Progress® Corticon®

Corticon Extensions

Generated on: 1 October 2025

1

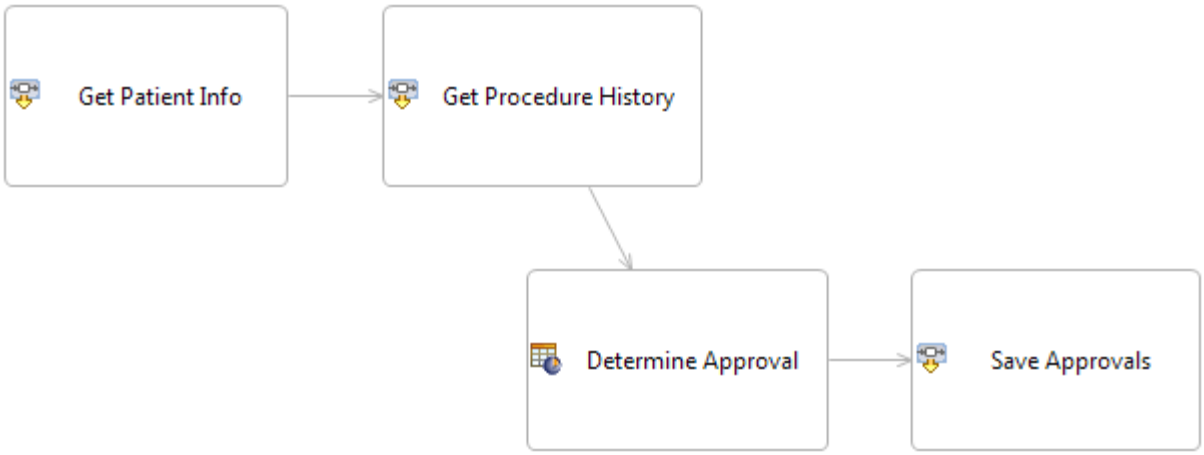
How to create custom service callouts

Service Callout (SCO) 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 Callout (SCO) 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 Callouts use the Progress Corticon Extension API to retrieve and update facts. The package `com.corticon.services.dataobject` contains two Java interfaces:

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

Your Service Callout 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 Callout methods must be declared `public static`.

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

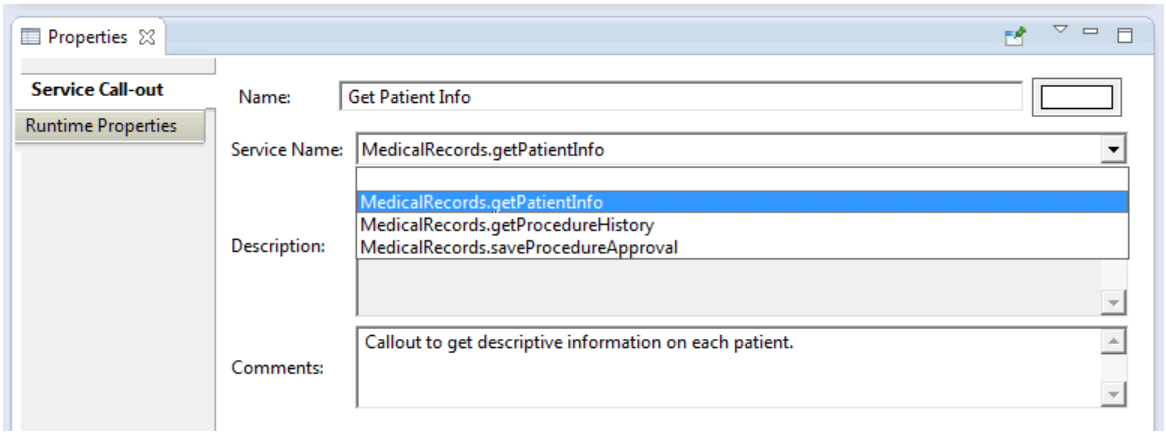
```
        public static void getPatientInfo(ICcDataObjectManager dataObjMgr) {
                // Get the set of "Patient" entities.
                Set<ICcDataObject> patients =
dataObjMgr.getEntitiesByName("Patient");
                if (patients != null) {
                        // Process each 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 Callout object is on a Ruleflow canvas:



The Service Callout 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.

Contents

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
```

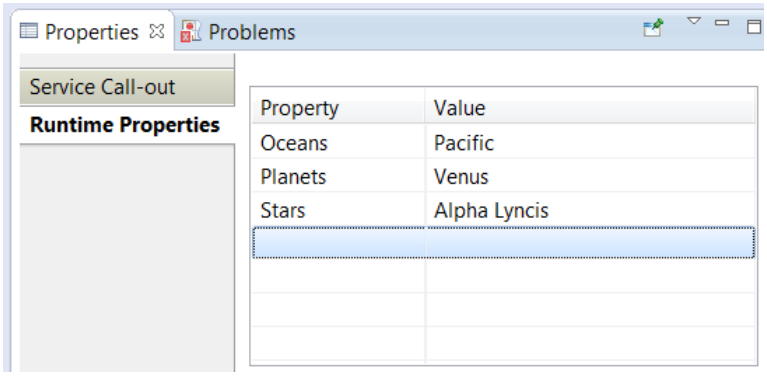
For details about the `IVocabularyMetadata` object, see the topic How to access the Vocabulary metadata of a Decision Service

Specify properties on a service callout instance

You can specify properties on a Service Callout (SCO) 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 Service Callout (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 Service Callout (SCO) must implement a specific Interface and the following methods for the Ruleflow to list the possible Property Names and their respective Values.

Interface:

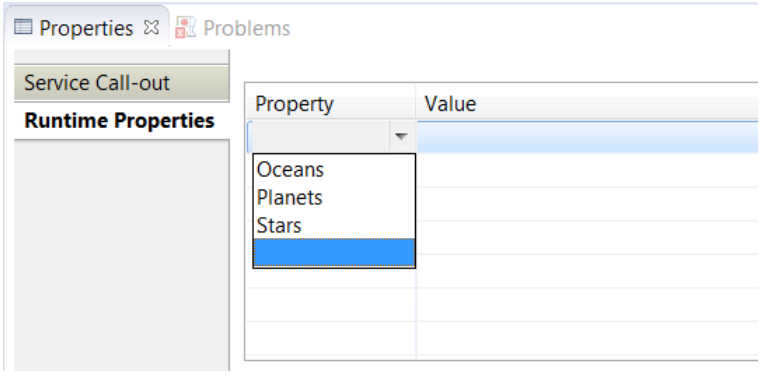
```
com.corticon.services.extensions.ICcPropertyProvider
```

Static Methods:

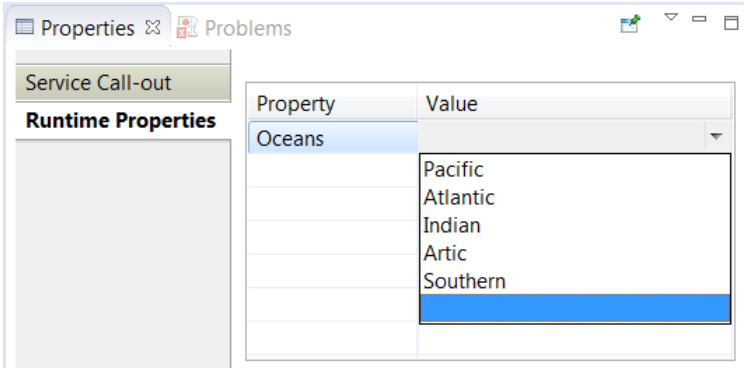
```
public List<String> getPropertyNameOptions()throws Exception;

public List<String> getProperyValueOptions(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-down list where the user selects the value:



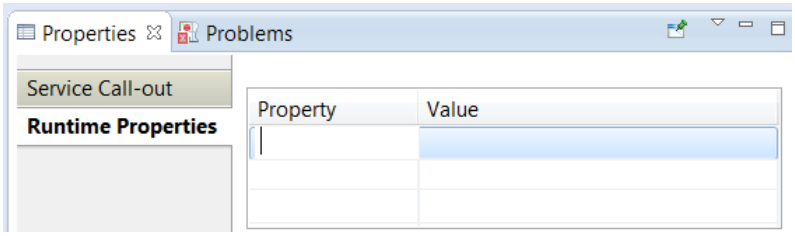
Note: This technique does not allow additional name/value pairs to be entered.

No defined Property Names and Values

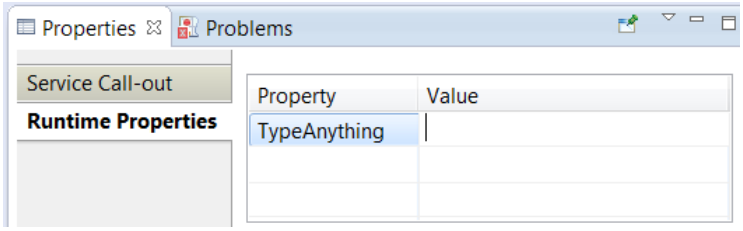
Undefined Property Names and Values occur when:

- 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 Text Boxes where names and values can be typed on many lines:



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



Note: Property Name and Value lists work independently - While `getPropertyNameOptions()` might return a `List<String>` with values so that the cell on the current **Property** line offers a drop-down 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.

Add the extension to the Rule Project, and then test it.

In order for the extension to get referenced by a Rules project, and then packaged with a Decision Service, you must Add the compiled Java classes to the Rules Project.

Create and run Ruletests that evaluate the range of possible values that could be presented to the extension. Be sure to include blanks and nulls so that you get complete coverage.