs](#) on page 69.

---

**Note:** Compatibility of extensions created in an earlier release, any extension operators and service callouts that are in `extended.core.jar` are shared across all Rule Projects. As a result, such extensions are always in the **Rule Operator** tab in every editor. Then, you can add your extended operators and service callouts to specific Rule Projects using the new mechanism.

---



---

## Creating custom service callouts

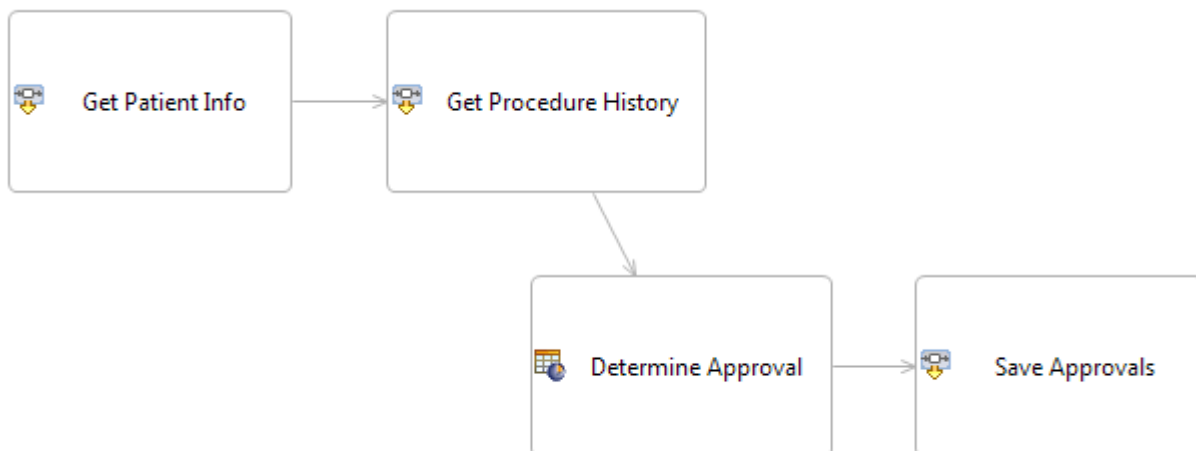
---

Service Call-out extensions are user-written functions that can be invoked in a Ruleflow.

In a Ruleflow, the flow of control moves from Rulesheet to Rulesheet, with all Rulesheets operating on a common pool of facts. This common pool of facts is retained in the rule execution engine's working memory for the duration of the transaction. Connection arrows on the diagram specify the flow of control. Each Rulesheet in the flow may update the fact pool.

When you add a Service Call-out to a Ruleflow diagram, you effectively instruct the system to transfer control to your extension class at a specific point in the flow. Your extension can directly update the fact pool, and your updated facts are available to subsequent Rulesheets.

Consider the sample:



The rule flow uses two service callouts (Get Patient Info and then Get Procedure History), then uses the data in the Determine Approval Rulesheet, and finally passes control to Service Callout extension class Save Approvals.

Your Service Call-outs use the Progress Corticon Extension API to retrieve and update facts. The package `com.corticon.services.dataobject` contains two Java interfaces:

Interface	Purpose
<code>com.corticon.services.dataobject.ICcDataObjectManager</code>	Provides access to the entire fact pool. Allows you to create, retrieve and remove entity instances.
<code>com.corticon.services.dataobject.ICcDataObject</code>	Provides access to a single entity instance. Allows you to update the entity instance, including setting attributes and creating and removing associations.

Your Service Call-out class must implement marker interface `ICcServiceCalloutExtension`.

Here is the source code for the service callout `MedicalRecords.java`:

```

/*
 * Copyright (c) 2016 by Progress Software Corporation. All rights reserved.
 */

package com.corticon.samples.extensions;

import java.util.ArrayList;
import java.util.Set;

import com.corticon.services.dataobject.ICcDataObject;
import com.corticon.services.dataobject.ICcDataObjectManager;
import com.corticon.services.extensions.Description;
import com.corticon.services.extensions.ICcServiceCalloutExtension;
import com.corticon.samples.simulation.*;

/**
 * This class provides sample Corticon service callouts. Callouts provide
 * a means to add custom features to Corticon for use in Corticon ruleflows.
 * Callouts are for integrating with external systems such as webservices.
 * In a callout you can create and delete instances of Corticon entities,
 * set their properties and define associations.
 *
 * The samples in this class provide a simulation of retrieving patient
 * medical data given a patient id. The class MedicalDataSimulator provides
 * a static set of patient and procedure data. In a real implementation
 * this could be a class which gets data from a webservice, database, or
 * other source.
 */
public class MedicalRecords implements ICcServiceCalloutExtension {

    private static MedicalDataSimulator md = new MedicalDataSimulator();

    /**
     * Service callout to retrieve patient descriptive information for
     * the set of Corticon "Patient" entities.
     *
     * This callout demonstrates how to iterate over a set of Corticon
     * entities and set attributes on them.
     */
    @Description(lang = { "en" }, values = { "Service callout to retrieve patient descriptive

```



---

```
information for the set of Corticon 'Patient' entities." }}
```

Service Call-out methods must be declared `public static`.

The system will recognize your Service Call-out method if the method signature takes one parameter and that parameter is an instance of `ICcDataObjectManager`.

```
public static void getPatientInfo(ICcDataObjectManager dataObjMgr) {
    // Get the set of "Patient" entities.
    Set<ICcDataObject> patients = dataObjMgr.getEntitiesByName("Patient");
    if (patients != null) {
        // Process each patient in the set.
        for (ICcDataObject patient : patients) {
            // Get the "id" attribute of the current patient.
            Long id = (Long) patient.getAttributeValue("id");

            if (id != null) {
                // Get the simulated information for this patient.
                PatientData pd = md.getPatientData(id.intValue());

                if (pd != null) {
                    // Set patient information as entity attributes.
                    patient.setAttributeValue("firstName", pd.getFirstName());
                    patient.setAttributeValue("lastName", pd.getLastName());
                }
            }
        }
    }
}

/**
 * Service callout to retrieve history of medical procedures for the
 * set of Corticon "Patient" entities.
 *
 * This callout demonstrates how to create new Corticon entities and
 * associate them with existing Corticon entities.
 */
@Description(lang = { "en" }, values = { "Service callout to retrieve history of medical
procedures for the set of Corticon 'Patient' entities." })
public static void getProcedureHistory(ICcDataObjectManager dataObjMgr) {
    // Get the set of "Patient" entities.
    Set<ICcDataObject> patients = dataObjMgr.getEntitiesByName("Patient");
    if (patients != null) {
        // Process each patient in the set.
        for (ICcDataObject patient : patients) {
            // Get the "id" attribute of the current patient.
            Long id = (Long) patient.getAttributeValue("id");

            if (id != null) {
                // Get the simulated information for this patient.
                PatientData pd = md.getPatientData(id.intValue());

                // Get the medical procedure history for this patient.
                ArrayList<ProcedureData> td = pd.getProcedureRecords();

                // Add procedures to the patient entity
                for (ProcedureData r : td) {
                    // Create a new "Procedure" entity.
                    ICcDataObject p = dataObjMgr.createEntity("Procedure");

                    // Set attributes on this entity.
                    p.setAttributeValue("code", r.getCode());
                    p.setAttributeValue("description", r.getDescription());
                    p.setAttributeValue("date", r.getDate());

                    // Associate it with the current patient.
                    patient.addAssociation("procedure", p);
                }
            }
        }
    }
}
```

```

    }
    }
}

/**
 * Service callout to save the approval state for each medical procedure
 * for a set of Corticon "Patient" entities.
 *
 * This callout demonstrates how to iterate over a set of entities and
 * associations to get the value of attributes.
 */
@Description(lang = { "en" }, values = { "Service callout to save the approval state
for each medical procedure for a set of Corticon 'Patient' entities." })
public static void saveProcedureApproval(ICcDataObjectManager dataObjMgr) {
    // Get the set of "Patient" entities.
    Set<ICcDataObject> patients = dataObjMgr.getEntitiesByName("Patient");
    if (patients != null) {
        // Process each patient in the set.
        for (ICcDataObject patient : patients) {
            // Get the "id" attribute of the current patient.
            Long id = (Long) patient.getAttributeValue("id");

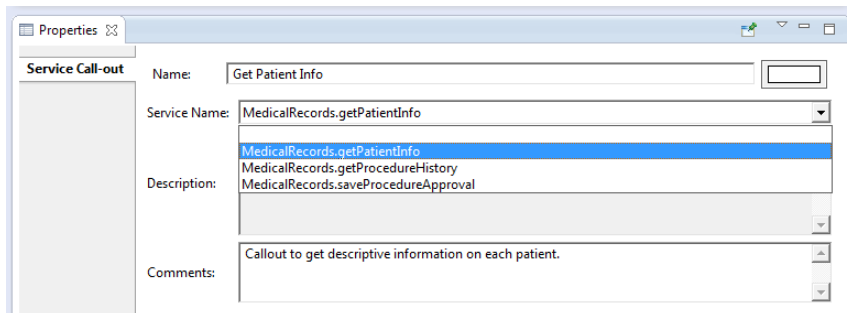
            // Process each "Procedure" association on the patient.
            Set<ICcDataObject> procedures = patient.getAssociations("procedure");
            for (ICcDataObject procedure: procedures) {

                // Get the value of the "approved" and "code" attributes on the procedure.
                Boolean approved = (Boolean) procedure.getAttributeValue("approved");
                String code = (String) procedure.getAttributeValue("code");

                // Update the procedure record.
                md.setProcedureApproval(id.intValue(), code, approved);
            }
        }
    }
}

```

Recognized classes and methods are displayed in the Ruleflow Properties View/Service Name drop-down list when a Service Call-out object is on a Ruleflow canvas:



The Service Call-out API provides your extension class complete access to the fact pool, allowing you to:

- Find entities in several ways
- Read and update entity attributes
- Create and remove entity instances
- Create and remove associations

Refer to Service Callout Java sample project and the *API Javadocs* for more information.

For details, see the following topics:

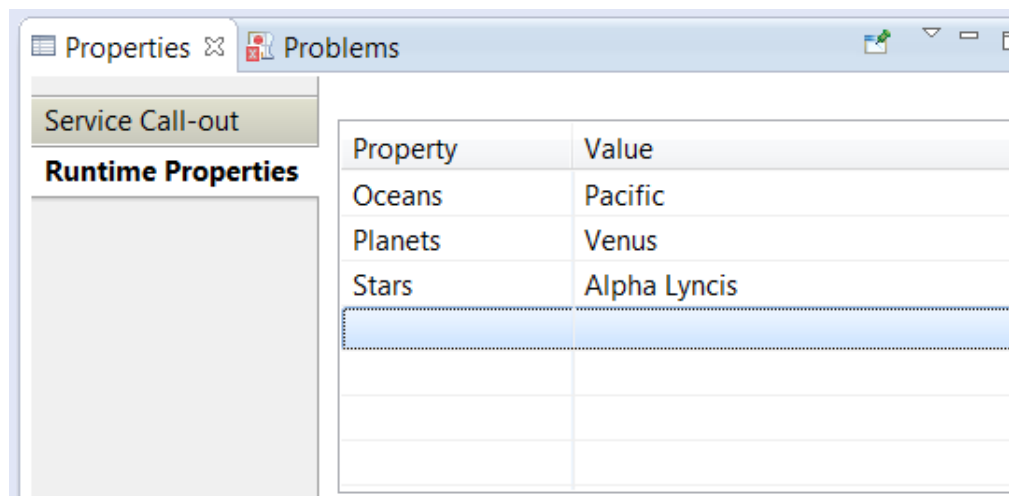
- [Specifying properties on a service callout instance](#)
- [Access to Vocabulary Metadata](#)
- [Using Advanced Data Callouts \(ADC\)](#)

## Specifying properties on a service callout instance

You can specify properties on a Service Callout that can be set per instance . That means that a SCO that retrieves data from a web service could use multiple instances of it in a Ruleflow where each instance has different parameters. The nature of the parameters is unrestricted; they are simple name/value pairs that a SCO can interpret as needed.

### Overview of SCO parameters

When a SCO is added to a Ruleflow canvas, its **Properties (View) > Runtime Properties** let you set name/value parameter pairs on this SCO instance. These name/value pairs will be passed to the SCO when the SCO is executed. For example:



To enable this functionality, the SCO's method must need to accept a `java.util.Properties` object in its method signature:

```
public static void processCreditReport(ICcDataObjectManager aDataObjectManager,
    Properties apropServiceCalloutProperties)
```

If the method does not accept a `Properties` object (as is the case for SCO's created before 5.6.1), the original method will still be called, providing both backward compatibility as well as an alternative approach to using parameters in SCOs.

```
public static void processCreditReport(ICcDataObjectManager aDataObjectManager)
```

If the SCO has implemented both methods, the method with the `Properties` object will be called during execution. If this method does not exist, then the alternative applies.

## Selecting the Runtime Properties for a SCO

### Defined Property Names and Values

Often you will want to constrain the Property Names and their respective Values to ensure that only valid combinations are selected by the user from a drop-down list box. The SCO must implement a specific Interface and the following methods for the Ruleflow to list the possible Property Names and their respective Values.

Interface:

```
com.corticon.services.extensions.ICcPropertyProvider
```

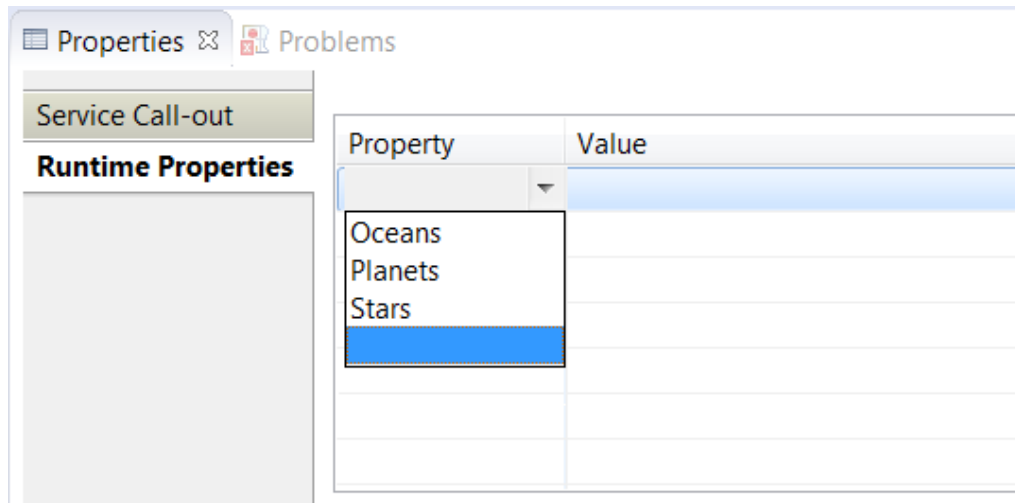
Static Methods:

```
public List<String> getPropertyNamesOptions() throws Exception;
```

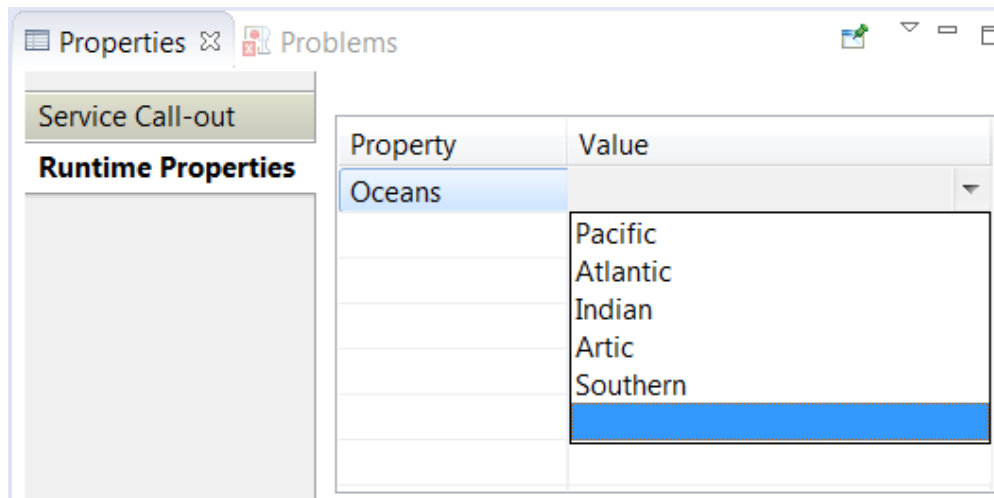
```
public List<String> getPropertyValueOptions(String astrPropertyName) throws Exception;
```

Example:

The user drops down the list and then chooses a property name:



The Ruleflow calls back to the SCO to get the possible Values for that Property name, and then lists the values in a drop-downlist where the user selects the value:



**Note:** This technique does not allow additional name/value pairs to be entered.

## No defined Property Names and Values

Undefined Property Names and Values occur when:

- The SCO does not implement the `ICcPropertyProvider` interface
- The interface and methods are implemented, but the methods return a null or empty list

Under these conditions, the Property Name and Property Value cells in the **Properties (View) > Runtime Properties** are TextBoxes where a can type names and values on as many lines as they want:

The screenshot shows a window titled 'Properties' with a sub-tab 'Runtime Properties'. On the left, there are buttons for 'Service Call-out' and 'Runtime Properties'. The main area contains a table with two columns: 'Property' and 'Value'. Both columns are empty text boxes, indicating that no values are defined for these properties.

There are no values defined for a free-form property name so a value must be typed in:

The screenshot shows the same window as before, but now the 'Property' cell contains the text 'TypeAnything'. The 'Value' cell remains empty, indicating that a value must be typed in when no predefined values are available.

**Note: Property Name and Value lists work independently** - While `getPropertyNamesOptions()` might return a `List<String>` with values so that the cell on the current **Property** line offers a dropdown list, the selected property might find that its `getPropertyValueOptions(..)` returns a null or empty list. In that case, the **Value** cell is provided as a text box for your free-form entry. However, each property name and value pair must have non-blank entries to complete valid service callout runtime properties.

# Access to Vocabulary Metadata

## Access to Vocabulary Metadata through the `ICcDataObjectManager`

Extended operators have access to the `ICcDataObjectManager`. This class has long been available to Service Callouts and provides access to metadata such as the Corticon Vocabulary and the entities being processed. To be passed an instance of `ICcDataObjectManager`, the extension class must define method signatures which take `ICcDataObjectManager` as a parameter. See the Corticon JavaDoc for more details.

The method:

```
ICcDataObjectManager.getVocabularyMetadata()
```

Return type:

```
com.corticon.services.metadata.IVocabularyMetadata
```

For details about the `IVocabularyMetadata` object, see the topic *"Accessing the Vocabulary metadata of a Decision Service" in the Corticon Rule Modeling Guide*.

## Using Advanced Data Callouts (ADC)

Corticon's Advanced Data Callout (ADC) provides an alternative to Corticon's Enterprise Data Connector (EDC) for accessing database data. It provides greater control over the query, and insert statements that are used. This is beneficial when you need finer control for performance or need to retrieve large amounts of data. Batch processing applications are a good example of where ADC is effective. With ADC you define a mapping of your vocabulary to a database, define queries, and control when queries are performed to retrieve data. With ADC you can quickly retrieve large amounts of data.

By contrast, with EDC you define a mapping of your vocabulary to a database and rely on Corticon to retrieve data as needed. EDC makes data access very simple and is a great option when small amounts of data are needed or performance is not paramount. EDC performs lazy loading of data in that it only loads data as needed. This is particularly evident when retrieving data for associations. When large amounts of data need to be retrieved or performance is paramount this can be inefficient.

### Comparing ADC and EDC

Consider a database with tables and relations such as:

- 'Person' Table has 1,000 rows
- 'Job' Table has 10,000 rows (10 associated Job records for each Person record)
- 'Duty' Table has 100,000 rows (10 associated Duty records for each Job)

In EDC, you create one SQL Statement to retrieve all Person Entities into Corticon, and then one SQL Statement is created for each Person to load that Person's associated Job Entities – another 1,000 SQL executions. Then, with one SQL Statement for each Job to load that Job's associated Duty Entities adds another 10,000 SQL executions to the process. A lot of time is dedicated to retrieving data. With ADC this can be reduced to three queries; one for each table. This can greatly reduce the time spent accessing data.

### How ADC Works

ADC functions as a service callout -- it accesses data as a step in a Ruleflow. To use ADC, do the following:

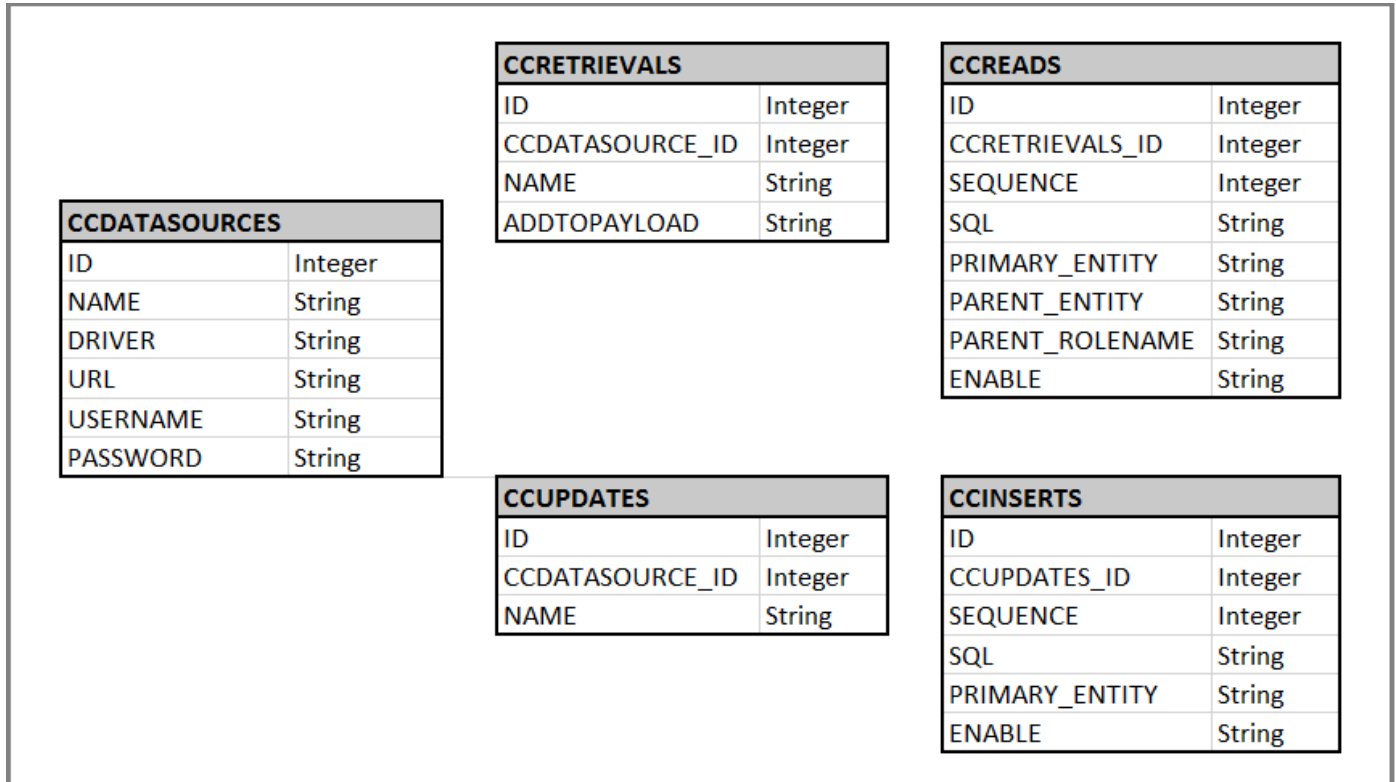
1. Map your vocabulary to a database. This is done in Corticon vocabulary editor the same as is done for EDC. The mapping tells ADC how to construct entities and associations for data retrieved from the database and how to save data when storing to the database.
2. Define parameterized SQL statements for the queries and to be performed. You have full control over these queries. They are parameterized such that substitutions can be performed at runtime. To make these statements easy to manage, they are also stored in a database. This can be the same or separate database from the data to be queried.
3. Add the ADC callout to a Ruleflow. This is done in the Corticon Ruleflow editor. When you add ADC to a Ruleflow you will need to configure it to identify the query or insert operation to be performed by selecting one of the SQL statements you have defined. To make this easier, you can give the SQL statements logical names.

When all steps are completed you are ready to deploy your Ruleflow or test it in the Corticon tester. When ADC runs it will perform substitutions into the statement and use it to access data. For queries, ADC will construct entities, set attributes, and define associations using the vocabulary mapping. For inserts, ADC will use the mapping data for storing to the database. You can use multiple instance of ADC in a Ruleflow. A typical use case would be to have an instance at the start of a Ruleflow to retrieve data and one later in the Ruleflow to save data.

## Configuration of the ADC Database Schema

The following diagram illustrates the Database Schema that ADC uses. SQL Scripts help you generate these Tables inside your Database. However, the Table and Column names cannot be changed because the ADC is hardcoded to look for specific names.

**Figure 1: Database Schema for Corticon's Advanced Data Access Service Callout**



The core operations that an ADC can perform are: retrieving data (using the CCRETRIEVALS Table) and updating data (using the CCUPDATES Table). Each one of these CCRETRIEVALS or CCUPDATES row instances can use a different DataSource (using the CCDATASOURCES Table). Because there is a Foreign Key from CCRETRIEVALS and CCUPDATES back to CCDATASOURCES (CCDATASOURCES\_ID), each CCRETRIEVALS and CCUPDATES can only connect to one CCDATASOURCES.

**Table 1: CCDATASOURCES Table**

Column Name : DataType	Note
ID : Integer	The <b>Primary Key</b> for the Table which then gets propagated down to each CCRETRIEVALS or CCUPDATES record.
NAME : String	A logical name that you want to associated with this DataSource. This name can be used in to lookup an already established Connection through a JNDI call. If the JNDI Lookup fails, then the new Connection is stored in memory, using this name, to be reused by other execution Threads.

Column Name : DataType	Note																														
DRIVER : String	<p>The Driver Class that will be used to make the Connection to the Database. This value has two different meanings 1) Progress' DataDirect Driver ID or 2) Third Party Driver Class. It is strongly recommended that the customer sets the DataDirect Driver ID, instead of a Third Party Class. Supported Databases with the associated DataDirect ID are:</p> <table> <thead> <tr> <th>Database Id</th><th>Database Name</th></tr> </thead> <tbody> <tr><td>com.corticon.database.id.Oracle12c</td><td>Oracle Database 12c</td></tr> <tr><td>com.corticon.database.id.Oracle10g</td><td>Oracle Database 10g</td></tr> <tr><td>com.corticon.database.id.Oracle</td><td>Oracle Database 11g</td></tr> <tr><td>com.corticon.database.id.DB210.5</td><td>IBM DB2 10.5</td></tr> <tr><td>com.corticon.database.id.DB2</td><td>IBM DB2 9.5</td></tr> <tr><td>com.corticon.database.id.MsSql</td><td>Microsoft SQL Server 2008</td></tr> <tr><td>com.corticon.database.id.MsSql2012</td><td>Microsoft SQL Server 2012</td></tr> <tr><td>com.corticon.database.id.MsSql2014</td><td>Microsoft SQL Server 2014</td></tr> <tr><td>com.corticon.database.id.MySQL</td><td>MySQL 5.6 Database</td></tr> <tr><td>com.corticon.database.id.PostgreSQL</td><td>PostgreSQL 9.4 Database</td></tr> <tr><td>com.corticon.database.id.OE11.5</td><td>Progress OpenEdge 11.5</td></tr> <tr><td>com.corticon.database.id.OE11.4</td><td>Progress OpenEdge 11.4</td></tr> <tr><td>com.corticon.database.id.OE11.3</td><td>Progress OpenEdge 11.3</td></tr> <tr><td>com.corticon.database.id.OE10.2</td><td>Progress OpenEdge 10.2</td></tr> </tbody> </table>	Database Id	Database Name	com.corticon.database.id.Oracle12c	Oracle Database 12c	com.corticon.database.id.Oracle10g	Oracle Database 10g	com.corticon.database.id.Oracle	Oracle Database 11g	com.corticon.database.id.DB210.5	IBM DB2 10.5	com.corticon.database.id.DB2	IBM DB2 9.5	com.corticon.database.id.MsSql	Microsoft SQL Server 2008	com.corticon.database.id.MsSql2012	Microsoft SQL Server 2012	com.corticon.database.id.MsSql2014	Microsoft SQL Server 2014	com.corticon.database.id.MySQL	MySQL 5.6 Database	com.corticon.database.id.PostgreSQL	PostgreSQL 9.4 Database	com.corticon.database.id.OE11.5	Progress OpenEdge 11.5	com.corticon.database.id.OE11.4	Progress OpenEdge 11.4	com.corticon.database.id.OE11.3	Progress OpenEdge 11.3	com.corticon.database.id.OE10.2	Progress OpenEdge 10.2
Database Id	Database Name																														
com.corticon.database.id.Oracle12c	Oracle Database 12c																														
com.corticon.database.id.Oracle10g	Oracle Database 10g																														
com.corticon.database.id.Oracle	Oracle Database 11g																														
com.corticon.database.id.DB210.5	IBM DB2 10.5																														
com.corticon.database.id.DB2	IBM DB2 9.5																														
com.corticon.database.id.MsSql	Microsoft SQL Server 2008																														
com.corticon.database.id.MsSql2012	Microsoft SQL Server 2012																														
com.corticon.database.id.MsSql2014	Microsoft SQL Server 2014																														
com.corticon.database.id.MySQL	MySQL 5.6 Database																														
com.corticon.database.id.PostgreSQL	PostgreSQL 9.4 Database																														
com.corticon.database.id.OE11.5	Progress OpenEdge 11.5																														
com.corticon.database.id.OE11.4	Progress OpenEdge 11.4																														
com.corticon.database.id.OE11.3	Progress OpenEdge 11.3																														
com.corticon.database.id.OE10.2	Progress OpenEdge 10.2																														
URL : String	URL to be used to connect to selected Database.																														
USERNAME : String	Username to authenticate on the selectetd Database.																														
PASSWORD : String	Password of the specified usernamme.																														

Table 2: CCRETRIEVALS Table

Column Name : DataType	Note
ID : Integer	The <b>Primary Key</b> for the Table which then gets propagated down to each CCREADS record.
CCDATASOURCE_ID : Integer	Foreign Key back to CCDATASOURCES . ID column. Each CCRETRIEVALS row can only have one associated CCDATASOURCES configuration.
NAME : String	A logical name that you want to associated with this CCRETRIEVALS. This is important because this is the name that will be specified inside of each ADC to determine which CCRETRIEVALS the ADC will perform.
ADDTOPAYLOAD : String (true or false)	<p>Depending on your Use Case, you may not want the huge amounts of data retrieved from the ADC to be added to the Payload. Depending on how much data is retrieved, adding this data to an XML or JSONObject Payload could be costly. We parameterized this option per CCRETRIEVALS because maybe some retrieval data you want in the returned Payload and some you don't.</p> <p>If value is null, or any value other than 'true', the default is 'false'.</p>



**Table 3: CCUPDATES Table**

This Table is very similar to the CCRETRIEVALS Table.

Column Name : DataType	Note
ID : Integer	The <b>Primary Key</b> for the Table which then gets propagated down to each CCINSERTS record.
CCDATASOURCE_ID : Integer	Foreign Key back to CCDATASOURCES . ID column. Each CCUPDATES row can only have one associated CCDATASOURCES configuration.
NAME : String	A logical name that you want to associated with this CCUPDATES. Just like the CcRetrievalsName in the Ruleflow Properties Section, there is a CcUpdatesName that will be used to specify which CCUPDATES Name the ADC will perform.

**Table 4: CCREADS Table**

The CCREADS and CCINSERTS Tables are the key Tables for ADC. These Tables contain the most pertinent information for the ACA SCO to perform its duties.

Column Name : DataType	Note
ID : Integer	The <b>Primary Key</b> for the Table.
CCRETRIEVALS_ID : Integer (required)	Foreign Key back to CCRETRIEVALS . ID column. There can be many CCREADS associated with a CCRETRIEVALS record.
SEQUENCE : Integer (required)	Because there can be multiple CCREADS associated with a CCRETRIEVALS record, we need to know what order to execute the CCREADS. The ADC will read in all the CCREADS into memory, and then sort them based on the value for SEQUENCE. Ideally, the values will be sequential, but since there is not Database Constraint on this Column, the values don't have to be sequential (1,4,6,7) or even unique. However, if the numbers are not unique (1,4,4,7), the CCREADS may fire in different orders per execution.
SQL : String (required)	An SQL Statement that is technically a template that will be used for this CCREADS operation. The SQL Statement is very flexible in that it can incorporate complex WHERE clause using Corticon's Variable Substitution. Variable Substitution allows you to specify Entity.Attribute keys in the SQL, which will be replaced with corresponding values based on what is currently in the working memory of the execution. This is explained in more detail later in this document.

Column Name : DataType	Note
PRIMARY_ENTITY : String (required)	<p>The results from the SQL Statement. This needs to be converted to map to Corticon Entities. This field is to tell the ADC which Entity to map the results.</p> <p>The ADC will not automatically “create” a new instance of the Entity in memory. First, it will determine if that Entity is already in memory, and if it is not already in memory, a new Entity instance of type (PRIMARY_ENTITY) will be created, using <code>ICcDataObjectManager.createEntity(&lt;PRIMARY_ENTITY&gt;)</code>. Then the Column Values for that Row will be added into that new instance of the Entity.</p> <p>Things get more complicated when there are already instances of the &lt;PRIMARY_ENTITY&gt; as the Entity that is in working memory. To prevent duplication of Entity instances, we want to first check to see if that Entity instance is already in memory. If we determine that the instance already exists, then we will use that instance, and then merge the Column Values into that Entity instance.</p> <p>An example of this is covered in section “Root Entity with no Variable Substitution : Example 2” below.</p>
PARENT_ENTITY : String and PARENT_ROLENAME : String (optional)	<p>These values are needed to create an Association between the Parent Entity (PARENT_ENTITY) to the Target Entity (PRIMARY_ENTITY) through Association Role Name (PARENT_ROLENAME).</p> <p>This is the most complicated part of ADA SCO.</p> <p>The Association’s Role Name: Join Expression is critical to the mapping of Associations between the PARENT_ENTITY and the PRIMARY_ENTITY.</p> <p>The ADC parses the Join Expression to determine which Attributes in the Parent Entity need to match which Attributes in the Primary Entity. For each Primary Entity retrieved, an extensive algorithm is used to match these values between two different Entities. If there is a match, the Primary Entity is added to the Parent Entity’s Association Role Name.</p>
ENABLE : String (true or false)	<p>For testing purposes, you may want to test some CCREADS out of all the ones associated with the CCRETRIEVALS. You can add all your CCREADS information and then incrementally expand the retrieval, while testing each step.</p> <p>If value is null, or any value other than ‘false’, the default is ‘true’.</p>

**Table 5: CCINSERTS Table**

The CCREADS and CCINSERTS Tables are the key Tables for ADC. These Tables contain the most pertinent information for the ADC to perform its duties.

Column Name : DataType	Note
ID : Integer	The <b>Primary Key</b> for the Table
CCUPDATES_ID : Integer	Foreign Key back to CCUPDATES . ID column. Because of this, there can be many CCINSERTS associated with a CCUPDATES record.

Column Name : DataType	Note
SEQUENCE : Integer	Because there can be multiple <code>CCINSERTS</code> associated with a <code>CCRETRIEVALS</code> record, we need to know what order to execute the <code>CCINSERTS</code> . The ADC will read in all the <code>CCINSERTS</code> into memory, and then sort them based on the value for <code>SEQUENCE</code> . Ideally, the values will be sequential, but since there is not Database Constraint on this Column, the values don't have to be sequential (1,4,6,7) or even unique. However, if the numbers are not unique (1,4,4,7), the <code>CCINSERTS</code> may fire in different orders per execution.
SQL : String	SQL Statement used as a template for this <code>CCINSERTS</code> operation. The SQL Statement is very flexible and uses Variable Substitution to add Primary Entity values directly into the SQL.  Since the structure of an <code>INSERT</code> statement is drastically different than a <code>SELECT</code> statement, Variable Substitution will not aggregate all values to create one SQL statement, but instead, it will use the SQL as a template to create a SQL statement for each Primary Entity instance.
PRIMARY_ENTITY : String	The Entity name that will be used to look up all instances of this Entity type from working memory in which Variable Substitution will be applied to the SQL statement to create an individual <code>INSERT</code> statement per Entity instance.
ENABLE : String (true or false)	For testing purposes, you may want to test some <code>CCINSERTS</code> out of all the ones associated with the <code>CCUPDATES</code> . You can add all your <code>CCINSERTS</code> information and then incrementally expand the updates, while testing each step. If value is null, or any value other than 'false', the default is 'true'.

## Configuration of the ADC Database Metadata

As described in the schema, `CCDATASOURCES` are defined in the database that is retrieved by the ADA SCO as Metadata stored in the `CcADASco.properties` file in the Server's `[CORTICON_WORK]` directory.

Storing Configuration Metadata in a database provides flexibility and security. The Metadata could be saved into the same database as core customer data, or different database. The ADC looks up metadata to perform its current operation.

**Table 6: CcADASco.properties File**

This file requires a value for each of the following properties:

Property Name	Property Values
<code>corticon.servicecallout.lookup.name</code>	The logical name for this Connection's DataSource. This name first looks up a DataSource through a <code>JNDI InitialContext</code> . If no DataSource has been registered, then the following properties establish a Connection DataSource. Then, this DataSource is stored in memory, using its name so that other execution Threads can use the same DataSource.
<code>corticon.servicecallout.lookup.drive</code>	As described for <code>CCDATASOURCES</code> , this value can be the DataDirect Driver ID or the third party driver name.
<code>corticon.servicecallout.lookup.url</code>	Standard connection URL to the database.

Property Name	Property Values
<code>corticon.servicecallout.lookup.username</code>	The user to be authenticated on the database. You can choose to encrypt the username.
<code>corticon.servicecallout.lookup.password</code>	The user's password. You can choose to encrypt

Example:

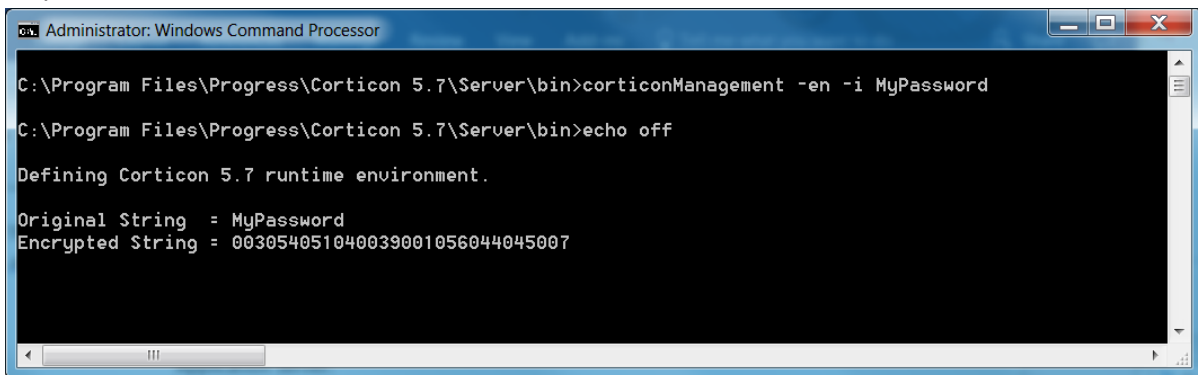
```
corticon.servicecallout.lookup.name=Corticon
corticon.servicecallout.lookup.driver=com.corticon.database.id.Oracle
corticon.servicecallout.lookup.url=jdbc:progress:oracle://<IP>:1521;databaseName=corticon
corticon.servicecallout.lookup.username=010014055008007043001000
corticon.servicecallout.lookup.password=030046016058035029061039110081
```

If there is a problem retrieving the metadata from the Database, Exceptions are logged into the Corticon Log file. If there is a problem during Runtime, a `CcRuleMessages -> Violation` message is posted and added to the Response.

### Encrypting database credentials defined in the `CcADASco.properties` File

Corticon uses a proprietary encryption algorithm which is the recommended technique for encrypting credentials, as follows:

1. In a Corticon Server installation, open a Command Window at `[CORTICON_HOME]\Server\bin\`.
2. Type `corticonManagement -en -i username`. An encrypted String for the `username` you entered is output.
3. Copy the encrypted String to the appropriate `CcADASco.properties` file, and enter it as the value for `corticon.servicecallout.lookup.username`.
4. Type `corticonManagement -en -i password`. An encrypted String for the `password` you entered is output, as shown:



```
Administrator: Windows Command Processor

C:\Program Files\Progress\Corticon 5.7\Server\bin>corticonManagement -en -i MyPassword

C:\Program Files\Progress\Corticon 5.7\Server\bin>echo off

Defining Corticon 5.7 runtime environment.

Original String = MyPassword
Encrypted String = 003054051040039001056044045007
```

5. Copy the encrypted String to the appropriate `CcADASco.properties` file, and enter it as the value for `corticon.servicecallout.lookup.password`.

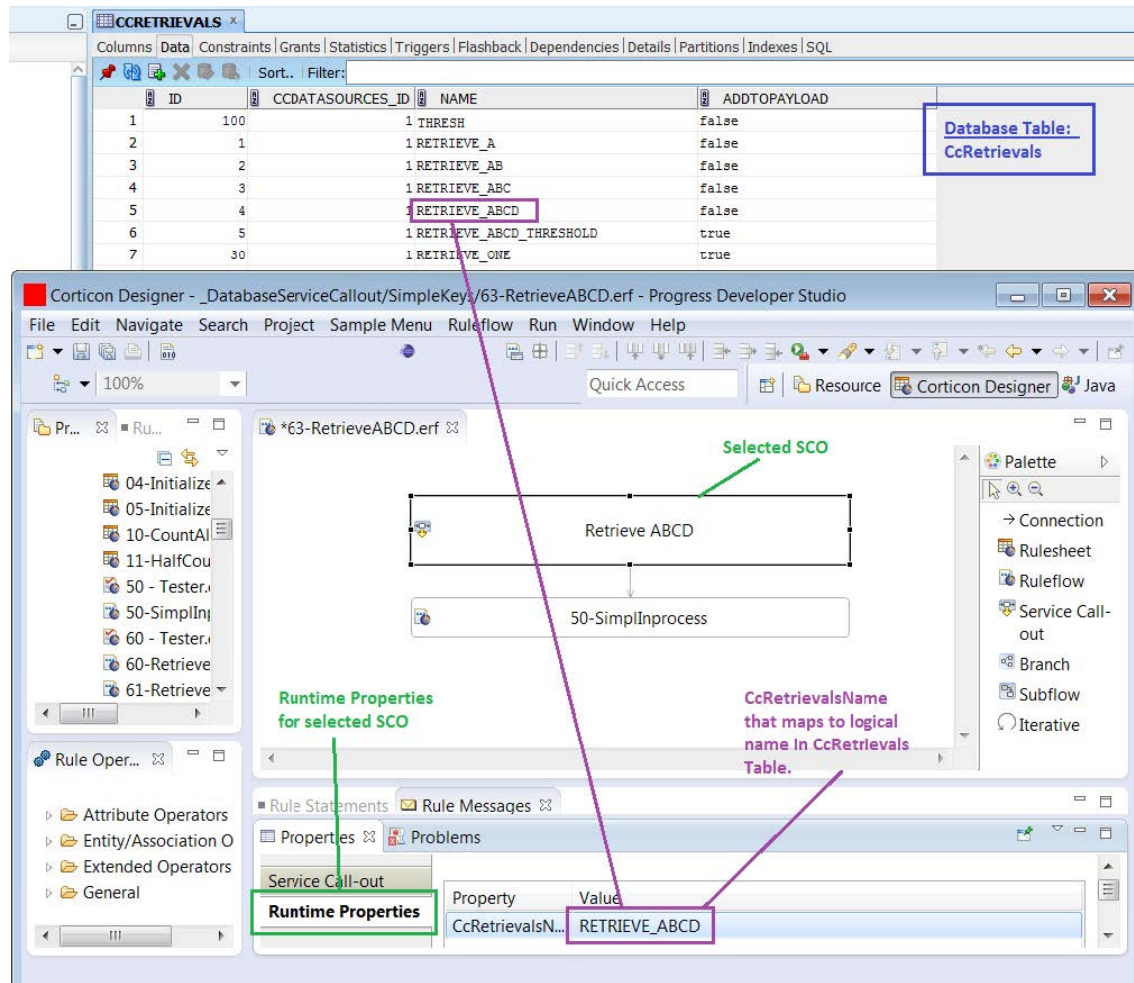
### Associating a `CcRetrievalsName` Or `CcUpdatesName` to a SCO

An ADC **Runtime Properties** on a Ruleflow canvas expects you to specify the property as either:

- **CcRetrievalsName** for retrievals of the name value specified
- **CcUpdatesName** for updates of the name value specified

For example:

Once you add the SCO to the Ruleflow, you can access to the **Properties -> Runtime Properties** section (shown in green). The value for **CcRetrievalsName** here is **RETRIEVE\_ABCD**, the NAME of a CCRETRIEVALS Record in the Database (shown in purple):



## Efficient retrieval of large amounts of data

The SCO uses the metadata in CCDATASOURCES, CCRETRIEVALS, and CCREADS tables to query Database Tables so that they retrieve large amounts of data to be added to the working memory of the execution.

```
SCO Service Name = CcDatabaseServiceCallout.retrieveEntitiesFromDatabase
```

The core retrieval of data is based on what metadata is in the CCREADS table. This section describes how the SEQUENCE, SQL, PRIMARY\_NAME, PARENT\_NAME, and PARENT\_ROLENAME are used in CCREADS records to retrieve large amounts of data.

There are four types of retrievals that can be configured:

1. Root Entity with no Variable Substitution
2. Associated Entities with no Variable Substitution
3. Root Entity with Variable Substitution
4. Associated Entities with Variable Substitution

## Root Entity with no Variable Substitution

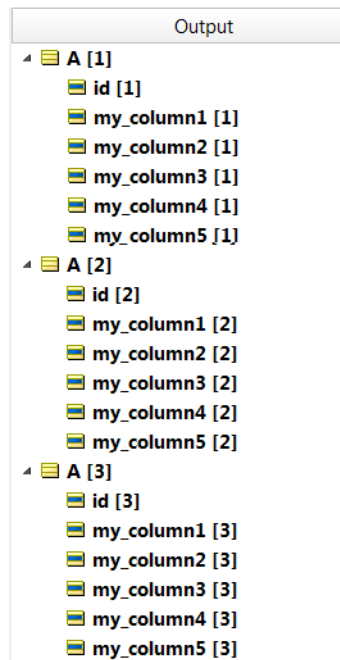
This type of retrieval utilizes the `SQL` and `PRIMARY_NAME` values inside the Database Row. The `SQL` retrieves rows from a Database Table and the results are mapped to what is defined in the `PRIMARY_NAME`.

### Example 1:

```
SQL = Select * from M_A
PRIMARY_NAME = A
```

The SQL Statement retrieves all records from the `M_A` Table. A new Vocabulary `A` Entity instance (based on `PRIMARY_NAME`) is created for each row, and the contents of the row are added to the `A` instance.

If there are three `M_A` records in the Database, the result from this retrieval is three `A` instances added to the working memory of the execution, as illustrated:



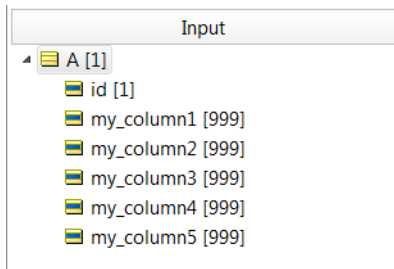
**Note:** All Entities and Attributes are in BOLD face (inside the Tester) -- that indicates that these Entities and Attributes were not part of the original payload.

### Example 2:

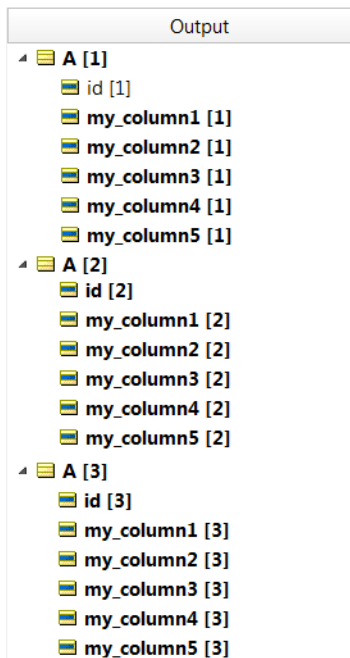
As stated in the `CCREADS > PRIMARY_ENTITY` description, the ADC will not unconditionally create a new `PRIMARY_ENTITY` instance for every row that is retrieved. The SCO first checks to see if that instance is already in memory by comparing the working memory `PRIMARY_ENTITY`'s Identity Attribute(s) and comparing those values with the corresponding values from the retrieved record. If there is a match, then the working memory instance is used, and all the Data retrieved from the Database will override what is currently in the working memory instance.

```
SQL = Select * from M_A
PRIMARY_NAME = A
```

The payload passed in, which gets added to working memory, is:



The output is:



A new instance A [2] and A [3] were created with the values from the Database.

However, A [1] -> id = 1 was already in memory along with default values of 999 for each of the column values. Based on Entity A's Identity (defined in the Vocabulary), the system determined to reuse instance A [1] -> id = 1, and then merge the values from the Database.

There is hint here: A.id is not **BOLD** in the Output column of the tester. That means this Entity.Attribute didn't change in Rule Processing. If it didn't change (or get added), then it was part of the payload. This tells you that the passed in A [1] Entity was reused. Also, if the system didn't look up for existing Entity A.id=1 instances, then a second A would be created with id=1.

## Associated Entities with no Variable Substitution

This type of retrieval utilizes the SQL, PRIMARY\_NAME, PARENT\_NAME, and PARENT\_ROLENAME values in the Database Row. This retrieval retrieves data that will be added in Entity instances -- but further, those instances will be added to an Entity's Association through an Association's Rolename.

### Example 1:

```
SQL = Select * from M_B
PRIMARY_NAME = B
PARENT_ENTITY = A
PARENT_ROLENAME = toB
```

As described in the previous section, “Root Entity with no Variable Substitution”, the `SQL` and `PRIMARY_NAME` work together to create new instances or lookup an existing Entity instance in working memory, based on what is retrieved from the Database.

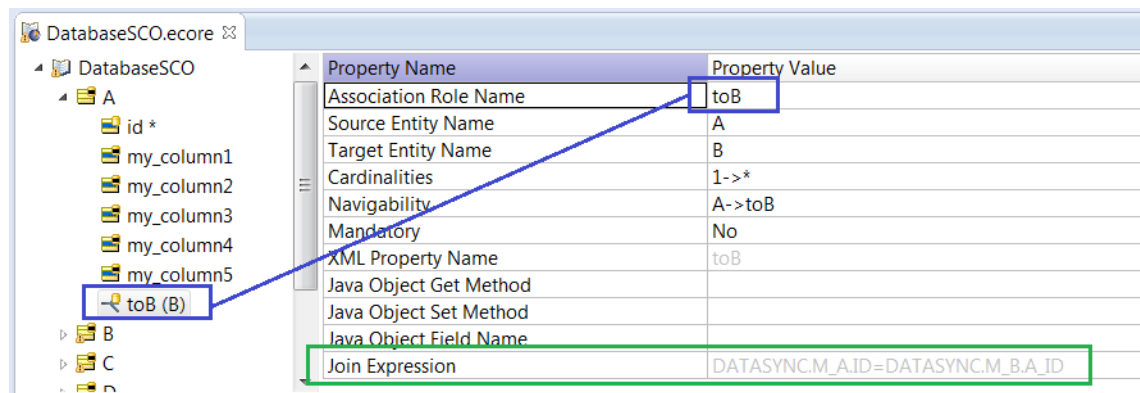
However, we don't want to create a new `PRIMARY_ENTITY` instance if we know it won't be associated to a `PARENT_ENTITY`. If the retrieved Database record won't be added to an Association to any of the `PARENT_ENTITY`, `PRIMARY_ENTITY` is valid and should be added to an `A.toB` Association, then it will be added.

The way that the ADC determines which `B` Entity is added to an individual `A.toB` Association is through the Vocabulary's Association Join Statement. The Join Statements defines which Columns from each Table need to match for these two Entities to be associated with each other.

Using the existing example, the Association Join Statement defined for `A.toB` is:

```
M_A.ID = M_B.A_ID
```

**Note:** To avoid confusion, the Schema Name (which precedes the Table Name) has been removed, . The actual expression could be `DATASYNC.M_A.ID = DATASYNC.M_B.A_ID`, with `DATASYNC` the Schema Name associated with Tables `M_A` and `M_B`).



The record `M_B` is retrieved from the Database to first check whether the `B` record will be added to an Association, and then – if yes – create a new `B` record or lookup from working memory and reuse.

Assuming that `M_B's -> A_ID = 1`, the Join Expression is applied to find the `A` instance that has `ID = 1`.

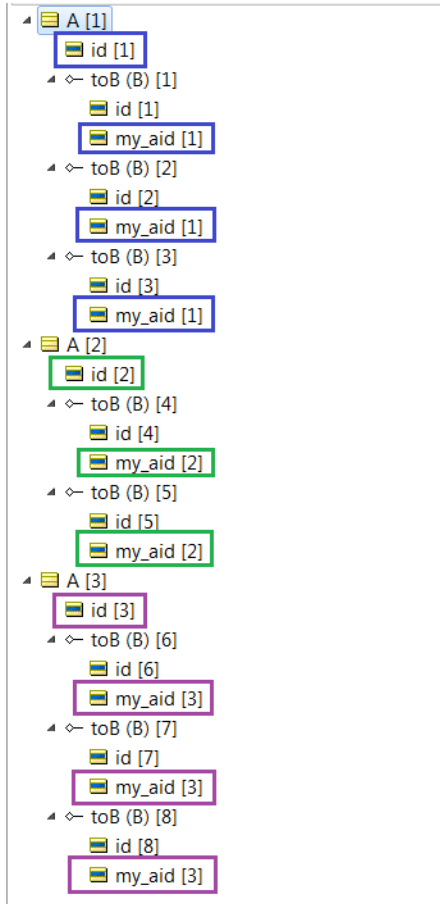
- If an `A` is found with `ID=1`, then the `B` instance will be added to that `A's toB` Association.
- If an `A` is not found with `ID=1`, then the `B` instance is ignored. It won't be added to working memory and it won't be associated with an `A` instance

Using the following data inside `M_A` and `M_B`...

M_A		M_B	
ID		ID	A_ID
1	1	1	1
2	2	2	1
3	3	3	1
	4	4	2
	5	5	2
	6	6	3
	7	7	3
	8	8	3



... the following Associations between A and B will be created:



**Note:** The B Entity has a `my_aid` Attribute that maps to Database Column `A_ID`. This demonstrates that the Vocabulary Attribute Name doesn't need to have the same name as the Column Name.

### Example 2:

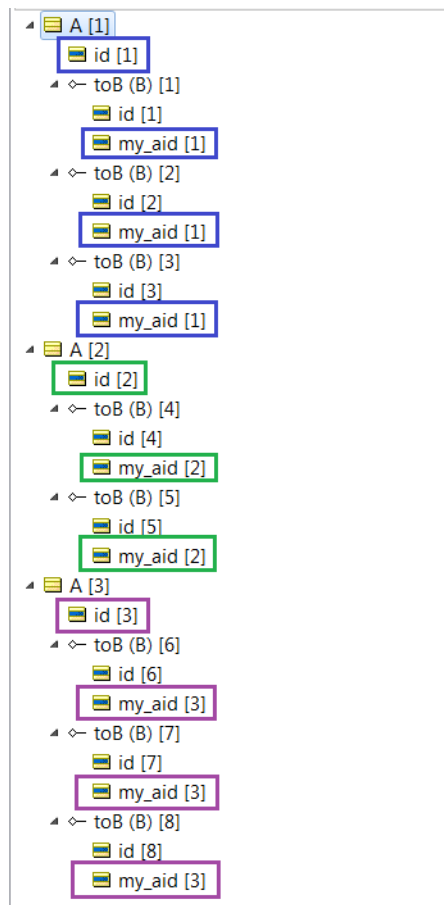
```
SQL = Select * from M_B
PRIMARY_NAME = B
PARENT_ENTITY = A
PARENT_ROLENAME = toB
```

This example is very similar to Example 1, but there is additional data in this example's `M_B` Table.

M_A		M_B	
ID		ID	A_ID
1		1	1
2		2	1
3		3	1
		4	2
		5	2
		6	3
		7	3
		8	3
		9	4
		10	4
		11	5
		12	5
		13	6

Because the SQL is `Select * from M_B`, all 13 `M_B` records will be retrieved from the Database. However, records with `ID = 9` to `13` don't have a valid `PARENT_ENTITY` to associate with. In this case, those `M_Bs` will not be created, will not be added to working memory, and will not be associated with any `PARENT_ENTITY` (A instance).

The results from Example 1 will be the same as Example 2.



In this particular case, it is advised to have a more defined SQL Statement so that not all the M\_B records are retrieved and then filtered inside the SCO. It is more efficient to create a dynamic WHERE clause using “Associated Entities with Variable Substitution”, discussed later in this section.

```
New SQL : Select * from M_B where a_id IN ({ A.id })
```

Using Variable Substitution, this new SQL would return only those M\_B records with a\_id values in the working set of A.id values. In the above case, the revised SQL would be dynamically changed to :

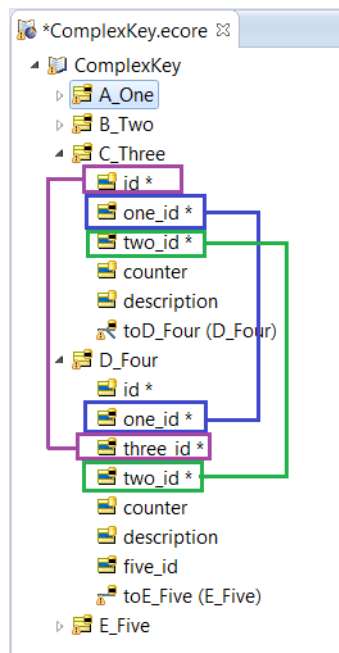
```
Select * from M_B where a_id IN (1, 2, 3)
```

### Example 3:

In the Example 1, the Join Expression linked two Entities based on only one Attribute. The next example shows that the ADC can handle more complex Join Expressions with multiple Attributes.

D\_Four has an Identity of (id, one\_id, two\_id, and three\_id).

As illustrated, D\_Four has Foreign Keys back to C\_Three Entity for Attributes one\_id, two\_id, and three\_id. This will make the Join Expression much more complex.



Join Expression for C\_Three.toD\_Four:

```
Q_THREE.ID = Q_FOUR.Q_THREE_ID,  
Q_THREE.Q_ONE_ID = Q_FOUR.Q_ONE_ID,  
Q_THREE.Q_TWO_ID = Q_FOUR.Q_TWO_ID
```

**Note:** The Schema name, which precedes the Table name has been edited off so that the Join Expression is easier to read.

```
SQL = Select * from Q_FOUR  
PRIMARY_NAME = D_Four  
PARENT_ENTITY = C_Threes  
PARENT_ROLENAME = toD_Four
```

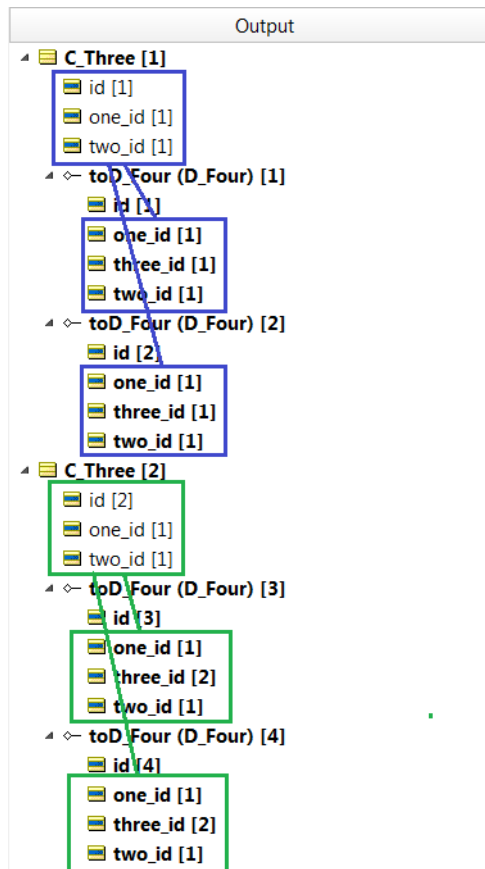
As described in the previous section, “Root Entity with no Variable Substitution”, the `SQL` and `PRIMARY_NAME` work together -- based on what is retrieved from the Database -- to create new instances, or lookup an existing Entity instance in working memory. Now, take the newly created or looked up `D_Four` instance, find the correct `C_Three` instance, and then add the `D_Four` to `C_Three.toD_Four` Association.

With multiple Attributes part of the Join Expression, the ADC needs a little more processing comparing each Attribute value in the new `D_Four` instance to the Attribute values in `C_Three`.

Using the following data inside `Q_Three` and `Q_Four`:

Q_Three			Q_Four			
ID	ONE_ID	TWO_ID	ID	ONE_ID	TWO_ID	THREE_ID
1	1	1	1	1	1	1
2	1	1	2	1	1	1
			3	1	1	2
			4	1	1	2

The following Associations between `C_Three` and `D_Four` will be created:



## Root Entity with Variable Substitution

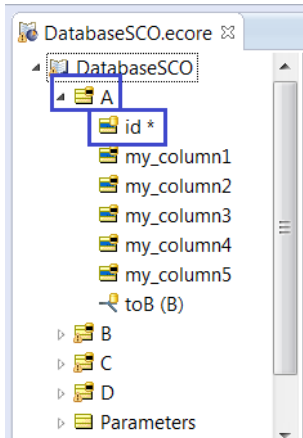
Like the “Root Entity without Variable Substitution, this type of retrieval utilizes the `SQL` and `PRIMARY_NAME` values inside the Database Row. As before, The `SQL` will be used to retrieve rows from a Database Table and the results will be mapped to what is defined in the `PRIMARY_NAME`. However, Variable Substitution allows the `SQL` to be more complex by dynamically adding variable values to the `WHERE` clause of the `SQL` statement.

**Example 1:**

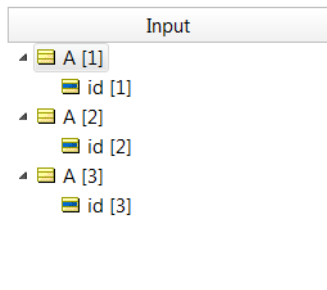
```
SQL = Select * from M_A where ID IN ( { A.id } )
PRIMARY_NAME = A
```

During execution of this CCREADS row, this SQL Statement will be processed as follows:

1. Read in the SQL and detect any Variable Substitutions have been added. In this case, there is a { A.id }. This A.id maps to the Vocabulary Entity=A and Attribute=id:



2. Query the working memory of the execution, and find all A Entities in memory, loop through each one, and then add each of A's id values to a List. Assuming there are three A Entities in working memory, the List of values associated with A.id would be 1, 2, 3.

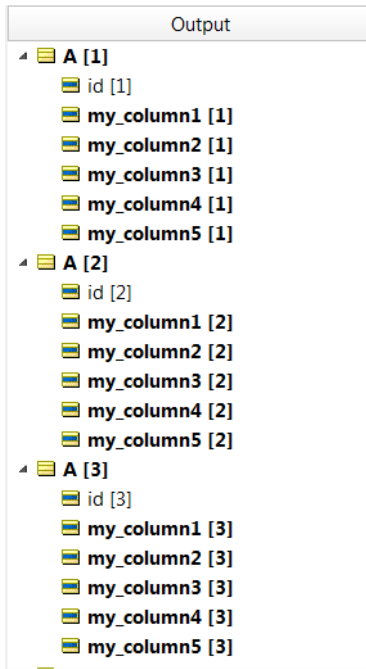


3. Use Variable Substitution to replace { A.id } with the list of values.

This produces a new SQL Statement: `Select * from M_A where ID IN ( 1, 2, 3 )`

This will be the actual SQL Statement that will be used by the CCREADS.

4. Because of the WHERE Clause limits where ID is in the set of (1, 2, 3), only three Rows will be returned from the Database. To prevent duplicate Entities in the working memory, the SCO will check to see if A:id=1, A:id=2, and A:id=3 are already in memory. For these Entities, they are reused and the data from the Database *enhances* each Entity by setting the my\_column1, my\_column2, my\_column3, my\_column4, and my\_column5 from the Database record, as illustrated:



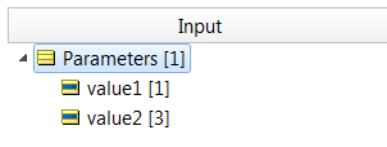
In this case, `A.id` for each `A` instance was passed into the execution through the Payload, which is why its values are not **BOLD** in the illustration. This demonstrates that the system found an existing `A` instance with `id=1`, and reused it instead of creating a new `A` instance.

### Example 2:

You are not limited to only one Variable Substitution per SQL. SQL can have any number of replacements, even from different Entities. In this example, Variable Substitution uses values from Entity `Parameters`.

```
SQL = Select * from A where ID > { Parameters.value1 } and ID < { Parameters.value2 }
PRIMARY_NAME = A
```

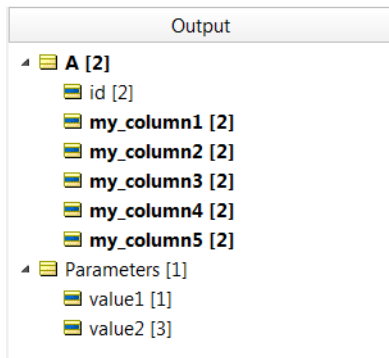
Use the following input...



... in the form:

```
{Parameters.value1} = 1
{Parameters.value2} = 3
Updated SQL = Select * from M_A where ID > 1 and ID < 3
```

Since there is only one `A` Entity that satisfy the `WHERE` clause, then only one `A` Entity is returned from the Database and added to the working memory, as shown:.



## Associated Entities with Variable Substitution

Like the “Associated Entities without Variable Substitution, this type of retrieval utilizes the `SQL`, `PRIMARY_NAME`, `PARENT_NAME`, and `PARENT_ROLENAME` values inside the Database Row. As before, the `SQL` will be used to retrieve rows from a Database Table and the results will be mapped to what is defined in the `PRIMARY_NAME`. These Entities will then be associated to the appropriate `PARENT_ENTITY`, through the `PARENT_ENTITY`’s `PARENT_ROLENAME`.

The difference in this scenario is that you can use Variable Substitution to better limit what is retrieved from the Database.

In Example #2 in “Associated Entities without Variable Substitution”, it was shown the by not using Variable Substitution, the `SQL` statement could retrieve extra rows that would not be added to the working memory.

The optimal use of Variable Substitution for Associated Entities is to try match the `WHERE` clause as closely as possible to the Foreign Key relationship between the two Tables. This relationship would also be expressed in the Vocabulary Association’s Join Expression.

In the examples below, we will revisit the examples in “Associated Entities without Variable Substitution” and show how Variable Substitution would better refine the retrieval of data.

### Example 1:

Original `SQL` without Variable Substitution = `Select * from M_B`

New `SQL` with Variable Substitution = `Select * from M_B where a_id IN ({ A.id })`

`PRIMARY_NAME` = `B`

`PARENT_ENTITY` = `A`

`PARENT_ROLENAME` = `toB`

As in a previous example, here is the Data in the `M_A` and `M_B` Table.

M_A		M_B	
ID		ID	A_ID
1		1	1
2		2	1
3		3	1
		4	2
		5	2
		6	3
		7	3
		8	3
		9	4
		10	4
		11	5
		12	5
		13	6

Using the following SQL (which includes Variable Substitution):

```
Select * from M_B where A_ID IN ({ A.id })
```

This then gets updated to `Select * from M_B where A_ID IN ( 1, 2, 3 )`

The SQL retrieval will only retrieve eight rows because of the `WHERE` clause.

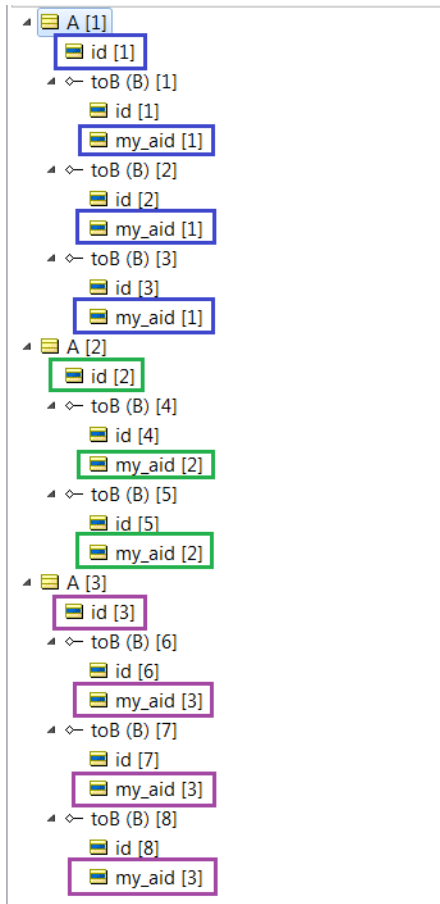
As noted before, the Variable Substitution should follow the Foreign Key relationship between the two Tables. In this case, `M_A` has a 1 -> Many relationship to `M_B`, where `M_B.A_ID` is a Foreign Key back to `M_A.ID`.

This Foreign Key relationship is expressed in the `WHERE` clause of `A_ID IN ({ A.id })`, which translates to `M_B.A_ID = M_A.ID` where `ID` is IN the set of ( 1, 2, 3 ).

The Vocabulary Association's Join Expression would also be expressed as: `M_B.A_ID = M_A.ID`

The results from this example will be the same as the results from "Associated Entities without Variable Substitution" Example #2. However, with less rows retrieved, translating the Database data into Corticon working memory will be faster.

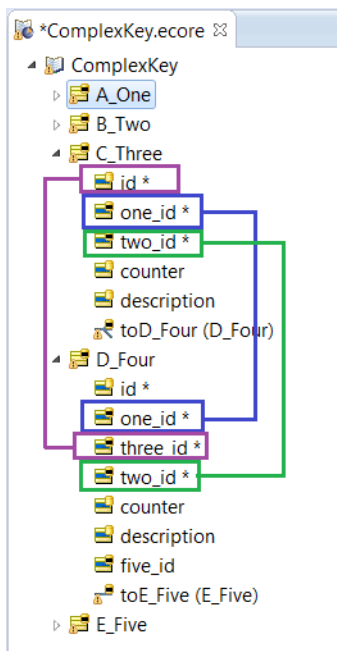




### Example 2:

This example is more complicated because the Foreign Key relationship includes multiple Columns. But as described in the previous example, the SQL statement with Variable Substitution should represent what the Foreign Key relationship is between the two Database Tables.

Revisiting “Associated Entities without Variable Substitution” Example #3...



Join Expression for C\_Three.toD\_Four:

```
Q_THREE.ID = Q_FOUR.Q_THREE_ID,
Q_THREE.Q_ONE_ID = Q_FOUR.Q_ONE_ID,
Q_THREE.Q_TWO_ID = Q_FOUR.Q_TWO_ID
```

This Join Expression would translate to the following Where clause:

```
:: Where Q_THREE.ID = Q_FOUR.Q_THREE_ID AND Q_THREE.Q_ONE_ID = Q_FOUR.Q_ONE_ID AND
      Q_THREE.Q_TWO_ID = Q_FOUR.Q_TWO_ID
```

Now replace the Q\_THREE values with Vocabulary Entity.Attribute information using an IN clause.

Final updated SQL statement:

```
Select * from Q_FOUR Where Q_FOUR.Q_THREE_ID IN ( {C_Three.id} )
      AND Q_FOUR.Q_ONE_ID = ( {C_Three.one_id} )
      AND Q_FOUR.Q_TWO_ID = ( {C_Three.two_id} )
```

```
SQL = Select * from Q_FOUR Where Q_FOUR.Q_THREE_ID IN ( {C_Three.id} )
      AND Q_FOUR.Q_ONE_ID = ( {C_Three.one_id} )
      AND Q_FOUR.Q_TWO_ID = ( {C_Three.two_id} )
```

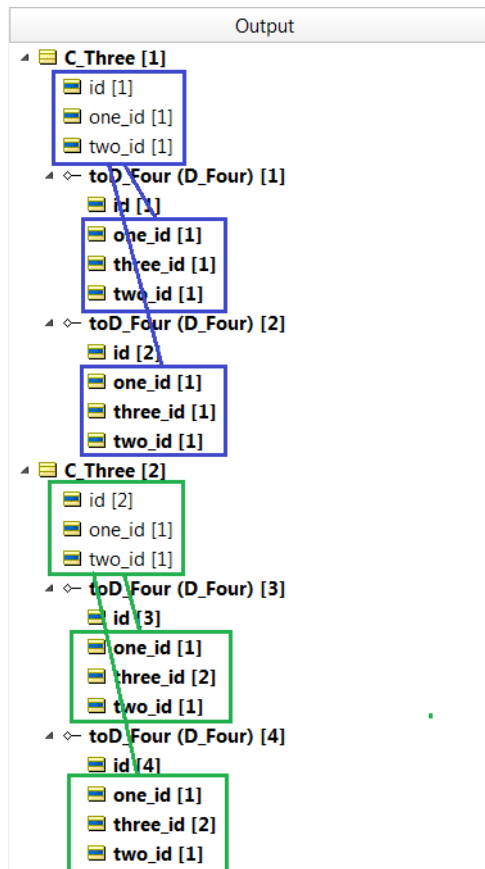
```
PRIMARY_NAME = D_Four
PARENT_ENTITY = C_Threes
PARENT_ROLENAME = toD_Four
```

Using the following data inside Q\_Three and Q\_Four:

Q_Three			Q_Four			
ID	ONE_ID	TWO_ID	ID	ONE_ID	TWO_ID	THREE_ID
1	1	1	1	1	1	1
2	1	1	2	1	1	1
			3	1	1	2
			4	1	1	2
			5	2	1	1
			6	2	1	2
			7	2	2	1
			8	2	2	2
			9	3	1	1
			10	3	1	2

Those rows with Q\_Four.ID = 5 through 10 will be filtered out by the WHERE clause, which reduces the ResultSet from the Database.

The results below are the same as in the Associated Entities without Variable Substitution Example #3.



## Tips and Techniques

The following topics describe insights into techniques and behavior you might find useful.

### When to use Comparison Operators instead an IN ( ) in the WHERE clause

It is highly recommended to use an `IN ( )` clause instead of an `=` sign in your `WHERE` clause. In essence, they mean the same thing, yet, the `IN ( )` clause can handle multiple values, while the `=` sign can only handle one value.

As outlined in the first example, there are three `A` Entities in memory. That means there are three values for { `A.id` }. In the following SQL note that the one with the `IN ( )` is valid while the `=` sign is not:

```
Select * from M_A where id IN ( 1, 2, 3 ) Valid
Select * from M_A where id = 1, 2, 3      Invalid
```

You cannot use an `IN` clause with `<`, `<=`, `>`, and `=>`. To prevent invalid `SQL` through Variable Substitution with `<`, `<=`, `>`, and `=>`, there can only be one instance of the Entity in working memory.

### Inserting multiple rows into specific Database Table(s)

```
SCO Service Name = CcDatabaseServiceCallout.insertEntitiesIntoDatabase
```

The ADC will use the metadata inside the `CCDATASOURCES`, `CCUPDATES`, and `CCINSERTS` Tables to determine which Entities in the Vocabulary will be used to insert into which Database Table.

The core Table that contains which Entity or Entities will be inserted into the Database is in the `CCINSERTS` Table. This section describes how the `SEQUENCE`, `SQL`, `PRIMARY_NAME` are used in one or multiple `CCINSERTS` to insert multiple records into the intended Table.

Much like the CCREADS' SEQUENCE field, the CCINSERTS' SEQUENCE field determines in which order the CCINSERTS will fire. For each CCINSERTS' SQL, there is a PRIMARY\_ENTITY, which is used to create individual Insert Statements to be used by the Database.

Variable Substitution is used to substitute the PRIMARY\_ENTITY values into the SQL Statement.

#### Example:

```
SQL = Insert into Z_OUTPUT1 (ID, TEMP01, TEMP02, TEMP03, TEMP04, TIMESTAMP)
      VALUES (zOutput1Sequence.nextval, {zOutput1.temp01} ,
              {zOutput1.temp02}, {zOutput1.temp03}, {zOutput1.temp04},
              TO_DATE(sysdate, 'YYYY/MM/DD HH:MI:SS'))

PRIMARY_ENTITY = zOutput1
```

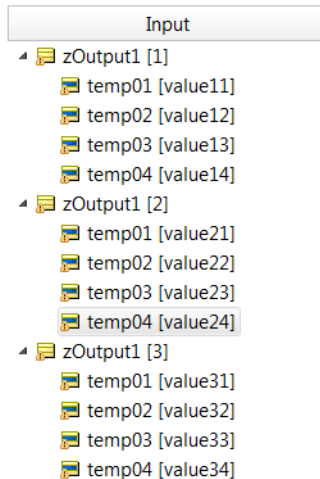
For every instance of zOutput1 in memory a new SQL Statement will get created using those values inside the zOutput1 instance.

With regards to the SQL statement, the user controls it, and can customize the SQL to match the Identity Strategy appropriate for a particular Database:

- In Oracle, Database Sequences are used to set the Primary Keys. You need to create your own Database Sequence and add that Sequence Name to the SQL statement. In the above SQL, the first VALUE is zOutput1Sequence.nextval – the name of the Sequence that is used in this example.
- In SQL Server, you can just set your Table to use Identity strategy to populate the Primary Key. In this case, the above SQL wouldn't have a Column: ID and VALUE: zOutput1Sequence.nextval in it.

Also, because you have control over the SQL, you can inject Database specific Value/Functions directly in the SQL. In this example, Column: TIMESTAMP is not related to Entity zOutput1. The value is determined by the sysdate function on Oracle.

Consider this data to be what is in working memory:



There are three zOutput1 Entity instances. Each one will use the SQL statement as a template to create their own INSERT statement.

```
Insert into Z_OUTPUT1 (ID, TEMP01, TEMP02, TEMP03, TEMP04, TIMESTAMP)
      VALUES (zOutput1Sequence.nextval, 'value11' , 'value12', 'value13', 'value14',
              TO_DATE(sysdate, 'YYYY/MM/DD HH:MI:SS'))
Insert into Z_OUTPUT1 (ID, TEMP01, TEMP02, TEMP03, TEMP04, TIMESTAMP)
      VALUES (zOutput1Sequence.nextval, 'value21' , 'value22', 'value23', 'value24',
              TO_DATE(sysdate, 'YYYY/MM/DD HH:MI:SS'))
Insert into Z_OUTPUT1 (ID, TEMP01, TEMP02, TEMP03, TEMP04, TIMESTAMP)
      VALUES (zOutput1Sequence.nextval, 'value31' , 'value32', 'value33', 'value34',
              TO_DATE(sysdate, 'YYYY/MM/DD HH:MI:SS'))
```

**Note:** Technical note: These are actually Prepared Statements in Batch Mode, but the concept that each instance of `zOutput1` gets its own SQL statements is easier to present here.

## Entity/Attribute names do not need to match Database Table/Column names

This section describes the level of integration between the Vocabulary Entity/Attribute/Association and the Database Table/Columns.

`CcRetrievals` will retrieve data from the Database and incorporate that data into the working memory of the existing execution. The way the ADC translates the data into working memory is by using what has been entered in the Vocabulary for Table Name (for Entities), Column Name (for Attributes), and Join Expression (for Associations).

### Entity -> Table Name (optional):

If this field is entered for the Entity, then it will be used by the ADC. If it is not entered, then the Entity Name will be used as the Table Name. It is only necessary to enter in a Table Name if the Vocabulary Entity Name does not match the Table Name in the Database.

#### Example 1: (Table Name not specified)

The screenshot displays the ADC configuration interface. On the left, the 'DatabaseSCO' vocabulary is shown with entities `M_A` and `M_B`. The middle pane shows the configuration for `M_A`, where the 'Table Name' field is empty. The right pane shows the 'DATABASE TABLES' with tables `M_A` and `M_B`. Orange lines connect `M_A` to its configuration and to the `M_A` table in the database. A blue box highlights the empty 'Table Name' field, with a note 'Table Name not specified in Vocabulary'.

Property Name	Property Value
Entity Name	M_A
Entity Identity	id
Inherits From	
XML Namespace	
XML Element Name	M_A
Java Package	
Java Class Name	
Datastore Persistent	Yes
Table Name	
Datastore Caching	
Identity Strategy	Table Name not specified in Vocabulary
Identity Column Name	
Identity Sequence	
Identity Table Name	
Identity Table Name Column Name	
Identity Table Value Column Name	
Version Strategy	
Version Column Name	

#### Example 2: (Table Name specified, Entity Name can be anything)

DatabaseSCO.ecore

DatabaseSCO

- A
  - id \*
  - my\_column1
  - my\_column2
  - my\_column3
  - my\_column4
  - my\_column5
- toB (B)
  - id \*
  - my\_aid
  - my\_column1
  - my\_column2
  - my\_column3
  - my\_column4
  - my\_column5
- toC (C)

If Table Name specified, then Entity Name can be anything.

Property Name	Property Value
Entity Name	A
Entity Identity	id
Inherits From	
XML Namespace	
XML Element Name	A
Java Package	
Java Class Name	
Datastore Persistent	Yes
Table Name	DATASYNC.M_A
Datastore Caching	
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	

DATABASE TABLES

Tables

- M\_A
  - ID
  - COLUMN1
  - COLUMN2
  - COLUMN3
  - COLUMN4
  - COLUMN5
- M\_B
  - ID
  - A\_ID
  - COLUMN1
  - COLUMN2
  - COLUMN3
  - COLUMN4
  - COLUMN5
- M\_C
- M\_D

### Attribute -> Column Name (optional)

This is similar to Entity -> Table Name. If the Column Name field is not entered, then the Attribute Name will be used for the Column Name.

### Example 1: (Column Name not specified)

\*DatabaseSCO.ecore

DatabaseSCO

- M\_A
  - id \*
  - column1
  - column2
  - column3
  - column4
  - column5
- toB (M\_B)
- M\_B
- M\_C
- M\_D
- Parameters
- zOutput1
- zOutput2

Additional Attributes that have no Column Name specified.

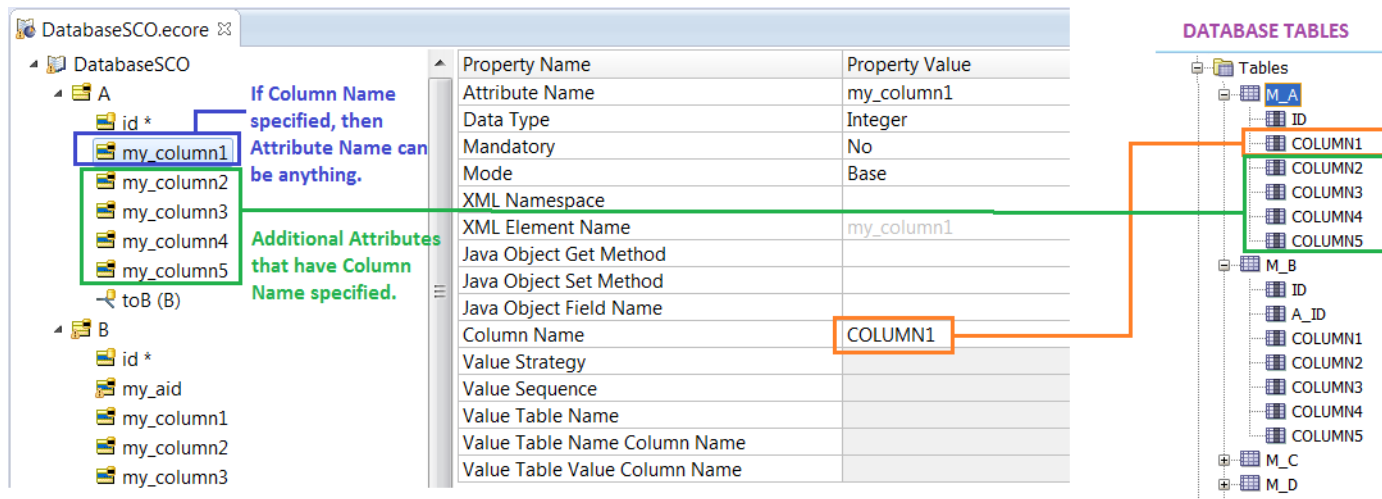
Property Name	Property Value
Attribute Name	column1
Data Type	Integer
Mandatory	No
Mode	Base
XML Namespace	
XML Element Name	column1
Java Object Get Method	
Java Object Set Method	
Java Object Field Name	
Column Name	column1
Value Strategy	
Value Sequence	Column Name not specified in Vocabulary
Value Table Name	
Value Table Name Column Name	
Value Table Value Column Name	

DATABASE TABLES

Tables

- M\_A
  - ID
  - COLUMN1
  - COLUMN2
  - COLUMN3
  - COLUMN4
  - COLUMN5
- M\_B
  - ID
  - A\_ID
  - COLUMN1
  - COLUMN2
  - COLUMN3
  - COLUMN4
  - COLUMN5
- M\_C
- M\_D

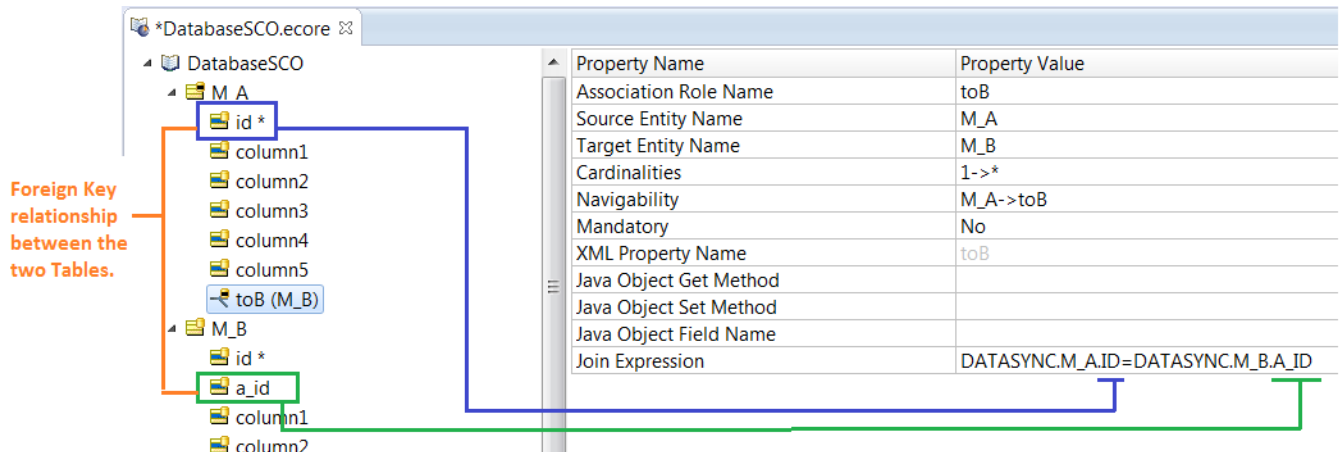
### Example 2: (Column Name specified, Attribute Name can be anything)



### Association -> Join Expression (required):

Join Expressions are required because there is no way to determine what the Foreign Key relationships are. If Database Metadata has been imported into the Vocabulary and the Vocabulary is able to decipher the Foreign Keys based on the Metadata, then the Join Expression doesn't need to be manually entered.

If the Join Expression needs to be manually added, the Schema name will need to precede the Table name. In this example, DATASYNC is the Schema name.



### Multiple ADC instances can be added to one or many Ruleflows

There is no restriction on how many ADC instances you can have in your Ruleflow. Its position on the Ruleflow canvas is based on your Use Case. When retrieving extra data is only needed in certain cases, you can put an ADC instance inside a Branch that will only fire under certain conditions. This can also be applied to CCUPDATES.

Example:

In this example, there are two ADC instances in the main Ruleflow. The CcRetrievals SCO is embedded in another Ruleflow, 63-RetrieveABCD.erf, which is added to the main Ruleflow, 81-RetrieveABCD\_UpdateBoth.erf, as:

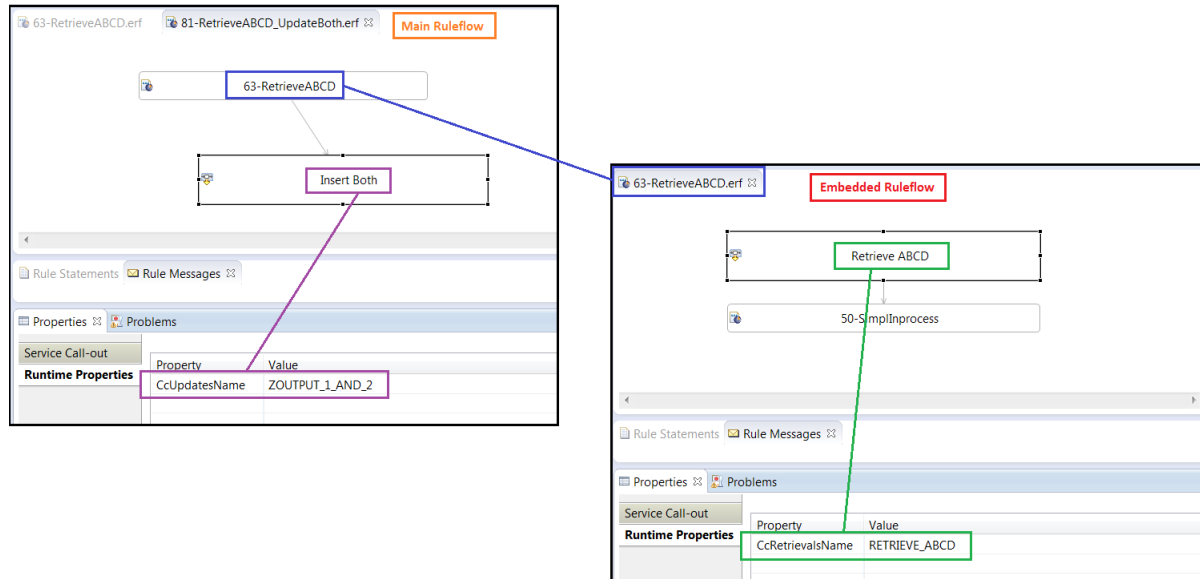
```
ADC (Retrieve ABCD) - CcDatabaseServiceCallout.retrieveEntitiesFromDatabase
```

```
CcRetrievalsName = RETRIEVE_ABCD
```

```
ADC (Insert Both) - CcDatabaseServiceCallout.insertEntitiesIntoDatabase
```

```
CcUpdatesName = zOUTPUT_1_AND_2
```

These instances are highlighted in this illustration of the Main Ruleflow and the Embedded Ruleflow:



Each instance of the ADC works independently to do what it is assigned to do.

## Each ADC task can use a different DataSource

Each instance of an ADC can call any `CCRETRIEVALS` or `CCUPDATES` operation, and for each `CCRETRIEVALS` and `CCUPDATES`, there is a `CCDATASOURCES` configuration. There is no restriction that all records in the `CCDATASOURCES` Table point to the same Database instance or even the same Database type.

### Example:

Two records in `CCDATASOURCES` Table:

- ID = 1, NAME=Corticon : This will be used by a `CCRETRIEVALS` to retrieve data from an Oracle Database.
- ID = 2, NAME=Corticon (inserts) : This will be used by a `CCUPDATES` to insert data into a SQL Server Database.

CCDATASOURCES				
Columns   Data   Constraints   Grants   Statistics   Triggers   Flashback   Dependencies   Details   Partitions   Indexes   SQL				
ID	NAME	DRIVER	URL	
1	1 Corticon	com.corticon.database.id.Oracle	jdbc:progress:oracle://172.29.39.120:1521;databaseName=corticon	
2	2 Corticon (inserts)	com.corticon.database.id.MsSql2014	jdbc:progress:sqlserver://172.29.37.10:1433;databaseName=QueryTes	

## Information when execution fails

Various errors can occur during the execution of the ADC. Some common issues are:

- `CcRetrievalsName` or `CcUpdatesName` does not exist.
- Bad SQL statement, possibly due to Variable Substitution issues.
- Bad Join Statement definition for an Association.
- Failed to connect to the Database.

Whatever the type of error, execution will not only stop on the SCO, but for the entire execution. If there is an issue in the SCO, then current working memory could be incomplete or corrupted. Either way, the safest play is to stop all execution.



Also, there will be an entry in the Corticon Log, with the Exception, and a CcRuleMessage -> Violation message added to the Response.

#### Example: (Database URL is bad)

The screenshot shows the Corticon IDE interface. The top pane displays the rule execution flow for `*60 - Tester.ert`. The rule is `/_DatabaseSCO/SimpleKeys/60-RetrieveA.ert`. The Input pane shows a collection `A [1]` with an element `id [1]`. The Output pane shows a collection `A [1]` with an element `id [1]`, and a `zOutput1 [1]` collection containing `temp01 [1]`, `temp02 [0]`, `temp03 [0]`, `temp04 [0]`, and `timestamp [28/03/17 2:26:24 PM]`.

The bottom pane shows the Rule Messages tab. It displays two messages:

Severity	Message
Violation	Failed to make a Connection with CcDataSource Name = Corticon
Violation	CcDatabaseServiceCallout.getConnection() Wrapping Exception in a SQLException

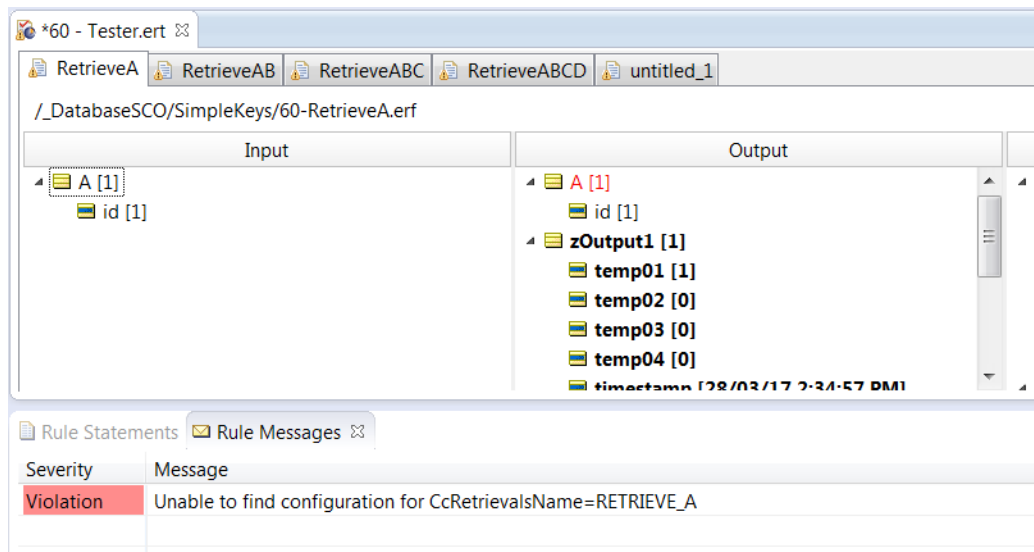
#### Example: (Bad SQL Statement)

The screenshot shows the Corticon IDE interface. The top pane displays the rule execution flow for `*60 - Tester.ert`. The rule is `/_DatabaseSCO/SimpleKeys/63-RetrieveABCD.ert`. The Input pane shows a collection `A [1]` with an element `id [1]`, and a collection `A [2]` with an element `id [2]`, and a collection `A [3]` with an element `id [3]`, and a collection `A [4]` with an element `id [4]`. The Output pane shows a collection `A [1]` with an element `id [1]`, and a collection `my_column1 [1]`, `my_column2 [1]`, `my_column3 [1]`, `my_column4 [1]`, and `my_column5 [1]`.

The bottom pane shows the Rule Messages tab. It displays three messages:

Severity	Message
Violation	CcRead at sequence (4) SQL Variable Replacement for : (C.idssssssss) failed. No such Attribute Name (idssssssss) for Entity (C) in Vocabulary.
Violation	CcRetrievals (RETRIEVE_ABCD) CcRead at sequence (4) Exception thrown when trying to retrieve from the Database.
Violation	CcRead at sequence (4) SQL Variable Replacement for : (C.idssssssss) failed. No such Attribute Name (idssssssss) for Entity (C) in Vocabulary.

#### Example: (Bad CcRetrievalsName)



## Scripts and examples

A Corticon installation provides scripts and examples located at `[CORTICON_HOME]\addons\adasco`.

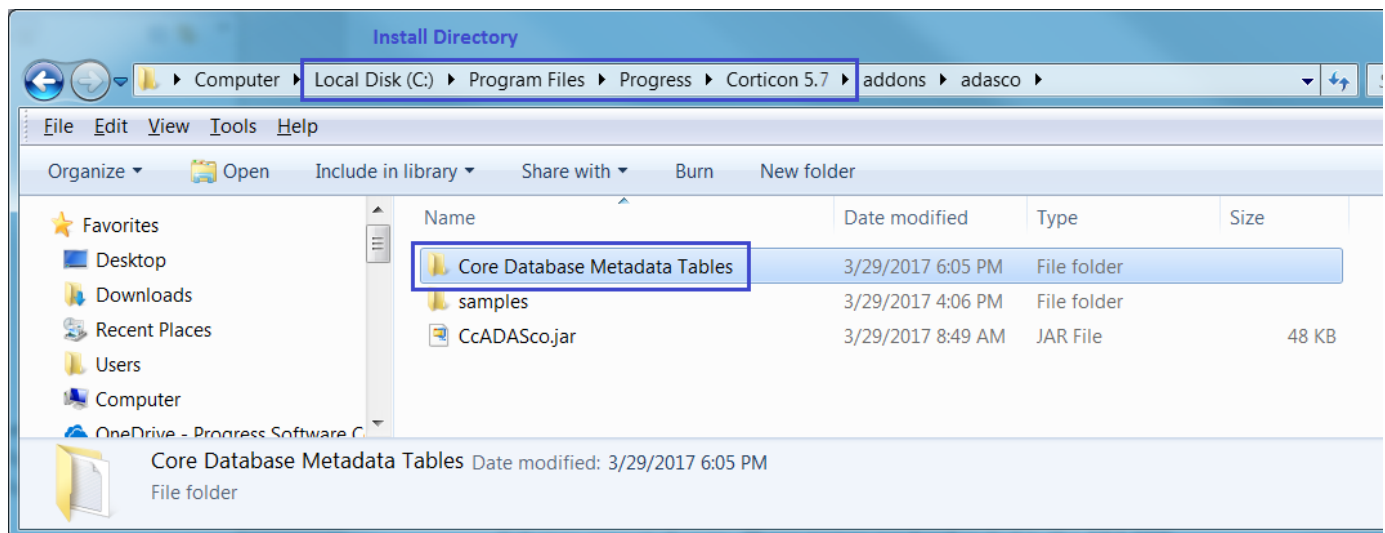
**Note:** The JAR file at that location, `CcADASco.jar`, must be added to your project. See [Building the Java classes and JARs](#) on page 69 for instructions.

## Installed Database Scripts that create required tables

Each type of Database can have subtle differences in their DDL (Data Definition Language). We are going to supply DDL Scripts (SQL Scripts) to create the necessary Tables in an Oracle Database. It is expected that a qualified DBA will be able to make the necessary alterations to the scripts so that they are compatible with their Database.

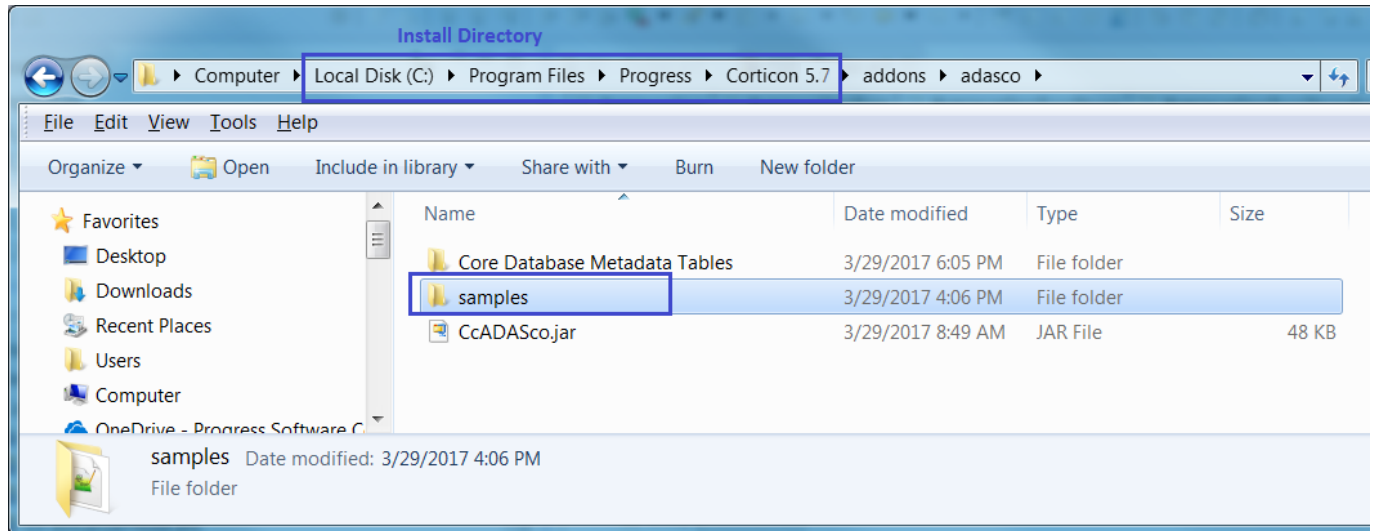
The only caveat about having the DBA alter the scripts would be, the **names** of the Tables and Columns must not be changed as these names are hardcoded to functionality in the ADC.

The SQL Scripts to generate the Oracle Database are distributed in Corticon Studio and Server Installers, as shown:



## Rule Assets with Database Scripts to populate core Database Tables

The Rule Assets shown in the illustrations in this section are distributed in Corticon Studio and Server Installers, as shown:



### Sample Simple Keys

Sample Simple Keys applies when the Database Tables only have one Column as a Primary Key. This makes Associations (through Join Expressions) similar to those using a multiple Columns as a Primary Key (which is covered in the ComplexKeys example). The sample provides database scripts and Corticon rule assets

#### Database Scripts for Sample Simple Keys

The order of running the Database Scripts is:

1. **InsertsIntoCoreMetadataTables.txt** - Inserts all the necessary Metadata into `CCRETRIEVALS`, `CCREADS`, `CCUPDATES`, and `CCINSERTS` Tables. A `CCDATASOURCES` record with `ID=1` must already be in the Table; otherwise, the `INSERT` Statements will fail because they rely on a Foreign Key `CCDATASOURCES_ID` back to `CCDATASOURCES` with `ID=1`.
2. **CreateTables.txt** - Creates the core Tables that will be retrieved or inserted into the execution of the Decision Service.
3. **DATA\_M\_A.csv**, **DATA\_M\_B.csv**, **DATA\_M\_C.csv**, **DATA\_M\_D.csv** - The `.csv` files that you can import into the Table that were created with `CreateTables.txt`. **NOTE:** Import them in the order (A, B, C, D) as there are Foreign Key considerations.

#### RuleAssets for Sample Simple Keys:

- **DatabaseSCO.ecore** - Vocabulary pre-configured with all the Database Mappings with Join Expressions for the Associations. The Database Mappings go hand in hand with the Database Scripts' `CreateTables.txt`.
- **Rulesheets** - These are just for initialization of variables to make the example cleaner.
- **Ruleflows** - A variety of Ruleflows that have ADC configured to run against different `CcRetrievalsName` and `CcUpdatesName`. These values go hand in hand with the data that was imported into the `CCRETRIEVALS` and `CCUPDATES` using the `.csv` files.
- **Ruletests** - Several Ruletests, `## - Tester.ert`, that demonstrate the various rule processing examples.

## Sample Complex Keys

Sample Complex Keys is an example that applies when the Database Tables have cascading Primary Keys from one Table to another. Because Tables will have multiple Columns in the Primary Key, this complicates Associations and the corresponding Join Expressions.

As discussed in: “Associated Entities with no Variable Substitution”, **Example 3**, the Join Expression that links three Columns on each Entity together is:

```
Q_THREE.ID = Q_FOUR.Q_THREE_ID,  
Q_THREE.Q_ONE_ID = Q_FOUR.Q_ONE_ID,  
Q_THREE.Q_TWO_ID = Q_FOUR.Q_TWO_ID
```

This sample also provides Database Scripts and RuleAssets.

### Database Scripts for Sample Complex Keys

The order of running the Database Scripts is:

1. InsertsIntoCoreMetadataTables.txt
2. CreateTables.txt
3. DATA\_Q\_ONE.csv, DATA\_Q\_TWO.csv, DATA\_Q\_THREE.csv, DATA\_Q\_FOUR.csv,  
DATA\_Q\_FIVE.csv

### RuleAssets for Sample Complex Keys :

- **ComplexKey.ecore** - Vocabulary pre-configured with all the Database Mappings with Join Expressions for the Associations. The Database Mappings go hand in hand with the Database Scripts' CreateTables.txt.
- **Rulesheets** - These are just for initialization of variables to make the example cleaner.
- **Ruleflows** - A variety of Ruleflows that have ADC configured to run against different CcRetrievalsName and CcUpdatesName. These values go hand in hand with the data that was imported into the CCRETRIEVALS and CCUPDATES using the .csv files.
- **Tester.ert** - A Ruletest that demonstrates the rule processing.

---

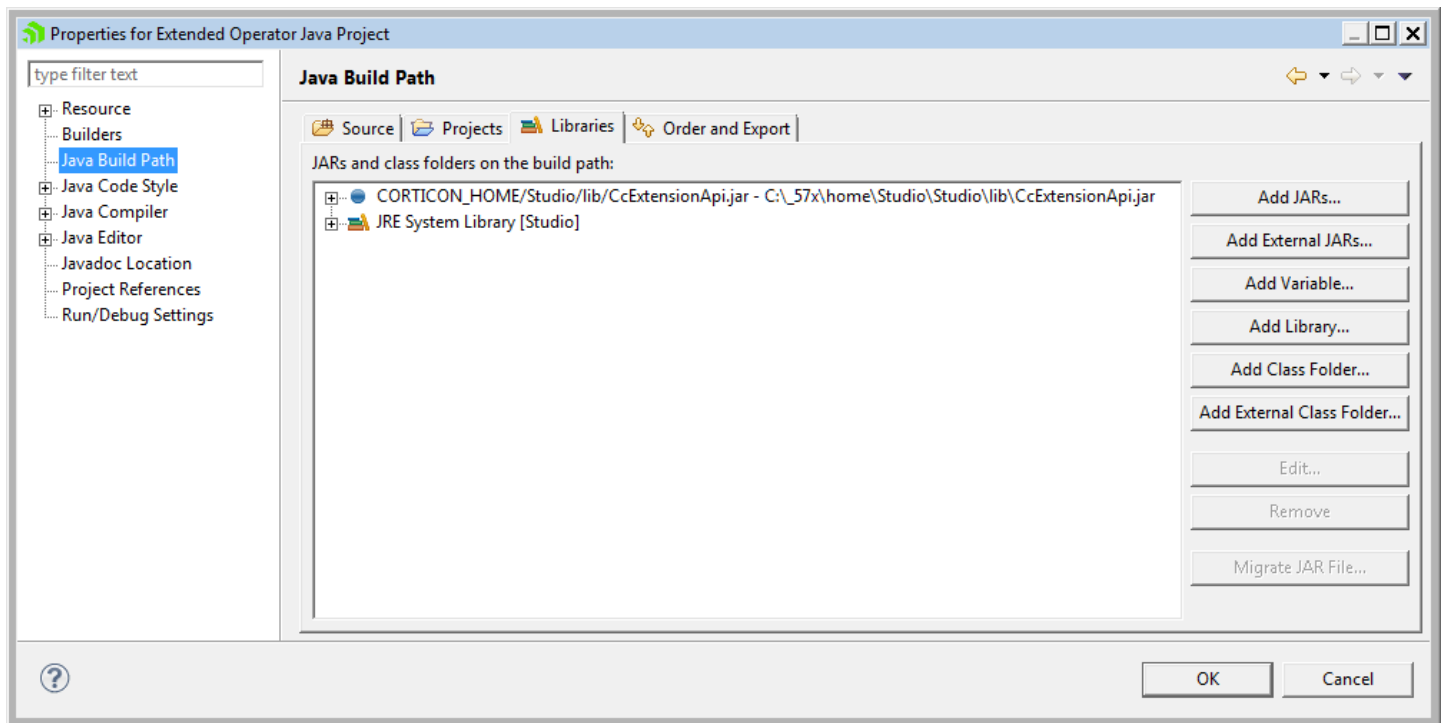
## Building the Java classes and JARs

---

The Extended Operator, Service Callout, and Advanced Data Callout samples contain Java projects demonstrating how to create a Java extension. These are standard Java projects. Each has the CorticonCorticon JAR file that defines its API:

- For Corticon extensions and Service Callouts, `CcExtensionApi.jar` located in a Studio installation at `[CORTICON_HOME]/Studio/lib/`
- For Corticon Advanced Data Callouts, `CcADAsco.jar` located in a Studio or Server installation at `[CORTICON_HOME]/addons/adasco/`

The appropriate JAR is added to the project from its Corticon installed location to the project's build path using the predefined Eclipse variable `CORTICON_HOME`. For example:



When you use the Studio feature of **Package and Deploy > Save for later Deployment**, the JAR will be added into the **.eds** file. If you use other techniques to compile and the JAR is not embedded, you need to locate the JAR on the Server, and then add it to the Server's classpath.

---

## Deploying Decision Services with extensions

---

Once you have added extension jars to your project, several deployment tools provide mechanisms to package the extension JARs into deployment. When you compile a project Ruleflow into an EDS file, the extension JARs are encapsulated within the encrypted `.eds`. That insures that regardless how you relocate or update a Decision Service, the extension JARs that are associated with it are consistent.

### Deployment from Studio

The three standard techniques in Studio that package and deploy Decision Services all incorporate the extension JARs that were associated with the project:

- Deploying directly to a server
- Deploying to a server through a Web Console application
- Packaging the EDS file locally for access by other tools or the Web Console to complete the deployment.

### Deployment using the Server's command line interface

When you use the tool `CorticonManagement` at a server's `[CORTICON_HOME]\Server\bin` location, the `compile` command provides parameters that will declare dependent JARs and then include them. Both parameters take comma-separated values and both parameters are required to achieve the packaging into EDS file.

```
-dj,--dependentjars dependentJar  add jar files required for this decision service  
-ij,--includedjars includedJar    add jar files to include in the generated eds file
```

A complete command might look like this:

```
corticonManagement
--compile
--input C:\myProject\myRuleflow.erf
--output C:\myProject\Output
--service MyDS
--dependentjars C:\myProject\myExtensions.jar,C:\myProject\myCallouts.jar
--includedjars C:\myProject\myExtensions.jar,C:\myProject\myCallouts.jar
```

With only required options specified, the result is C:\myProject\Output\MyDS.eds

### Additions to Ant macro compile arguments

If you want to use Ant macros for the `corticonManagement` command line utilities that are in the file `[CORTICON_HOME]\Server\lib\corticonAntMacros.xml`, you can set the required extension JARs in the arguments for the `compile` macro so that you can use them in the call:

```
<attribute name="input" default=""/>
<attribute name="output" default="" />
<attribute name="service" default="" />
<attribute name="version" default="false" />
<attribute name="edc" default="false" />
<attribute name="failonerror" default="false" />
<attribute name="dependentjars" default="" />
<attribute name="includedjars" default="" />
```

Example of a call to the compile macro:

```
<corticon-compile
  input="${project.home}/Order.erf"
  output="${project.home}"
  service="OrderProcessing"
  dependentjars="${project.home}/myExtensions.jar,${project.home}/myCallouts.jar"
  includedjars="${project.home}/myExtensions.jar,${project.home}/myCallouts.jar"
/>
```

---

**Note: Deployment to a Corticon .NET Server** - Once a project that includes extension JARs is packaged into a Decision Service, it deploys and performs as expected on Corticon .NET Server.

---



---

## Access to Corticon knowledge resources

---

**TUTORIALS:** Learn about Corticon from online lessons at the [Corticon Learning Center](#).

DOCUMENTATION: About this release	
<i>What's New in Corticon</i> <a href="#">PDF</a> <a href="#">HTML</a>	Describes the enhancements and changes to the product since its last point release.
<i>Corticon Installation Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	Step-by-step procedures for installing Corticon Studio and Servers in this release on Windows and Linux platforms.

DEVELOPMENT DOCUMENTATION: Define and Model Business Rules	
<i>Rule Modeling Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	Presents the concepts and purposes of 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. Describes Ruleflow features and reports. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting of Rulesheets and Ruleflows. Includes <i>Test Yourself</i> exercises and answers.
<i>Quick Reference Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.

<i>Rule Language Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	Reference information for all operators available in the Corticon Studio Vocabulary. Rulesheet and Ruletest examples are provided for many of the operators.
<i>Guide to Creating Extensions</i> <a href="#">PDF</a> <a href="#">HTML</a>	Detailed technical information about the Corticon extension framework for extended operators and service callouts. Describes several types of operator extensions, their samples, and how to create custom extensions and service callouts.
<i>Javadoc for Extensions API</i> <a href="#">HTML</a>	Complete Java API reference for Corticon Extensions.

**DEPLOYMENT DOCUMENTATION: Run Decision Services on Servers**

<i>Integration and Deployment Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the various publishing mechanisms. 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. Shows how to setup and use rule execution recording. Details the Java and REST APIs, and the complete set of exposed Corticon properties.
<i>Web Console Guide</i> <a href="#">PDF</a> <a href="#">HTML</a>	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.
<i>Deploying Web Services with Java</i> <a href="#">PDF</a> <a href="#">HTML</a>	Details setting up an installed Corticon Server as a Web Services Server, and then deploying and exposing Decision Services as Web Services on the <a href="#">Progress Application Server (PAS)</a> and other Java-based servers. Includes samples of XML and JSON requests.
<i>Deploying Web Services with .NET</i> <a href="#">PDF</a> <a href="#">HTML</a>	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.
<i>Javadoc for Corticon Server API</i> <a href="#">HTML</a>	Complete Java API reference for Corticon Server.