

Decision Modeling

An Introductory Workshop

With

Corticon Studio for Analysts

6/14/2013

V3

Mike Parish

mparish@progress.com

Table of Contents

- Outline 4
- Summary of Key Steps 4
- Preparation 5
 - Download Corticon Studio and Server 5
- Basic Example 6
 - Open Studio 6
 - Create a Rule Project 6
 - Create the Vocabulary 7
 - Model the Rules..... 11
 - Test the Rules 13
 - Resolving the Problems 16
- What Happens when it gets More Complex?..... 19
 - Add Custom Data Types 20
 - Date Operators 21
 - Rules for Invalid or Unrecognized Values. 23
- Advanced Example..... 24
 - Establish the Relationship between Groups and Persons..... 26
 - Create a Rule Flow 31
 - Create a Test Sheet..... 31
 - Add Some New Rules 33
 - Add the Individual Risk Rule Sheet to the Rule Flow 36
 - Some things to Consider 39
 - Another Enhancement..... 39
 - Risk Scoring..... 39
- How to Deploy Decision Services 40

Install the Corticon Server.....	40
Start the Corticon Server	40
Configure the Decision Service	41
Publish the Decision Service	42
How to Monitor Decision Services	45
Start the Web Console	45
Configure the Decision Service	49
How to Execute Decision Services	50
From Corticon Studio	50
From SOAP UI	53
Start SOAP UI	53
Create a new WSDL Project.....	53
Enter Details for the New Project.....	54
Create the Corticon Request block.....	57
Create a new TestSuite	59
Create a new TestCase	60
How to Execute Decision Services from Open Edge	64
Create an Open Edge Project	65
Create the Corticon Decision Service	65
Convert the WSDL into ABL.....	65
Add ABL code to control the GUI	65
Appendix.....	67
A Skydiver Client GUI in Open Edge	67
ABL Code (Overall Structure)	68

Outline

This workshop covers all the steps necessary to model, analyze, test and deploy, execute and monitor a Corticon Decision service.

It also shows how this decision service can be called from Open Edge via web services.

Summary of Key Steps

1. Download the installers
2. Install Corticon Studio and Server
3. Open Studio
4. Create a new rule project
5. Create a new vocabulary
6. Create a new rule sheet
7. Create a new test sheet
8. Create a rule flow
9. Start the Server
10. Publish the rule flow (as a Decision Service)
11. Create WSDL (using the deployment console)
12. Import WSDL into your client code
13. Set your data as the web service request
14. Call the web service
15. Extract the results from the web service response

Preparation

Download Corticon Studio and Server

<http://bit.ly/TfsIBN> or use installers on the USB drive

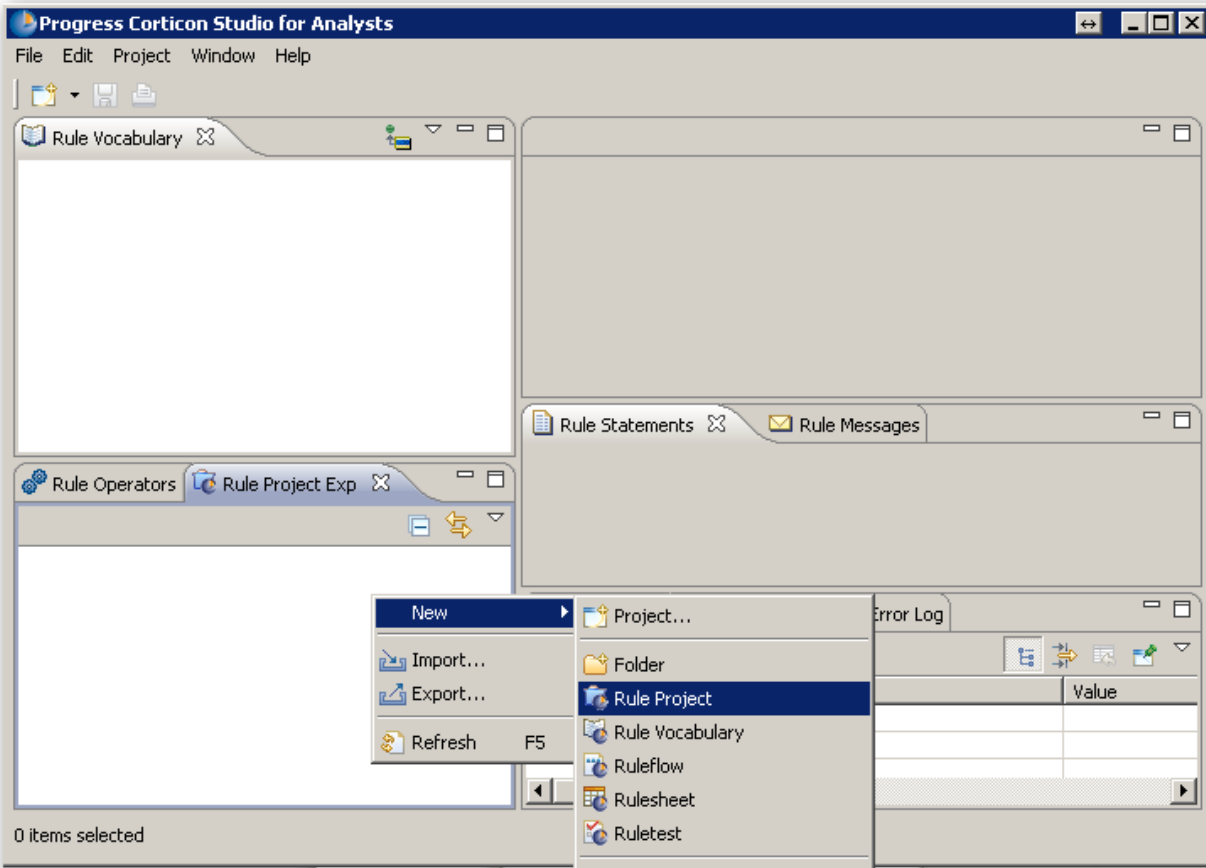
No special configuration required

Basic Example

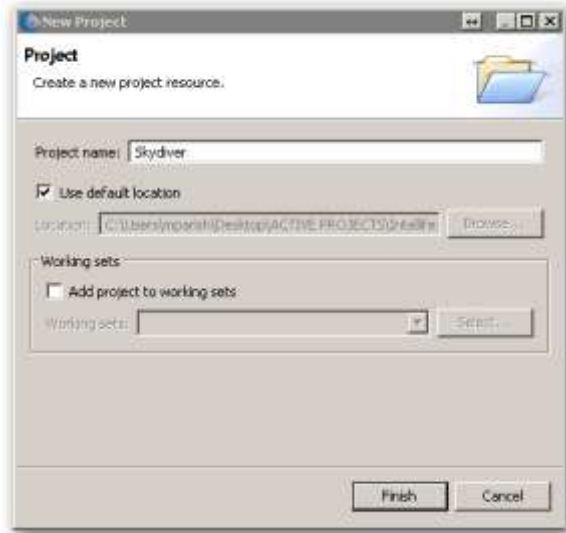
Open Studio

When you first open studio all the sections will be blank

Create a Rule Project

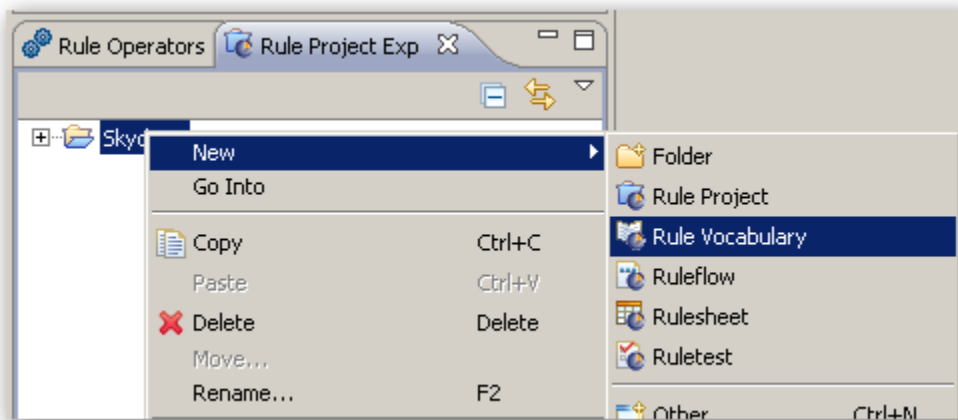


1. Right click in the Rule Project Explorer window and create a new project.
2. Name it "Skydiver"

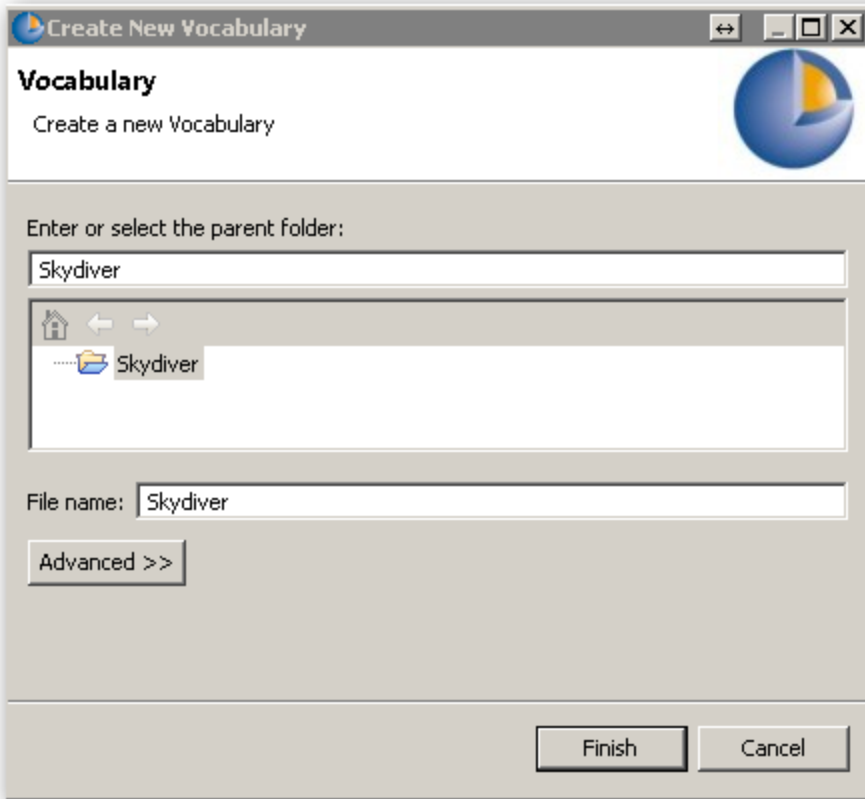


Create the Vocabulary

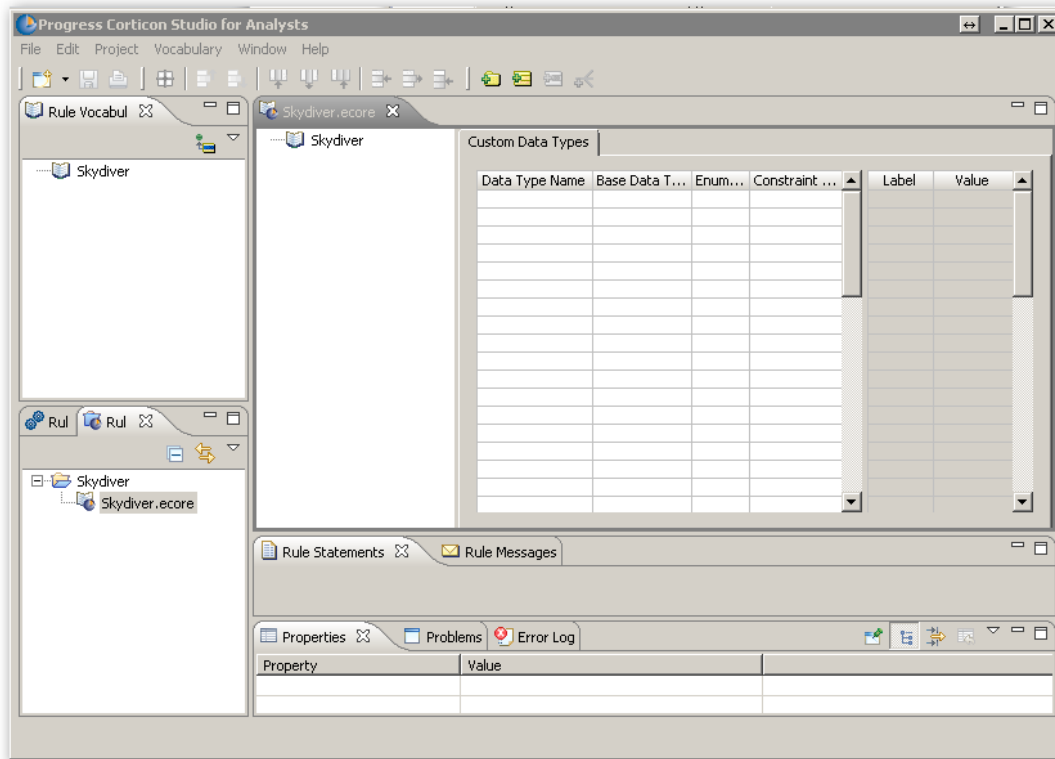
1. Right click on the project in the Project Explorer window and create a new vocabulary



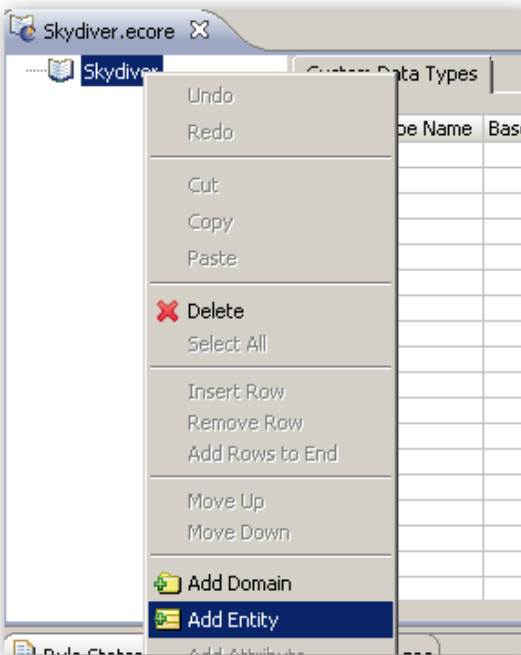
This can also be named “Skydiver”



Your screen should now look like this



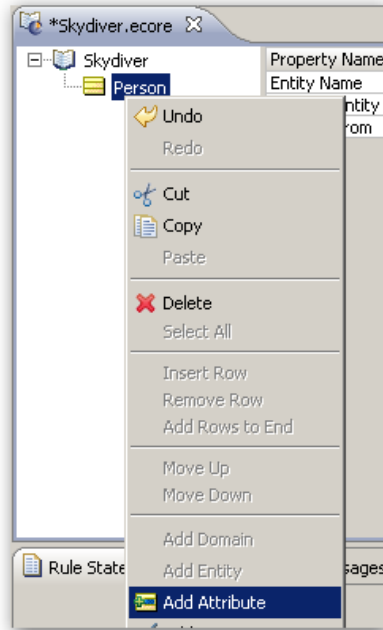
2. Right click “add Entity” (on the Skydiver in the Skydiver.ecore window)



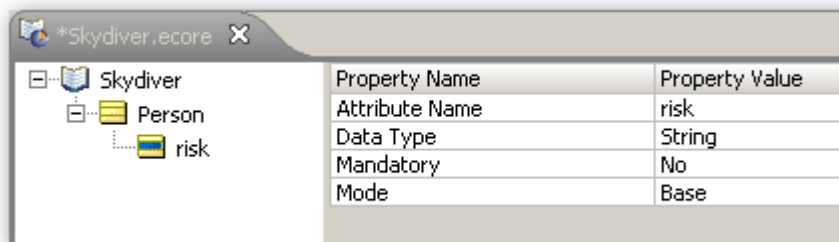
3. Change its name from Entity_1 to Person



4. Now add an attribute to Person



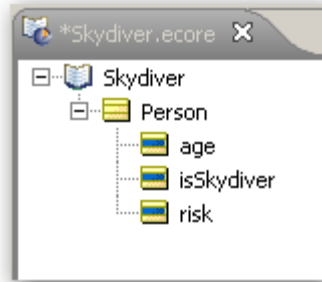
5. Name it “risk”. Leave its data type as string.



6. Create two more attributes:

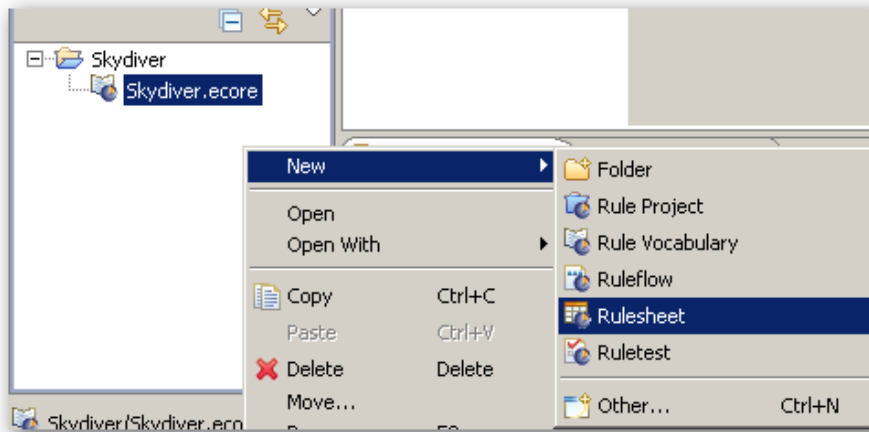
- Age of type “integer”, mandatory = “yes”
- isSkydiver of type Boolean, mandatory = “yes”

Your screen should look like this:

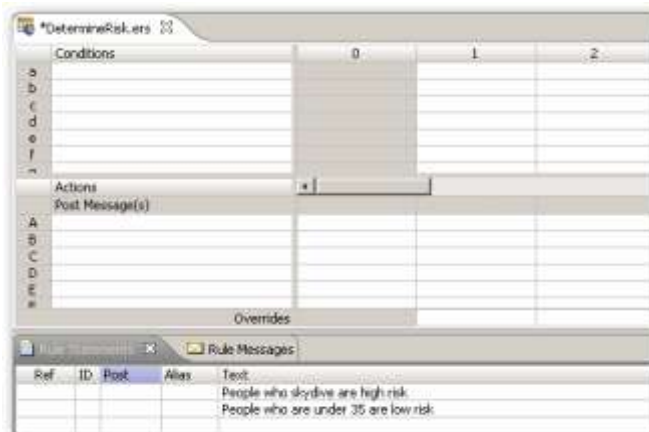


Model the Rules

1. Now create a new rule sheet called “DetermineRisk”

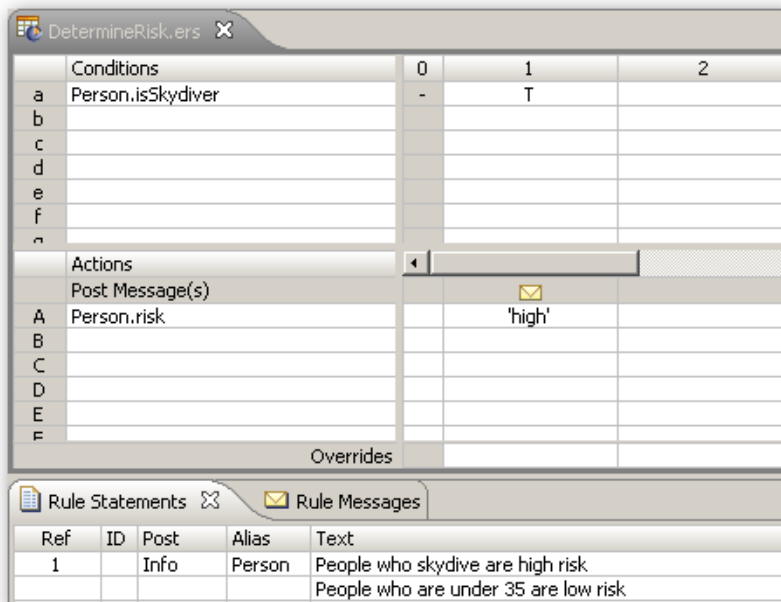


2. Type (or copy) these two rule statements into the TEXT field in the RULE STATEMENT tab at the bottom
 - a. Skydivers are high risk
 - b. People under 35 are low risk



3. Drag the “isSkydiver” attribute from the rule vocabulary into condition (a)
4. Select T (short for true) from the drop down in cell a1
5. Drag the “risk” attribute to Actions (A)
6. Type high in cell (A1) (no quotes – they will be added automatically)
7. Type 1 in the Ref cell of the first rule statement
8. Select “Info” from the Post dropdown
9. Select “Person” from the Alias dropdown

It should look like this:



10. Drag age to condition (b)
11. Type <35 in cell (b2)
12. Type low in cell (A2)
13. Update the reference to the rule statement

Conditions		0	1	2
a	Person.isSkydiver	-	T	
b	Person.age			< 35
c				
d				

Actions		0	1	2
Post Message(s)			✉	✉
A	Person.risk		'high'	'low'
B				
C				

Ref	ID	Post	Alias	Text
1		Info	Person	People who skydive are high risk
2		Info	Person	People who are under 35 are low risk

14. Save the rule sheet

Test the Rules

1. Create a new rule test named "Test"

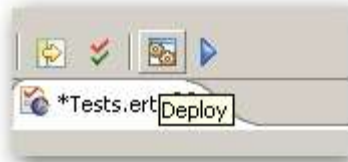
Initially it will look like this

Input	Output	Expected

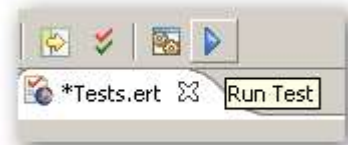
2. Drag a Person to the input column and expected column
3. Delete the risk attribute from the input (the rules will figure out its value)
4. Set the values as follows

Input	Output	Expected
<ul style="list-style-type: none"> Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] 		<ul style="list-style-type: none"> Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] risk [low]

5. Click the "Deploy" button to compile the rules



6. Click the “Run Test” button to run the rules



You should see this result:

Input	Output	Expected
Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] 	Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] risk [low] 	Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] risk [low]

Severity	Message	Entity
Info	People who are under 35 are low risk	Person[1]

7. Now add some more persons with the following expected results:

- age = 36, skydiver = true, (Expected risk= high)
- age = 24, skydiver = true, (Expected risk = high)
- age = 36, skydiver = false, (Expected risk = medium)

8. Run the rules and check the results:

Input	Output	Expected
<ul style="list-style-type: none"> Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] Person [2] <ul style="list-style-type: none"> age [36] isSkydiver [true] Person [3] <ul style="list-style-type: none"> age [24] isSkydiver [true] Person [4] <ul style="list-style-type: none"> age [36] isSkydiver [false] 	<ul style="list-style-type: none"> Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] risk [low] Person [2] <ul style="list-style-type: none"> age [36] isSkydiver [true] risk [high] Person [3] <ul style="list-style-type: none"> age [24] isSkydiver [true] risk [low] Person [4] <ul style="list-style-type: none"> age [36] isSkydiver [false] 	<ul style="list-style-type: none"> Person [1] <ul style="list-style-type: none"> age [24] isSkydiver [false] risk [low] Person [2] <ul style="list-style-type: none"> age [36] isSkydiver [true] risk [high] Person [3] <ul style="list-style-type: none"> age [24] isSkydiver [true] risk [high] Person [4] <ul style="list-style-type: none"> age [36] isSkydiver [false] risk [medium]

Notice that the output does not match our expected results for persons 3 and 4.

In the case of Person[3] the result given by the rules is risk = low while the expected result is risk = high (because the person is a skydiver they should be high risk)

In the case of Person[4] the rules failed to assign a value to risk.

What went wrong? Let's take a more detailed look

If you select Person[3] in the Output column you will see that it highlights the corresponding rule statements below:

Severity	Message	Entity
Info	People who skydive are high risk	Person[3]
Info	People who skydive are high risk	Person[2]
Info	People who are under 35 are low risk	Person[1]
Info	People who are under 35 are low risk	Person[3]

You can see that for Person[3] two rules were applied. The first rule determined the risk to be high because the person is a skydiver. However a second rule also applied and determined that the risk is low (because the person is under 35).

The effect of this is that our correct answer (high) gets overwritten by the incorrect answer (low).

This situation is referred to as a rule conflict or ambiguity and we'll need to look at our rules more closely to see what is happening.

In the case of Person[4] the rules simple failed to produce an output. This means that a rule for this situation is probably missing from our model.

Resolving the Problems

Here is our rule sheet again.

So what did we do wrong in modeling these rules?

	Conditions	0	1	2
a	Person.isSkydiver	-	T	-
b	Person.age	-	-	< 35
r				
	Actions			
	Post Message(s)		✉	✉
A	Person.riskRating		'High'	'Low'

Ref	Post	Alias	Text
1	Info	Person	Applicants who skydive have a High Risk rating.
2	Info	Person	Applicants less than 35 years of age have a Low Risk rating.

Well, nothing! We really did model the rules exactly as specified.

Unfortunately the rules as specified are ambiguous.

So how do we resolve this?

Most likely the intent of rule #2 was to apply only to non-skydivers – in fact this vital piece of information might well have been on a yellow post-it note on the business user's computer and not in the official rule documentation.

So first update the rule statement to reflect the correct rule:

Ref	Post	Alias	Text
1	Info	Person	Applicants who skydive have a High Risk rating.
2	Info	Person	Applicants less than 35 years of age have a Low Risk rating. (non skydiver)

Next update the rule sheet to correspond to the new rule #2:

Conditions	0	1	2
Person.isSkydiver		T	F
Person.age		-	< 35
Actions			
Post Message(s)		✉	✉
Person.risk		'High'	'Low'

Now, without having to rerun the test cases, recheck for ambiguity.

Corticon will confirm that there are no conflicts.

Next we need to address the missing rules. Click on the completeness button.

Corticon will add the missing conditions:

Conditions	0	1	2	3
Person.isSkydiver	-	T	F	F
Person.age	-	-	< 35	>= 35
Actions				
Post Message(s)		✉	✉	
Person.risk		'High'	'Low'	

Obviously it cannot add the action. The rule author needs to do that. So add Medium.

And since there are now three rules you should add a new rule statement that corresponds to the new rule.

Ref	Post	Alias	Text
1	Info	Person	Applicants who skydive have a High Risk rating.
2	Info	Person	Applicants less than 35 years of age have a Low Risk rating. (non skydiver)
3	Info	Person	Applicants 35 and over who don't skydive are medium risk

Now save the rule sheet and rerun the test case.

The actual output should now match the expected output:

The screenshot shows a testing interface with two side-by-side tree views labeled 'Output' and 'Expected'. Both trees show four person objects with their attributes and risk levels. Below the trees is a console window with a table of messages and entities.

Message	Entity
Applicants who skydive have a High Risk rating.	Person[3]
Applicants who skydive have a High Risk rating.	Person[1]
Applicants less than 35 years of age have a Low Risk rating. (non skydiver)	Person[2]
Applicants 35 and over who don't skydive are medium risk	Person[4]

What Happens when it gets More Complex?

Assuming we have resolved the ambiguities and incompleteness we might next ask “What happens when it gets more complex”

So let’s change rules 1, 2 and 3 and add three more:

1. Heavy Male Applicants who skydive have a High Risk rating.
2. Married female applicants less than 35 years of age who live in California have a Low Risk rating.
3. Any non-heavy person 35 and over who doesn't skydive is a medium risk
4. Red haired people are low risk but only in Arizona
5. Single skydivers are high risk
6. Divorced, widowed or separated Arizonans are medium risk (but only on Tuesdays during leap years)

Do we have any conflicts in these rules and under what circumstances?

Are there any missing rules?

Try figuring it out manually and then put the rules into Corticon and see how much easier it is. You will need to add some more attributes to the vocabulary to support these rules.

What’s wrong with having Boolean attributes for single, married, divorced, separate, widowed?

Try writing this in java. Or Drools. Or ABL

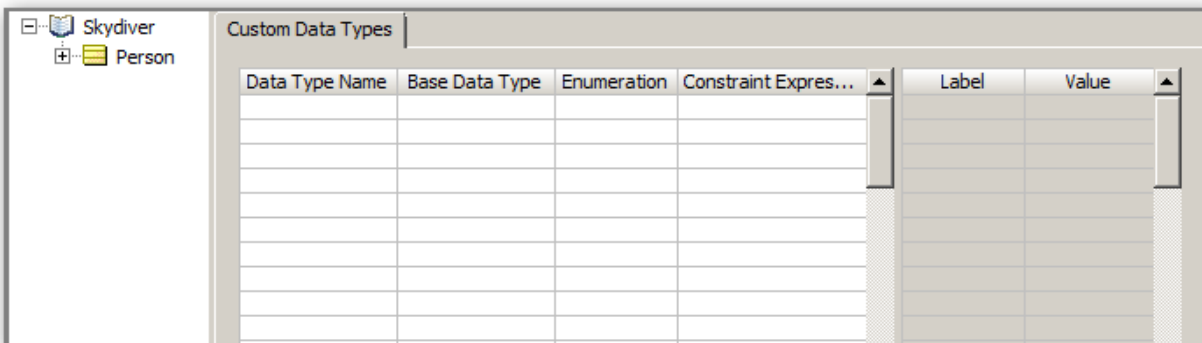
Conditions		1	2	3	4	5	6
a	Person.isHeavy	T	-	F	-	-	-
b	Person.gender	'male'	'female'	-	-	-	-
c	Person.isSkydiver	T	-	F	-	T	-
d	Person.maritalStatus	-	'married'	-	-	'single'	{'divorced', 'widowed', 'separated'}
e	Person.age	-	< 35	>= 35	-	-	-
f	Person.state	-	'CA'	-	'AZ'	-	'AZ'
g	Person.hairColor	-	-	-	'red'	-	-
h	today.dayOfWeek	-	-	-	-	-	3
i	Person.isLeapYear	-	-	-	-	-	T
Actions							
Post Message(s)							
A	Person.risk	'high'	'low'	'medium'	'low'	'high'	'medium'

Add Custom Data Types

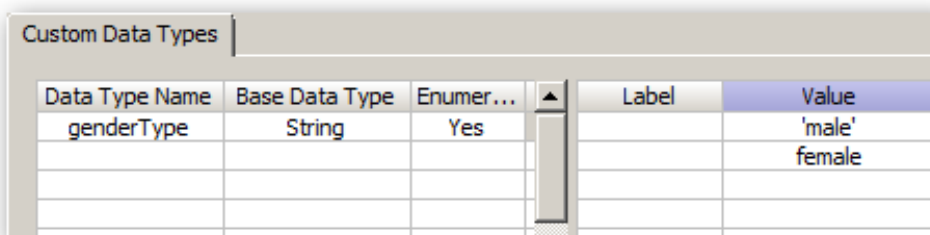
Up to this point you have been typing in the values used in the rule columns.

It's possible to predefine these values as custom data types.

To do this, open the vocabulary and select the top level "Skydiver"



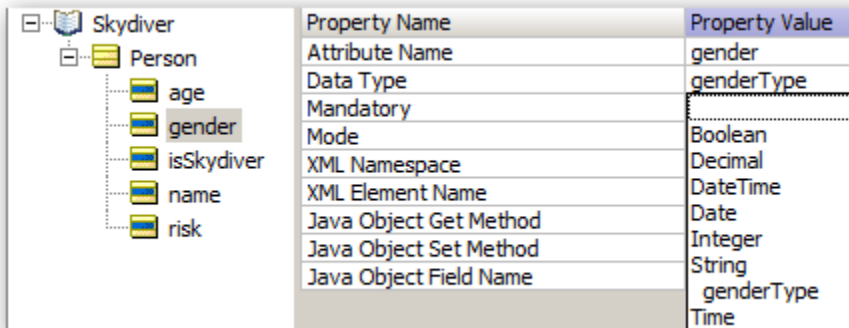
Enter a custom data type for gender as follows:



Do the same thing for any other attributes for which you want to have predefined lists.

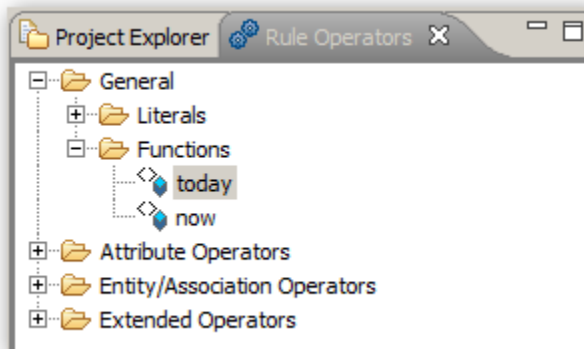
e.g. maritalStatus = {married, single, divorced, widowed, separated}

In order for a custom data type to be used you must select it in the attribute data type property:

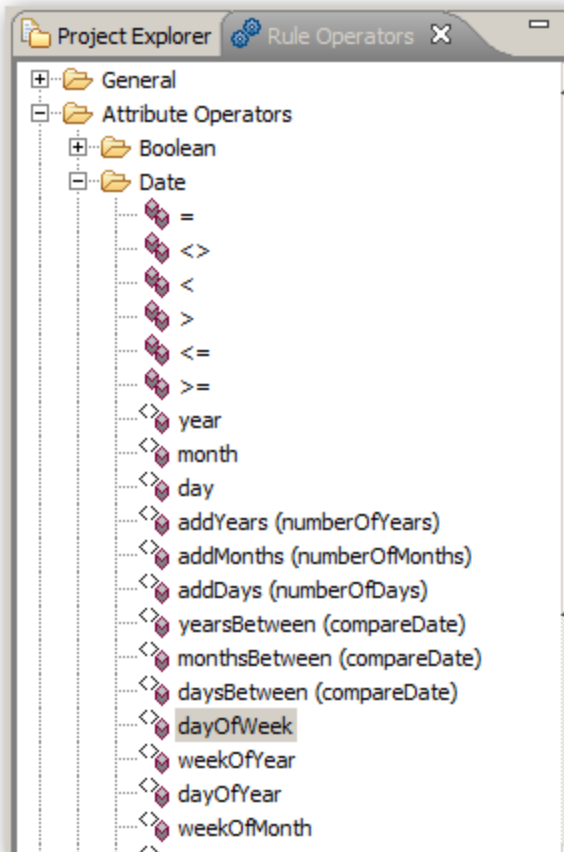


Date Operators

The condition in row h is constructed by dragging the “today” function from the Operator tab:



This gives us a date. Then we can drag the “dayOfWeek” operator from the list of data related operators:



Since there is no built-in function called “isLeapYear”, either that attribute needs to be passed in to the rules or we need to create a rule sheet that figures out if the current date is a leap year.

Such a rule sheet might look like this:

Conditions	1	2	3	4
Calculation.dateYear.mod (4) = 0	F	T	T	T
Calculation.dateYear.mod (100) = 0	-	F	T	T
Calculation.dateYear.mod (400) = 0	-	-	F	T
Actions				
Post Message(s)	✉	✉	✉	✉
Calculation.isLeapYear	F	T	F	T

Rules for Invalid or Unrecognized Values.

If we cannot be sure that the calling application will pass in valid data we may wish to add a validation rule sheet such as this

	Conditions	17	18	19	20
a	Person.isHeavy	-	-	-	-
b	Person.gender	-	other	-	-
c	Person.isSkydiver	-	-	-	-
d	Person.maritalStatus	other	-	-	-
e	Person.age	-	-	-	-
f	Person.state	-	-	other	-
g	Person.hairColor	-	-	-	other
h	today.dayOfWeek	-	-	-	-
i	Person.isLeapYear	-	-	-	-

other (not in quotes) means any value that is not defined in the custom data type associated with this attribute.

The Missing Rules

	Conditions	7	8	9	10	11	12	13	14	15	16
a	Person.isHeavy	T	T	T	T	F	F	F	F	F	F
b	Person.gender	'female'	'female'	'female'	'male'	'female'	'female'	'male'	'male'	{'female', 'male'}	{'female', 'male'}
c	Person.isSkydiver	F	-	-	F	T	F	T	F	T	F
d	Person.maritalStatus	'single'	'married'	{'divorced', 'separated', 'widowed'}	{'divorced', 'married', 'separated', 'single', 'widowed'}	'married'	'single'	'married'	{'married', 'single'}	{'divorced', 'separated', 'widowed'}	{'divorced', 'separated', 'widowed'}
e	Person.age	-	>= 35	-	-	>= 35	< 35	-	< 35	-	< 35
f	Person.state	'CA'	'CA'	'CA'	'CA'	'CA'	'CA'	'CA'	'CA'	'CA'	'CA'
g	Person.hairColor	'red'	'red'	'red'	'red'	'red'	'red'	'red'	'red'	'red'	'red'
h	today.dayOfWeek	-	-	-	-	-	-	-	-	-	-
i	Person.isLeapYear	-	-	-	-	-	-	-	-	-	-
Actions											
Post Message(s)											
A	Person.risk	'medium'	'medium'	'medium'	'medium'	'low'	'low'	'low'	'low'	'low'	'low'

After Compression

Advanced Example

In the basic example the rules applied only to individuals. Even if we passed in many individuals the rules evaluated them independently. Corticon automatically does this so we didn't need to do anything special for that to happen.

1. Skydivers are high risk
 2. Anyone under 35 is low risk if they don't skydive
 3. Anyone 35 or over is medium risk if they don't skydive

Conditions	0	1	2	3
a Person.isSkydiver	-	T	F	F
b Person.age	-	-	< 35	>= 35
c				
Actions	<input type="text" value=""/>			
Post Message(s)		✉	✉	✉
A Person.risk		'High'	'Low'	'Medium'
o				

Let's suppose now that our skydivers have organized themselves into skydiving groups and the groups want to apply for insurance.

Now in addition to the rules that assess the risk of the individuals we will have some additional rules that assess the eligibility of the group as a whole.

Here are the rules:

G1: If there are fewer skydivers than non-skydivers then the group is INELIGIBLE

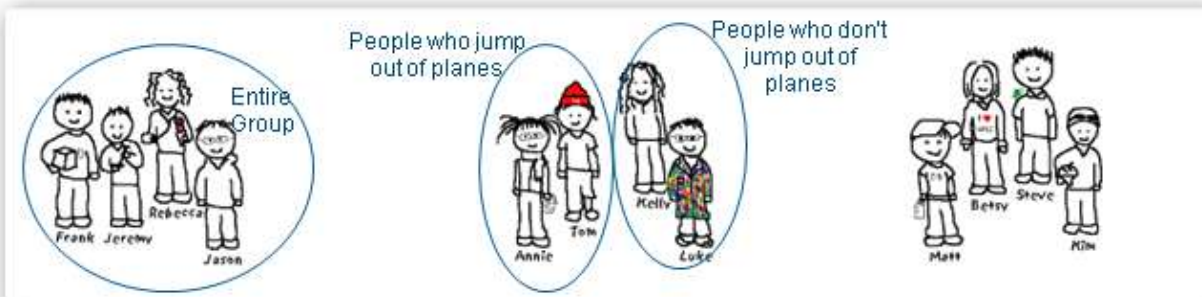
G2: If there are 10 or more members then the group is INELIGIBLE.

G3: If there is any skydiver over 65 then the group is INELIGIBLE

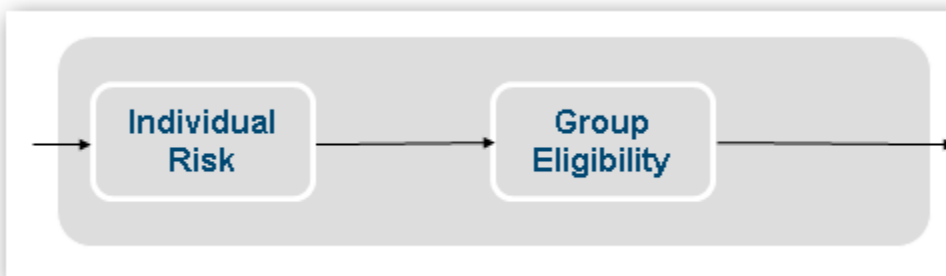
G4: If none of G1..G3 apply then Group is ELIGIBLE

These rules introduce some new concepts:

1. Skydivers – the people in a group that jump out of planes
2. Non-skydivers – the people in a group that do not jump out of places
3. Members – all the people that belong to a skydiving group



We'll also need a new rule sheet that evaluates the group. This new rule sheet will work in conjunction with the original rule sheet:

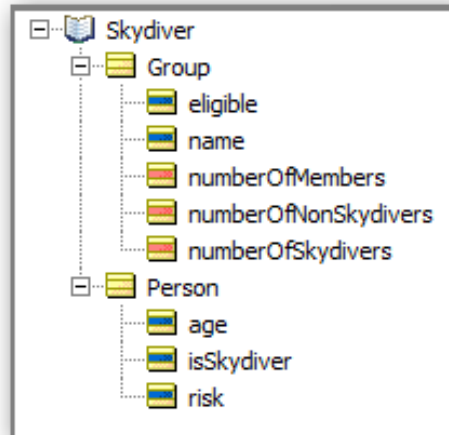


Since we now have a new business object to consider (the Group) we need to modify the vocabulary to model that object and its relationship to the individuals.

1. Create a new rule project called GroupSkydiver
2. COPY/PASTE ALL YOUR ORIGINAL BASIC SKYDIVER RULE MODEL FILES INTO THE NEW PROJECT
3. Open the new Vocabulary (double click on Skydiver.ecore)
4. Add Entity.
5. Name it "Group"
6. Add the following attributes:
 - name (string)
 - eligible (Boolean)

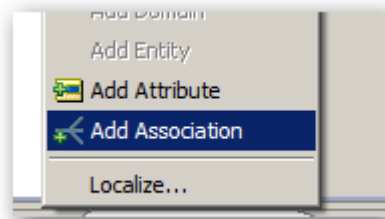
- numberOfSkydivers (integer, transient, mandatory)
- numberOfNonSkydivers (integer, transient, mandatory)
- numberOfMembers (integer, transient, mandatory)

TIP: once you have created the first integer attribute you can copy/paste to create the others. Your vocabulary should now look like this:



Establish the Relationship between Groups and Persons

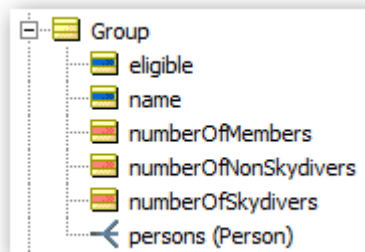
1. Right click on Group and select “add Association”



2. Enter the following

NOTE: add an “s” to the source-to-target name to indicate that there are many persons in a group. This is not required, but reads better. In fact you can choose any name you like for this.

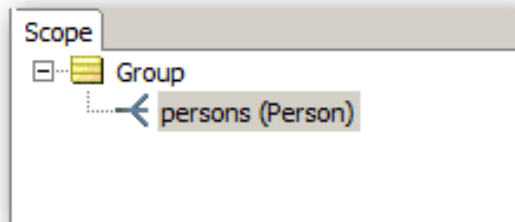
Now your vocabulary will look like this



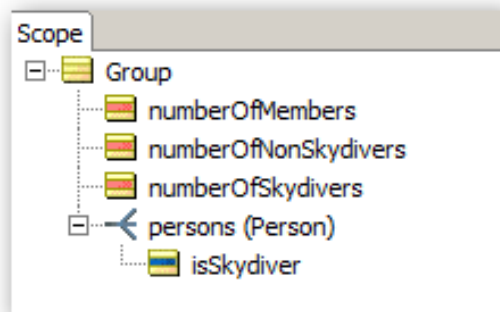
1. Save the vocabulary
2. Create a new rule sheet named “Calculations”
3. Open the advanced View



4. Expand Group in the Vocabulary Tab
5. Drag Group into the Scope section
6. Drag the embedded **persons** (not the free standing **Person**) and drop onto the Group in the Scope section. This is what tells Corticon that you want to work with the persons that are associated with each group:

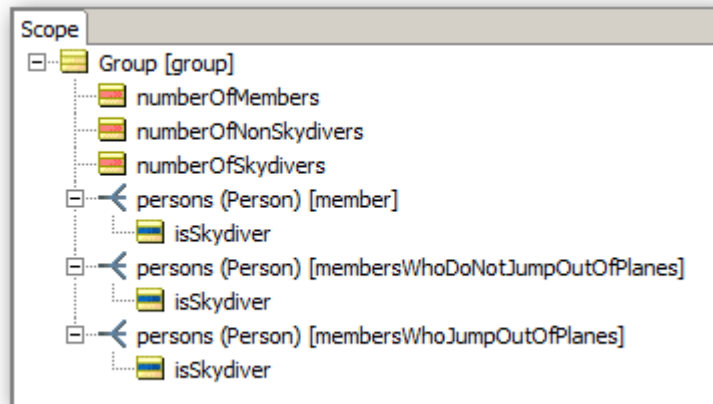


7. Now drag attributes as follows:



1. Double click on Group and enter **group** (lower case)
2. Double click on persons and enter **member** (lower case)
3. Drag person twice more and assign names **membersWhoJumpOutOfPlanes** and **membersWhoDoNotJumpOutOfPlanes**
4. Also drag the **isSkydiver** attribute to both

The Scope should now look like this:



What we have done here is to define three different ways (aliases) to refer to the persons that belong to a skydiving group.

At this point each of the aliases refers to ALL the members of the group. Simply naming an alias “membersWhoJumpOutOfPlanes” does not cause that alias to contain only skydivers. We have to tell Corticon how to determine the persons that are in that alias. This is done by using the filter section

5. In the filter section add these two statements (by dragging from the Scope)

Filters	
1	membersWhoDoNotJumpOutOfPlanes.isSkydiver = false
2	membersWhoJumpOutOfPlanes.isSkydiver = true
3	

This causes only those persons for whom isSkydiver=false to go into the membersWhoDoNotJumpOutOfPlanes collection; similarly those with isSkydiver=true go into the membersWhoJumpOutOfPlanes collection.

This now gives us a convenient way to refer to the two subsets of the group members.

6. Add the following to the Action section of the rule sheet.

Actions		
	Post Message(s)	
A	group.numberOfMembers = member->size	<input checked="" type="checkbox"/>
B	group.numberOfSkydivers = membersWhoJumpOutOfPlanes->size	<input checked="" type="checkbox"/>
C	group.numberOfNonSkydivers = membersWhoDoNotJumpOutOfPlanes->size	<input checked="" type="checkbox"/>

If you get an error make sure you have defined the attributes as type integer.

NOTE: The checkmarks are in column **zero** which means they are executed unconditionally (but not necessarily first or even in that order – Corticon will analyze the rules and determine the correct order of execution)

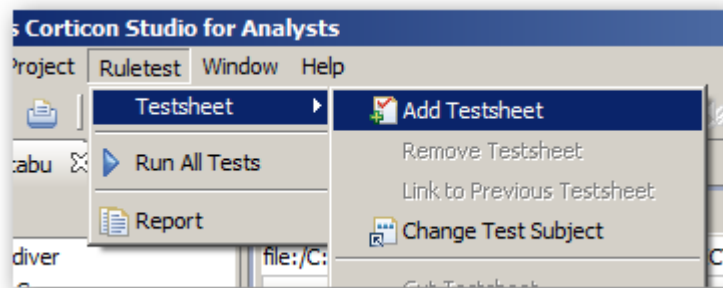
7. Save the rule sheet.

Create a Rule Flow

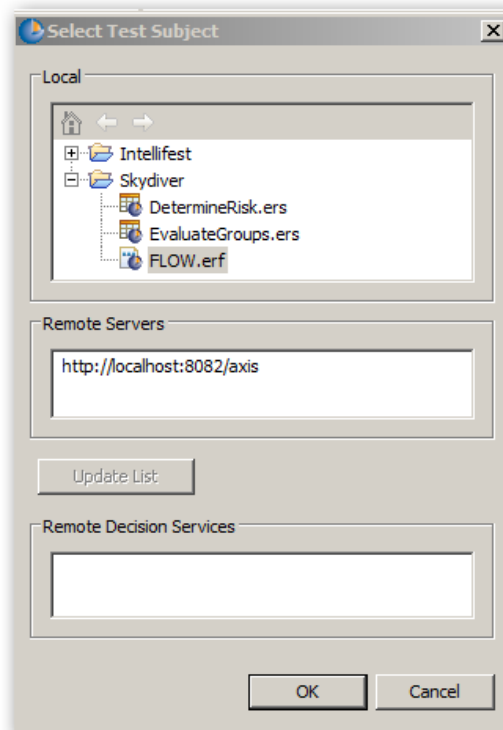
1. Create a new rule flow
2. Drag the Calculations rule sheet on to it.
3. Save it

Create a Test Sheet

1. Open the Test from earlier.
2. Add a test sheet



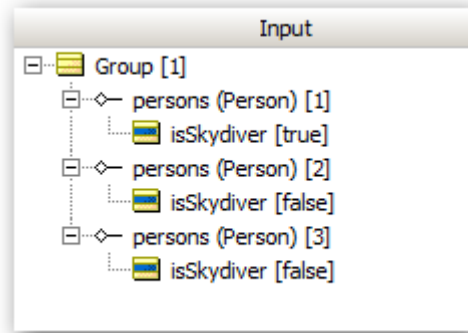
3. Choose the rule flow you just created:



NOTE: if the OK button is greyed out it may be because the rule flow does not use the same vocabulary as your test sheet.

4. Drag a group into the input.

5. Drag some persons the group (make sure you drag the embedded **persons** and not the entity)
6. Set the isSkydiver attribute for each person.
7. For now you can remove the other attributes:



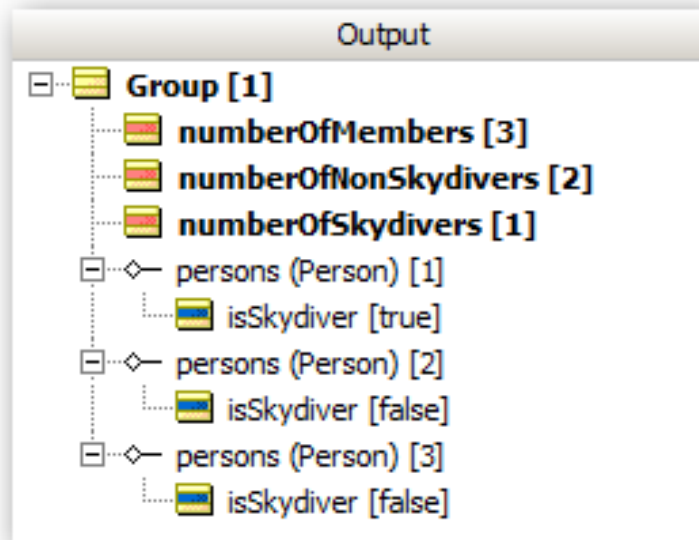
8. Now deploy the rules



9. And run the test



You should get something like this:



Add Some New Rules

Now create a new rule sheet called EvaluateGroup and add these rule statements to the rule sheet (you can copy/paste)

G1: If there are fewer skydivers than non skydivers then the group is INELIGIBLE

G2: If there are 10 or more members then the group is INELIGIBLE.

G3: If there is any skydiver over 65 then the group is INELIGIBLE

G4: If none of G1..G3 apply then Group is ELIGIBLE

Now we can easily build rules 1 and 2

For rule 3 we will write `membersWhoJumpOutOfPlanes->exists(age >65)`

Finally you can either add rule 4 manually, or use the completeness checker to add it.

The rule sheet should now look like this

Conditions		1	2	3	4
a	group.numberOfSkydivers < group.numberOfNonSkydivers	T	-	-	F
b	group.numberOfMembers	-	>= 10	-	< 10
c	membersWhoJumpOutOfPlanes->exists(age>65)	-	-	T	F
Actions					
Post Message(s)		✉	✉	✉	✉
A	group.eligible	F	F	F	T
Overrides					

Ref	Post	Alias	Text
1	Warning	group	G1: If there are fewer skydivers ({group.numberOfSkydivers}) than non skydivers ({group.numberOfNonSkydivers}) then the group is INELIGIBLE
2	Warning	group	G2: If there are 10 or more members then the group is INELIGIBLE. There are {group.numberOfMembers}.
3	Warning	group	G3: If there is any skydiver over 65 then the group is INELIGIBLE
4	Info	group	G4: Group {group.name} is eligible for insurance

Now save this and open the test sheet and add some more data to test that this rule sheet is working. You should add at a test case corresponding to each rule column.

file:/C:/Users/mparish/Desktop/ACTIVE PROJECTS/Intellifest 2012 San Diego/workspace/Skydiver/FLOW.erf

Input	Output
<ul style="list-style-type: none"> Group [1] <ul style="list-style-type: none"> persons (Person) [1] <ul style="list-style-type: none"> age [66] isSkydiver [true] persons (Person) [2] <ul style="list-style-type: none"> isSkydiver [false] persons (Person) [3] persons (Person) [4] persons (Person) [5] persons (Person) [6] persons (Person) [7] persons (Person) [8] persons (Person) [9] persons (Person) [10] persons (Person) [11] 	<ul style="list-style-type: none"> Group [1] <ul style="list-style-type: none"> eligible [false] numberOfMembers [11] numberOfNonSkydivers [10] numberOfSkydivers [1] persons (Person) [1] <ul style="list-style-type: none"> age [66] isSkydiver [true] persons (Person) [2] persons (Person) [3] persons (Person) [4] persons (Person) [5] persons (Person) [6] persons (Person) [7] persons (Person) [8] persons (Person) [9] persons (Person) [10] persons (Person) [11]

Severity	Message	Entity
Info	G3: If there is any skydiver over 65 then the group is INELIGIBLE	Group[1]
Info	G2: If there are 10 or more members then the group is INELIGIBLE.	Group[1]
Info	G1: If there are fewer skydivers than non skydivers then the group is INELIGIBLE	Group[1]

Make sure you also create at least one group that is eligible.

What happens if there are no skydivers or no non-skydivers?

Try setting minimum filters by right clicking the filter column

Filters	
1	membersWhoDoNotJumpOutOfPlanes.isSkydiver = false
2	membersWhoJumpOutOfPlanes.isSkydiver = true
3	

Precondition
Minimum Filter

Does it work now?

NOTE:

It is possible to combine the calculations and the rules in a single sheet.

The calculations would be entered as actions using column zero.

The screenshot shows the 'EvaluateGroups.ers' interface. It contains two main tables: 'Conditions' and 'Actions'.

Conditions		0	1	2	3	4
a	group.numberOfSkydivers < group.numberOfNonSkydivers	-	T	-	-	F
b	group.numberOfMembers	-	-	>= 10	-	< 10
c	membersWhoJumpOutOfPlanes->exists(age >65)	-	-	-	T	F
d						

Actions		0	1	2	3	4
Post Message(s)			☑	☑	☑	☑
A	group.numberOfMembers = member->size	☑				
B	group.numberOfSkydivers = membersWhoJumpOutOfPlanes->size	☑				
C	group.numberOfNonSkydivers = membersWhoDoNotJumpOutOfPlanes->size	☑				
D	group.eligible		F	F	F	T
Overrides						

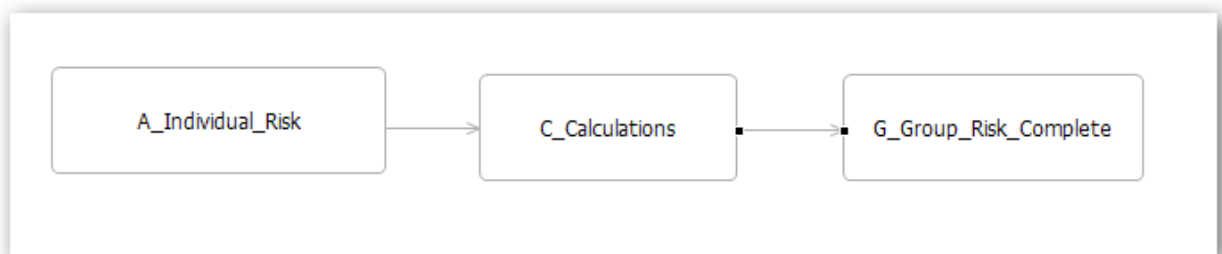
Ref	Post	Alias	Text
1	Info	group	G1: If there are fewer skydivers than non skydivers then the group is INELIGIBLE
2	Info	group	G2: If there are 10 or more members then the group is INELIGIBLE.
3	Info	group	G3: If there is any skydiver over 65 then the group is INELIGIBLE
4	Info	group	G4: If none of G1..G3 apply then Group is ELIGIBLE

If you have time try this out and observe that Corticon correctly figures out the appropriate order of execution of the combined rules.

However, it's probably easier to follow if they are separated.

Add the Individual Risk Rule Sheet to the Rule Flow

1. Now open the rule flow
2. Drag the "IndividualRisk" rule sheet
3. Connect it to the Calculations sheet:



4. Save it
5. Rerun the tests.

You will probably find some warning messages like these

Severity	Message	Entity
Info	People who skydive are high risk	Person[12]
Info	People who skydive are high risk	Person[13]
Info	People who skydive are high risk	Person[1]
Warning	Person.age is mandatory and must have a value. Some instances of Person have a null age which means some rules m...	Person[3]
Info	People who are under 35 are low risk	Person[12]
Warning	Person.age is mandatory and must have a value. Some instances of Person have a null age which means some rules m...	Person[2]
Info	G3: If there is any skydiver over 65 then the group is INELIGIBLE	Group[1]
Info	G1: If there are fewer skydivers than non skydivers then the group is INELIGIBLE	Group[1]
Info	G4: If none of G1..G3 apply then Group is ELIGIBLE	Group[2]

That's because some data is missing from your test cases.

Fill in the missing data and rerun the tests.

Now you should find you are getting messages for both Groups and Persons

file:/C:/Users/mparish/Desktop/ACTIVE PROJECTS/Intellifest 2012 San Diego/workspace/Skydiver/FLOW.erf

Input

- Group [1]
 - persons (Person) [1]
 - age [66]
 - isSkydiver [true]
 - persons (Person) [2]
 - age [22]
 - isSkydiver [false]
 - persons (Person) [3]
 - age [45]
 - isSkydiver [false]
- Group [2]
 - persons (Person) [12]
 - age [22]
 - isSkydiver [true]
 - persons (Person) [13]
 - age [64]
 - isSkydiver [true]

Output

- Group [1]
 - eligible [false]
 - numberOfMembers [3]
 - numberOfNonSkydivers [2]
 - numberOfSkydivers [1]
 - persons (Person) [1]
 - age [66]
 - isSkydiver [true]
 - risk [high]
 - persons (Person) [2]
 - age [22]
 - isSkydiver [false]
 - risk [low]
 - persons (Person) [3]
 - age [45]
 - isSkydiver [false]
 - risk [medium]
- Group [2]
 - eligible [true]
 - numberOfMembers [2]
 - numberOfNonSkydivers [0]
 - numberOfSkydivers [2]
 - persons (Person) [12]
 - age [22]
 - isSkydiver [true]
 - risk [high]

Rule Statements Rule Messages X

Severity	Message	Entity
Info	People who skydive are high risk	Person[12]
Info	People who skydive are high risk	Person[13]
Info	People who skydive are high risk	Person[1]
Info	People who are under 35 are low risk	Person[2]
Info	medium	Person[3]
Info	G3: If there is any skydiver over 65 then the group is INELIGIBLE	Group[1]
Info	G1: If there are fewer skydivers than non skydivers then the group is INELIGIBLE	Group[1]
Info	G4: If none of G1..G3 apply then Group is ELIGIBLE	Group[2]

If your messages are not showing up make sure that you have filled in the Ref, Post and Alias columns in the rule statement section.

If you are getting too many messages, make sure you are posting to the correct alias

Some things to Consider

What happens if there is a skydiver with no age?

What about a non-skydiver with no age?

What happens if we don't know whether they are a skydiver or not?

What about invalid values?

Another Enhancement

Add a rule (in a new rule sheet) that checks to see if the average age of the skydivers is greater than 60.

If so then set a flag that an additional premium is required.

Risk Scoring

Add rules to assign a numeric risk score to each individual (eg high=10, med=8, low=5)

Add a rule sheet to classify the entire group based on the average score of all the members

R1: VERY LOW: Average risk score ≤ 5

R2: LOW: Average risk score 5..8

R3: MEDIUM: Average risk score 8..12

R4: HIGH: Average risk score 12..15

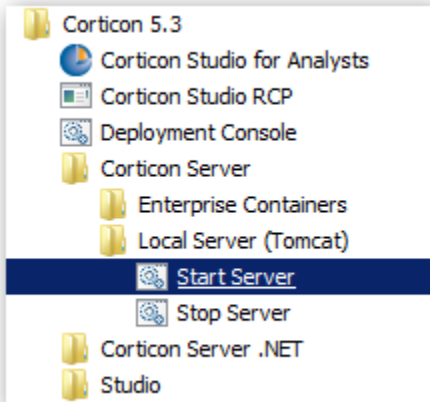
R5: VERY HIGH: Average risk score ≥ 15

How to Deploy Decision Services

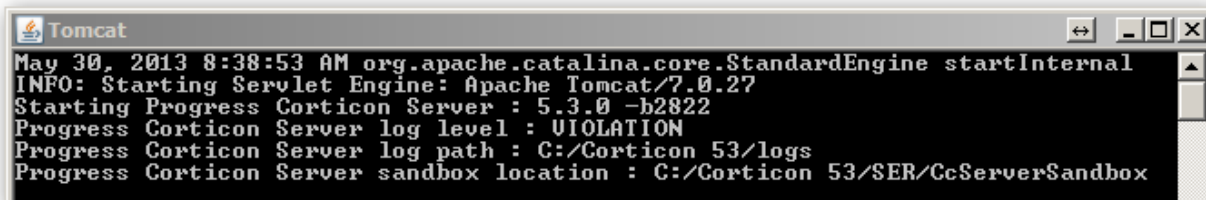
Install the Corticon Server

Install the Corticon Server from the USB drive.

Start the Corticon Server



Make sure the server is started. You should see a DOS window like this:



The license file for the default installer may have expired so copy the CcLicense.jar file from the USB drive to this directory:

Server\Tomcat\webapps\axis\WEB-INF\lib

The license for Studio is good for 90 days

The current server license will expire on 8/1/2013

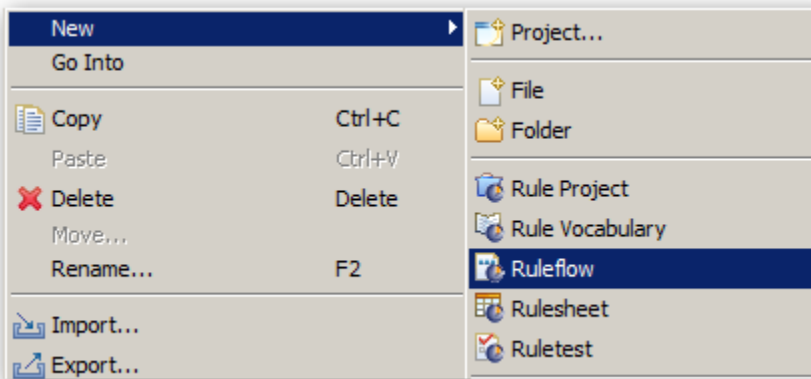
Restart the server to pick up the new license file

NOTE: You must start server once in order for the WEB-INF folder to be expanded from the installer WAR file.

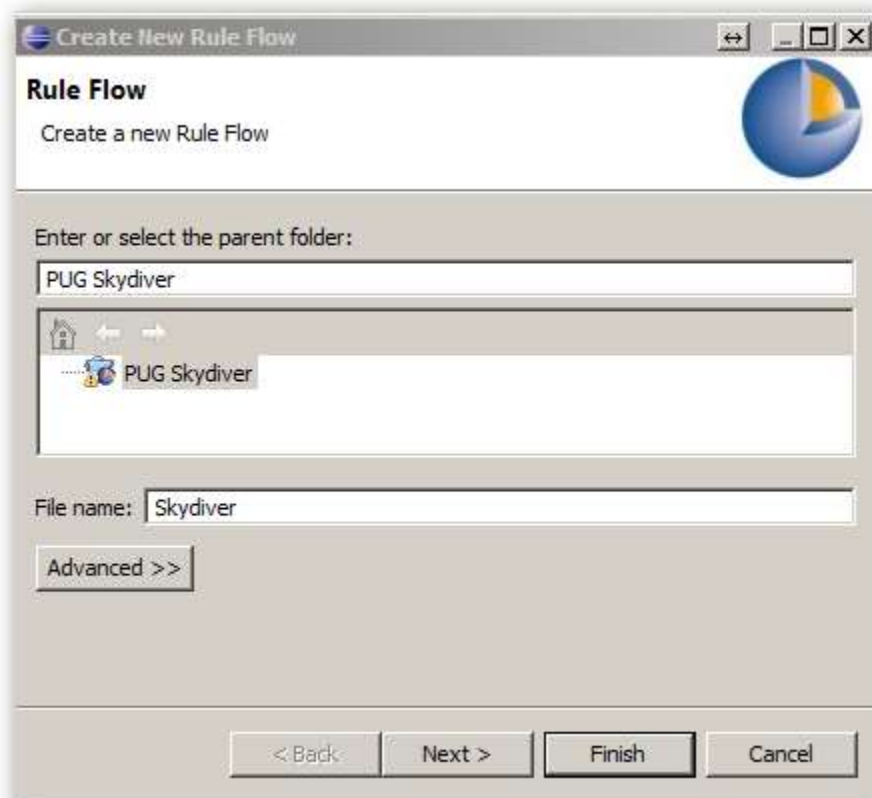
Configure the Decision Service

Let's take the basic Skydiver rule model that you created first and deploy it to the Corticon Server as a web based decision service

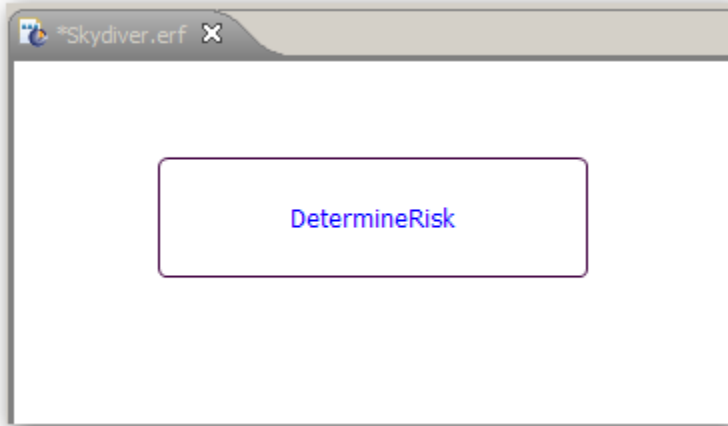
Create a rule flow



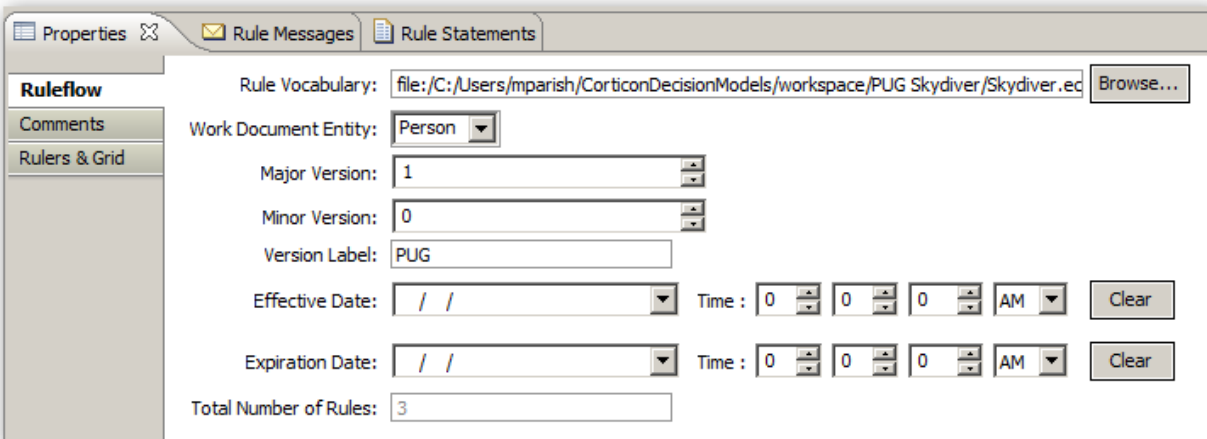
Name it Skydiver



Drag the rule sheet DetermineRisk onto the rule flow

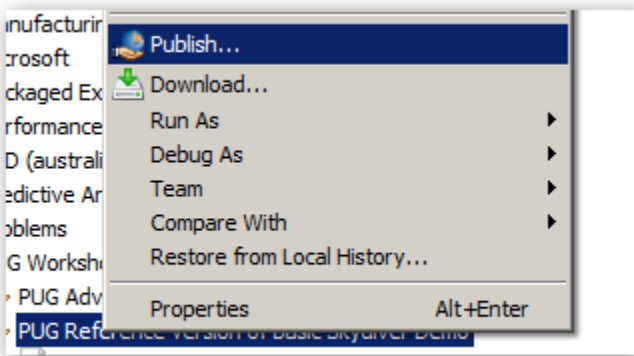


Edit the properties as follows



Publish the Decision Service

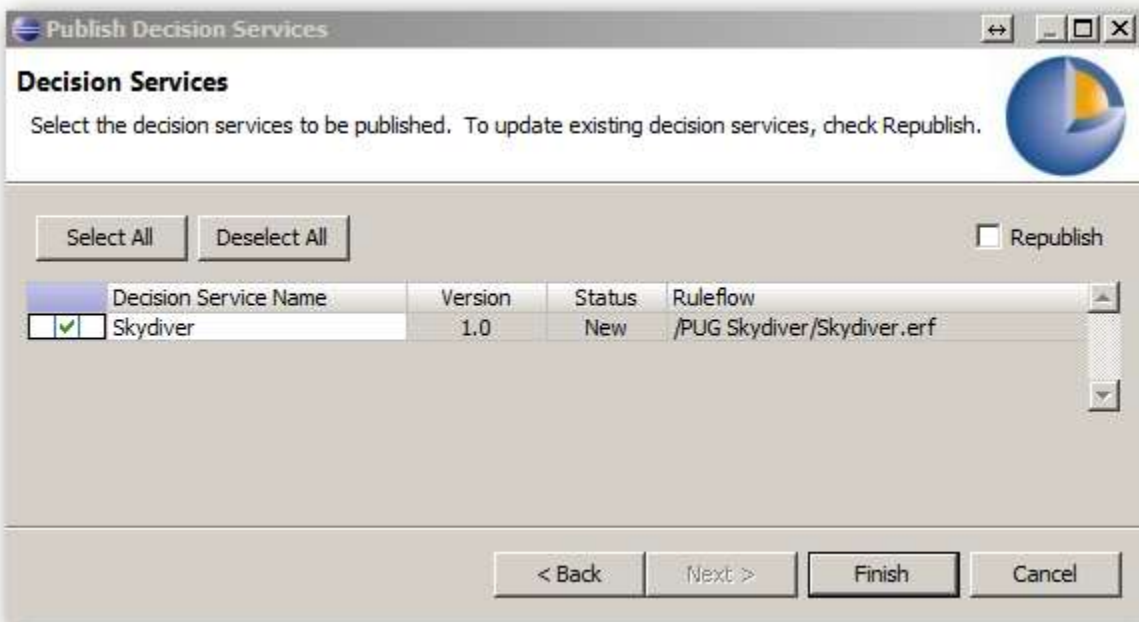
Select the Publish option by right clicking in the project explorer:



If necessary, select the server URL and enter admin/admin for the user and password.



Select the decision service from the list. You may also see your other decision services in this list.

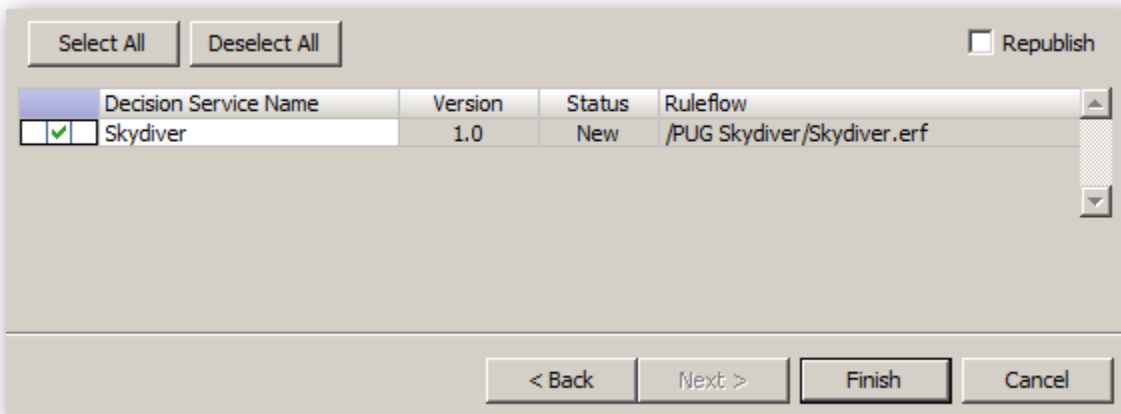


You can rename it here if you wish.

NOTE: The first time you deploy its status will show as “New”

	Decision Service Name	Version
	skydiverHealthplan	1.11
<input checked="" type="checkbox"/>	SkydiverPUG	5.7

When you republish an existing decision service you will see the status listed as “Update” and you will need to explicitly check the “Republish” button in order to activate the “Finish” button. This is a safety feature to prevent accidentally updating the wrong decision service.



TIP: If you have a lot of projects and rule models open at the same time, Corticon will search for and display ALL valid rule flows. This can take some time so you may want to open the project you are working on.

How to Monitor Decision Services

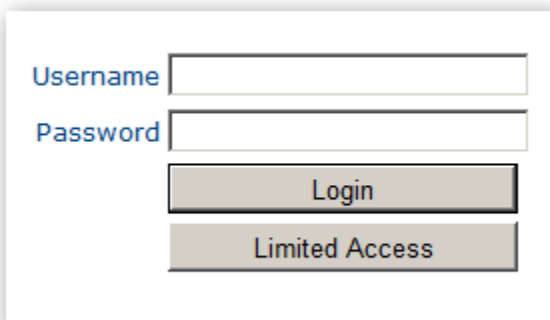
You can verify that your decision service was successfully deployed by starting the Corticon Web Console.

Start the Web Console

Enter this URL in Internet Explorer (other browsers may or may not work)

<http://localhost:8082/axis/>

Login using admin/admin



A login form with two input fields: "Username" and "Password". Below the fields are two buttons: "Login" and "Limited Access".

This will bring up the main menu:



- Decision Services
- Deploy Decision Service
- Configure Rules Server
- Monitor Rules Server
- Server API WSDLs
- Logout

Select "Decision Services" and you will see something like this

Business Rules Server Console -> Home
[Deployed Decision Services](#)

Service Name	Version	Live	Effective Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats
AllocateTrade	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear
Candidates	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear
ProcessOrder	1.10	<input checked="" type="checkbox"/>		Yes	No	0	0	Clear
Skydiver	1.0	<input checked="" type="checkbox"/>		No/Remove	No	0	0	Clear

Click on the Skydiver entry to bring up the Decision Service Details:

Business Rules Server Console -> Home -> Decision Services
[Decision Service Details](#)

Overview | Service Configuration | Rules Report | Test Execution

General Settings

Decision Service Name: Skydiver (version 1.0) (WSDL)

Deployed As Test Decision Service: No

Deployed EDS file: ./C91369968918950/Skydiver_v1_0.eds

Ruleflow URL: ./U31369968916022/Skydiver.erf (View)

Rulesheet URLs: DetermineRisk (View)

XML Message Style: Auto-detect

Dynamic Reload: No

Loaded from CDD file: No

Deployment Timestamp: May 30, 2013 6:46:23 PM

Ruleflow Timestamp: May 30, 2013 7:55:16 PM

EDS Timestamp: May 30, 2013 7:55:18 PM

Total Number of Rules Deployed: 3

Effective Date Start:

Effective Date End:

Total Execution Count: 1

Average Execution Time (ms): 4

Pool Settings

Minimum / Maximum Pool Size: 1 / 1

Available / Total Instances in Pool: 1 / 1

If you click the “view” hyperlink you can see the rules displayed in the browser window:

Rules					
Conditions		0	1	2	3
a	Person.isSkydiver	-	T	F	F
b	Person.age	-	-	< 35	>= 35
Actions					
A	Person.risk	-	High	Low	Medium
Overrides:		-	-	-	-

Rule Statements		
Reference	Severity	Text
1	Info	Applicants who skydive have a High Risk rating.
2	Info	Applicants less than 35 years of age have a Low Risk rating.
3	Info	Anyone 35 and over who doesn't skydive is a medium risk.

You can also switch to the business view that shows the natural language form of the conditions and actions:

Rules					
Conditions		0	1	2	3
a	Does the applicant jump out of planes?	-	T	F	F
b	How old is the applicant?	-	-	< 35	>= 35
Actions					
A	The applicant's risk rating is	-	High	Low	Medium
Overrides:		-	-	-	-

Selecting the “Create New Version” button will create a copy of the decision service that you can edit in the web interface. Any changes you make will not be accessible via web services call until you “Promote to Live” (a new button will appear)

Create New Version
Modify Current Version
Promote To Live
Download Rule Models

Overview
Service Configuration
Rules Report
Test Execution

General Settings

Decision Service Name:	SkydiverPUG (version 6.0) (WSDL)
Deployed As Test Decision Service:	Yes
Deployed EDS file:	./C31370916945883/SkydiverPUG_v6_0.eds
Ruleflow URL:	./TR01370916943055/RuleAssets/Applicant.erf (Edit)
Rulesheet URLs:	Applicant_COMPLETE (Edit)
XML Message Style:	Auto-detect

The new decision service will appear in the list as follows:

The new version will have its major version number incremented.

SkydiverPUG	5.7	<input checked="" type="checkbox"/>	Jan 1, 2013	No/Remove
SkydiverPUG	6.0		Jan 1, 2013	No/Remove

Notice the “Live” column is not checked.

“Modify Current Version” will increment the minor version number.

By clicking “No/Remove” (in the “Deployed from CDD” column) you can revert back to the prior version.

Decision services that were deployed as CDDs using the deployment console cannot be removed using the web interface. The CDD must be removed from the CDD folder.

Configure the Decision Service

Select Service Configuration and add Person.risk to the monitored attributes:

The screenshot shows the 'Service Configuration' tab in the Business Rules Server Console. It features several input fields and buttons:

- Current Rule Asset URL:** /C51369964783735/Skydiver_v1_0.eds
- Rule Asset URL:** An empty text box with a 'Browse...' button.
- Minimum Pool Size:** 1
- Maximum Pool Size:** 1
- XML Message Style:** Auto-detect (dropdown menu)
- Update** button
- No Monitored Attributes Registered** message
- Monitored Attribute:** Person.risk
- Analysis Bucket:** An empty text box
- Add** button

Make sure that you have these settings under “Decision Service Options” under the “Configure Rules Server” menu item.

The screenshot shows the 'Decision Service Options' tab in the Business Rules Server Console. It features several toggle switches and an 'Update' button:

- Decision Service Automatic Update Service:** On (dropdown menu)
- Decision Service Performance Monitoring Service:** On (dropdown menu)
- Decision Service Time Interval Performance Analysis Service:** On (dropdown menu)
- Decision Service Results Distribution Analysis Service:** On (dropdown menu)
- Update** button

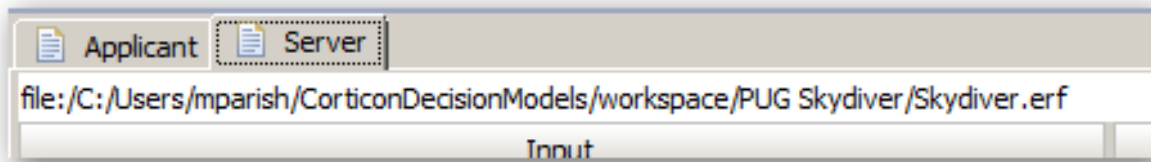
These settings are necessary if you want the server to track decision service performance.

How to Execute Decision Services

From Corticon Studio

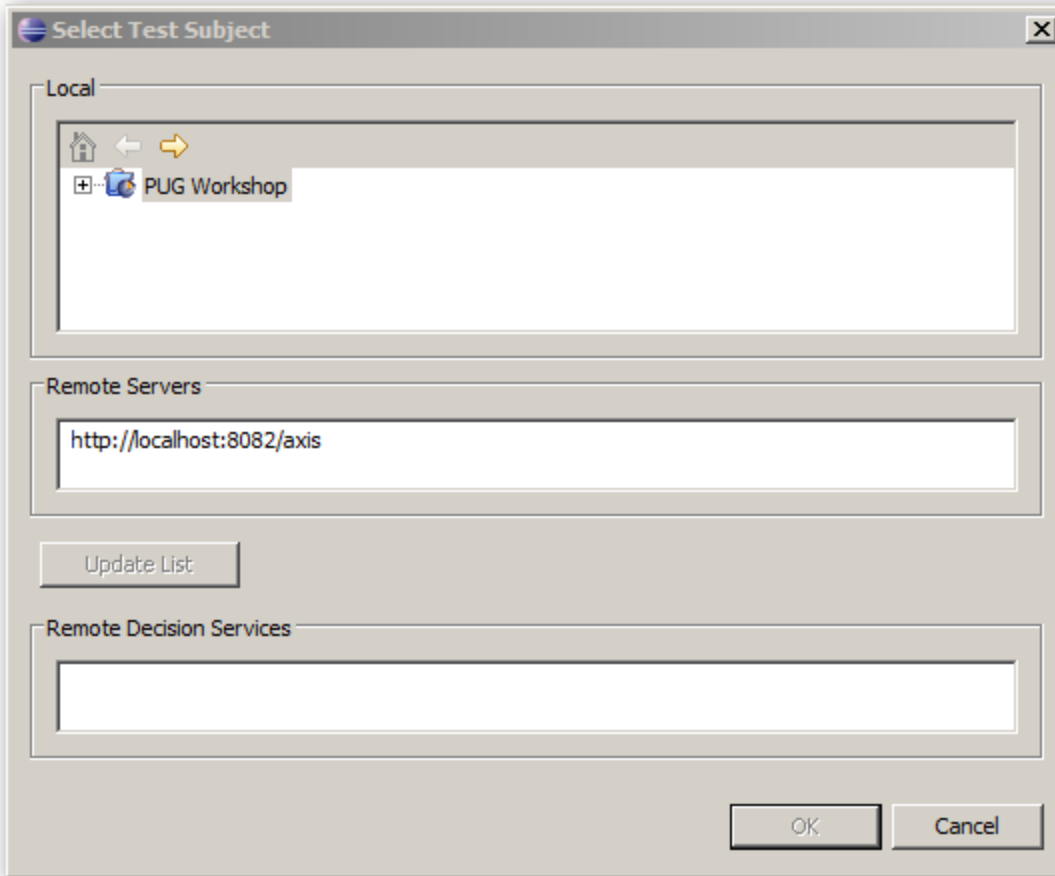
Now that the decision service has been deployed and configured we can execute it from the Corticon Studio Tester:

Start with an existing test case and make a copy named “Server”



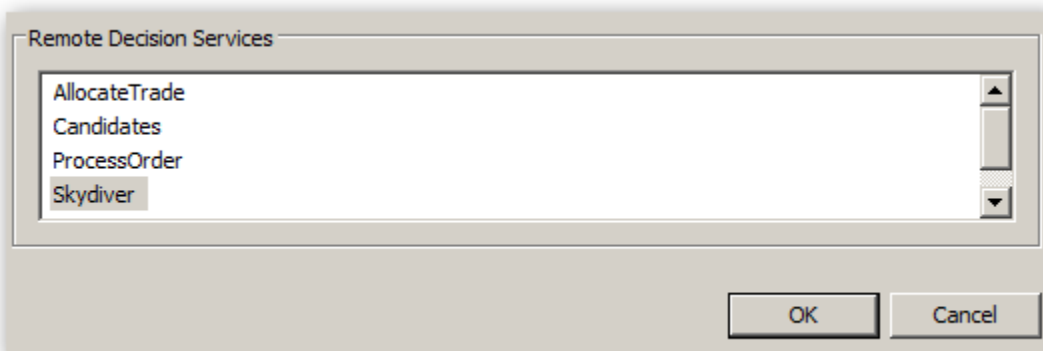
Double click on the file name (in this case the Skydiver.erf)

Select the appropriate server if more than one is listed.

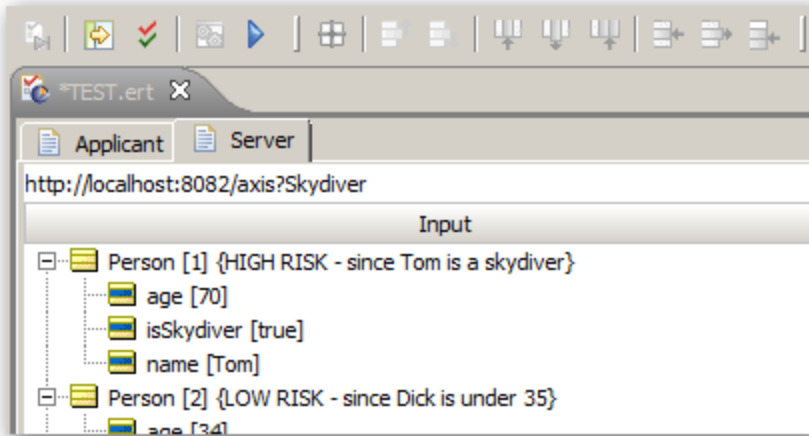


Then click the Update List button.

Then choose the correct decision service from the list:



Now your test case should look like this:



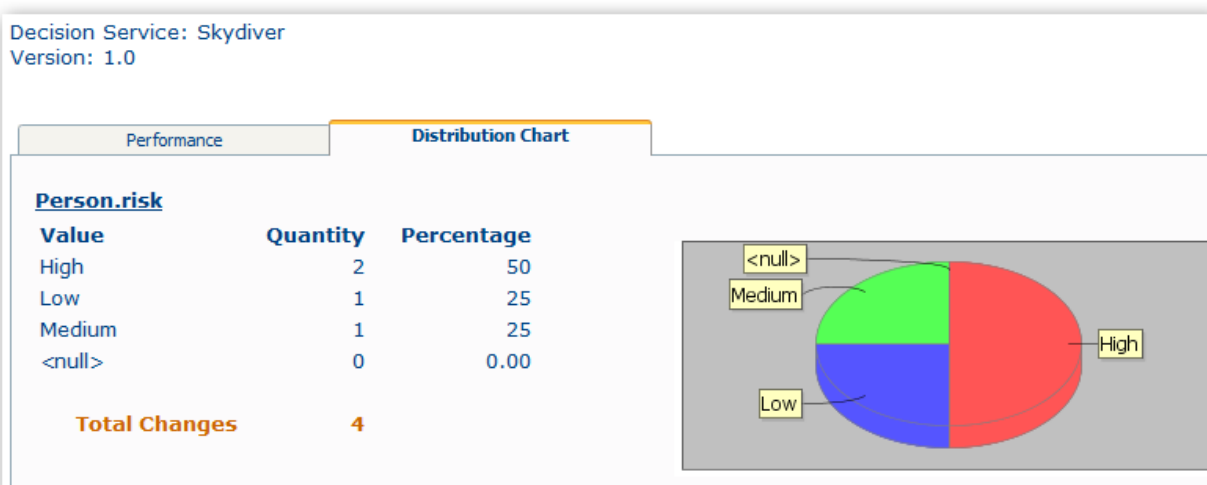
Now run the test (notice that the compile option is disabled since we are executing the already compiled version in the server)

Now go back the web console and refresh the list of decision services:

Service Name	Version	Live	Effective Expires	Deployed from CDD	Dynamic Reload	Executions	Avg Time (ms)	Clear Stats
AllocateTrade	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear
Candidates	1.14	<input checked="" type="checkbox"/>		Yes	Yes	0	0	Clear
ProcessOrder	1.10	<input checked="" type="checkbox"/>		Yes	No	0	0	Clear
Skydiver	1.0	<input checked="" type="checkbox"/>		No/Remove	No	1	4	Clear

You should see that the decision service called Skydiver has been executed once.

Click on the number 1 and look at the Distribution Chart



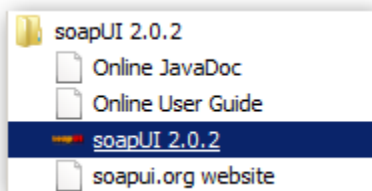
You can see that the single execution processed the four transactions that were in the test data input. In a similar fashion you can monitor any of the attributes used in the decision. For numeric attributes you can also choose to specify ranges of values for the pie chart rather than discrete values.

From SOAP UI

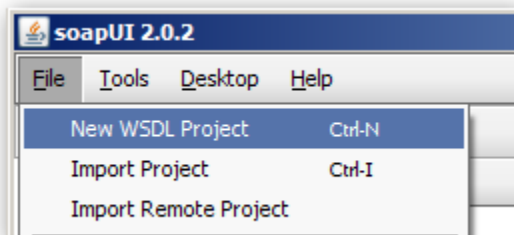
If you are familiar with SOAP UI then you can use that as a test client to invoke Corticon decision services.

Simply create a new project, import the Corticon WSDL, and fill in the decision service name and some data values.

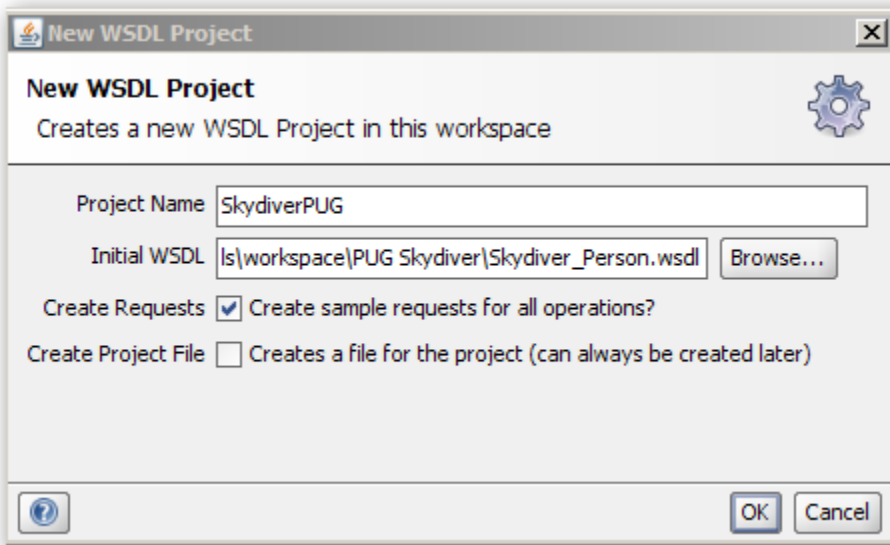
Start SOAP UI



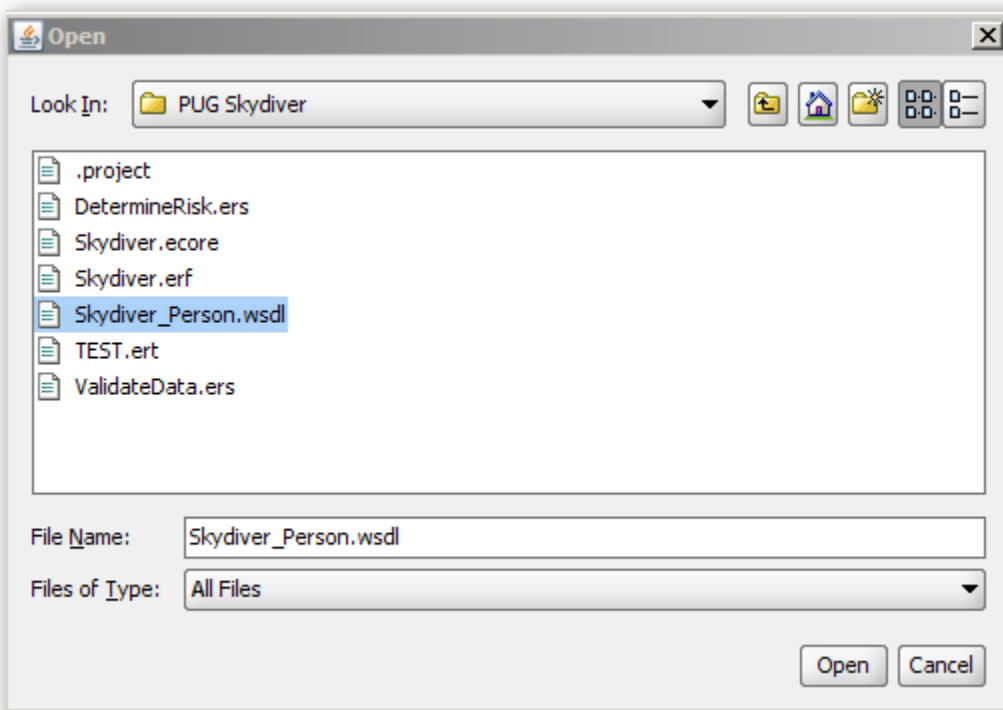
Create a new WSDL Project



Enter Details for the New Project

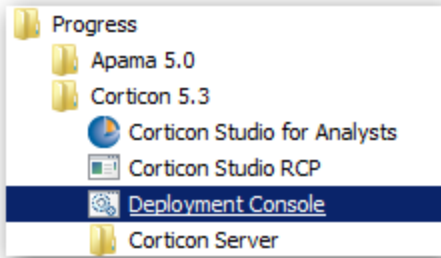


Locate the Initial WSDL

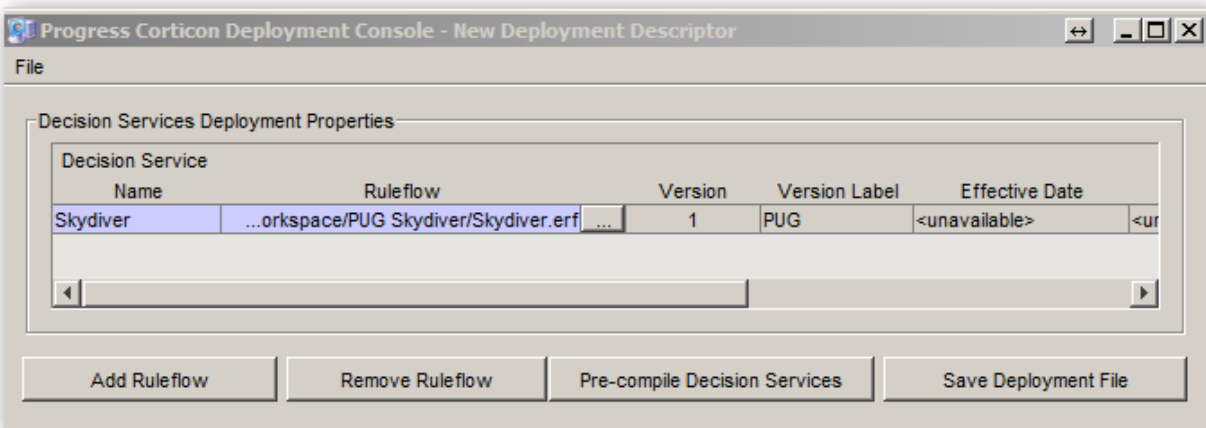


This will be in the project workspace.

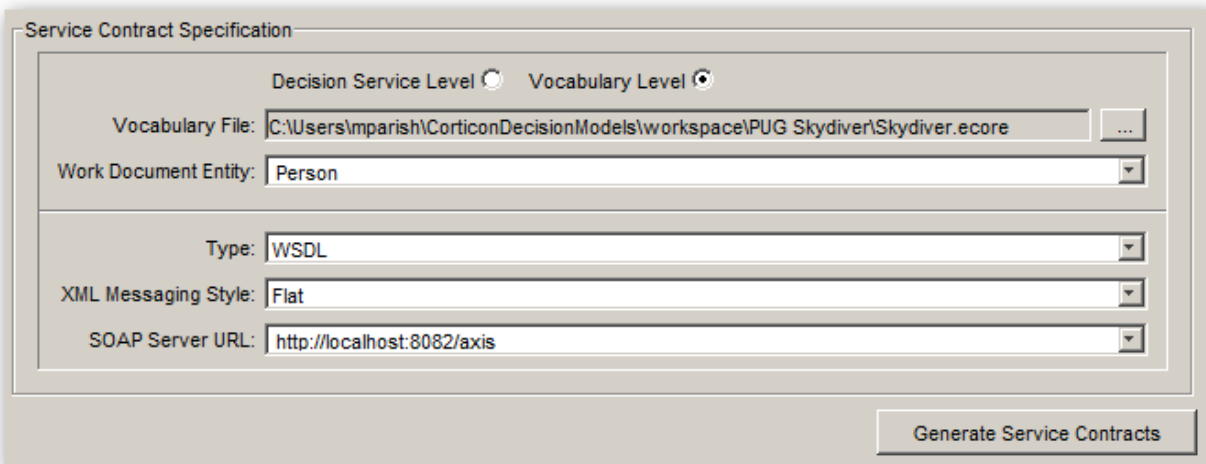
If it's not there then you will need to create it using the Corticon Deployment Console:



Select the rule flow in the Top Section



Choose WSDL in the bottom section



If necessary select the Work Document Entity "Person".

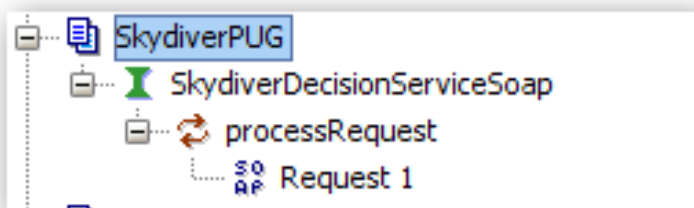
Ensure that "Vocabulary Level" is selected

This ensures that all entities and attributes are included in the WSDL even if they are not currently being used by the rules. In general it's better to choose this because then you will not need to recreate the WSDL if a rule author should start to use a new attribute.

However, if the vocabulary is very big and you are only using a few attributes then it may be more efficient to choose "Decision Service Level". In this case only the attributes actually used in rule sheets will be included in the WSDL.

Save the WSDL. If you are going to be integrating with OE then a good place to save it is in the project Workspace

This should show up in the SOAP UI project



You should see something like this:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <urn:CorticonRequest
      decisionServiceName=""
      decisionServiceTargetVersion=""
      decisionServiceEffectiveTimestamp=""
      usage="">
      <urn:WorkDocuments messageType="FLAT">
        <!--You have a CHOICE of the next 1 items at this level-->
        <urn:Person id="">
          <!--Optional:-->
          <urn:age?</urn:age>
          <!--Optional:-->
          <urn:isSkydiver?</urn:isSkydiver>
          <!--Optional:-->
          <urn:name?</urn:name>
          <!--Optional:-->
          <urn:risk?</urn:risk>
        </urn:Person>
      </urn:WorkDocuments>
    </urn:CorticonRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Create the Corticon Request block

Enter the DecisionServiceName as “Skydiver”

Remove the target version, effective date and usage

Later you can experiment with providing either the version number or the effective date – but not both.

Remove the comments (in black)

Remove the risk entry.

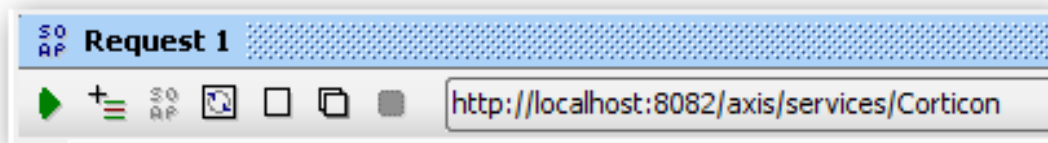
Enter values for the other fields

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <urn:CorticonRequest
      decisionServiceName="Skydiver">
      <urn:WorkDocuments messageType="FLAT">
        <urn:Person id="1">
          <urn:age>24</urn:age>
          <urn:isSkydiver>True</urn:isSkydiver>
          <urn:name>Tom</urn:name>
        </urn:Person>
      </urn:WorkDocuments>
    </urn:CorticonRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

This corresponds to what you entered in the Corticon Tester.

In fact you can generate this Request payload from the Tester

Now submit the request by clicking the green triangle:



You should get a result that looks like this:

```

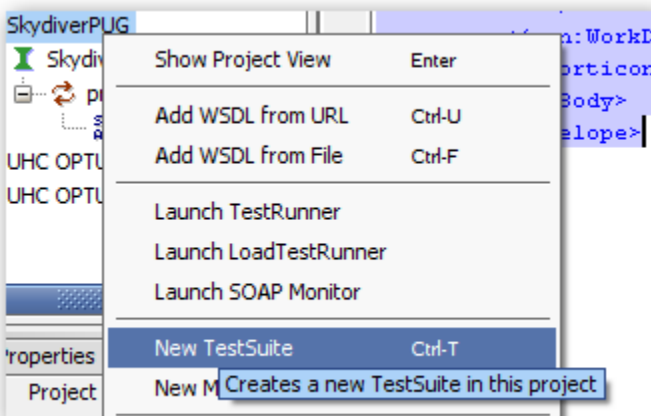
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd
  <soapenv:Body>
    <nsl:CorticonResponse decisionServiceName="Skydiver" xmlns:nsl="urn:decision:Sk
      <nsl:WorkDocuments messageType="FLAT">
        <nsl:Person id="1">
          <nsl:age>24</nsl:age>
          <nsl:isSkydiver>True</nsl:isSkydiver>
          <nsl:name>Tom</nsl:name>
          <nsl:risk>High</nsl:risk>
        </nsl:Person>
      </nsl:WorkDocuments>
      <nsl:Messages version="1.0">
        <nsl:Message>
          <nsl:severity>Info</nsl:severity>
          <nsl:text>Applicants who skydive have a High Risk rating.</nsl:text>
          <nsl:entityReference href="#1"/>
        </nsl:Message>
      </nsl:Messages>
    </nsl:CorticonResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

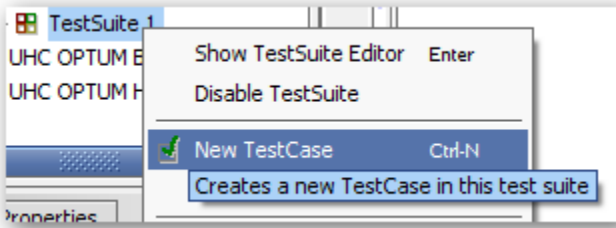
If you want, you can put multiple persons in the payload. Just make sure to give them unique ids so you can match up the resulting rule messages (using the href)

You can use SOAP UI to do load testing by putting this request in a continuous loop:

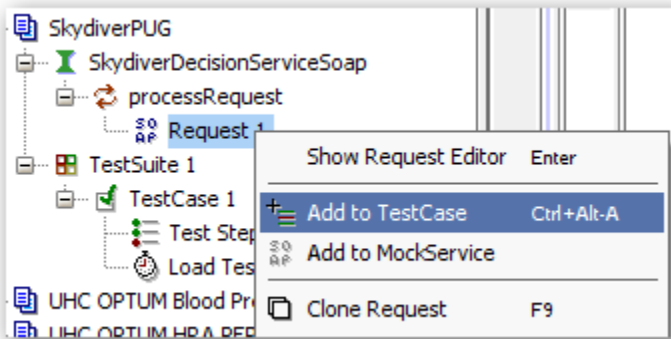
Create a new TestSuite



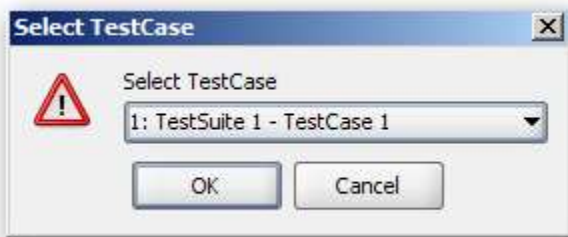
Create a new TestCase



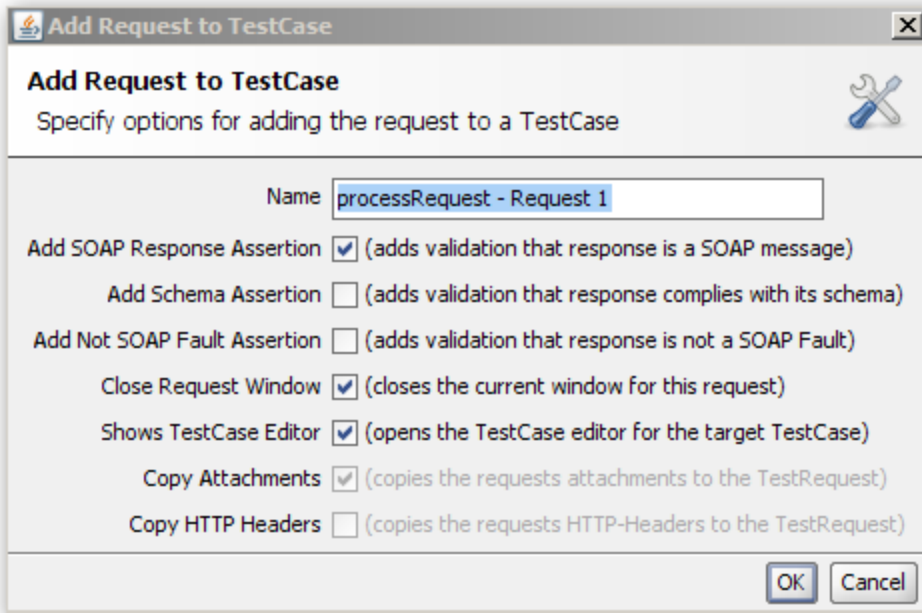
Add the request to the testcase



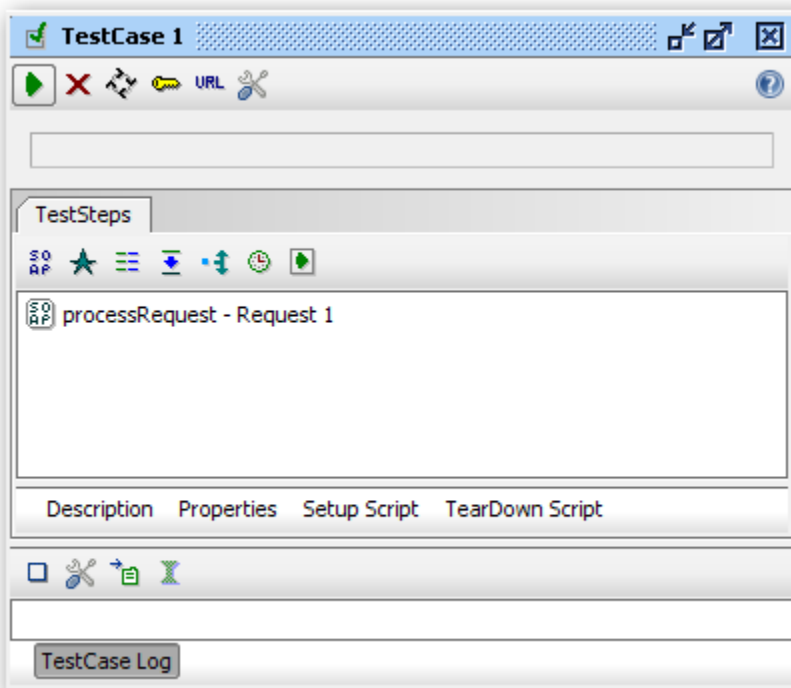
Choose the TestSuite and TestCase



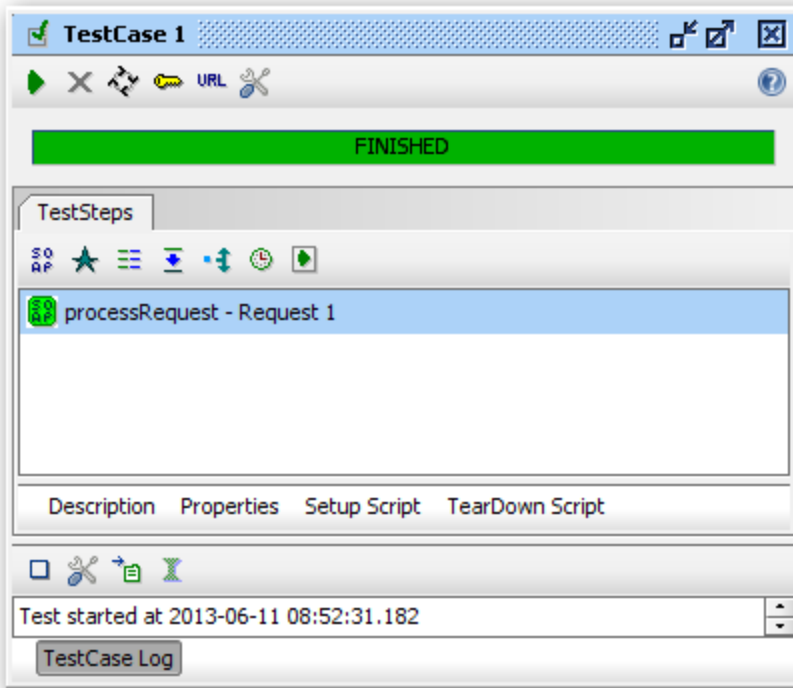
Use the defaults



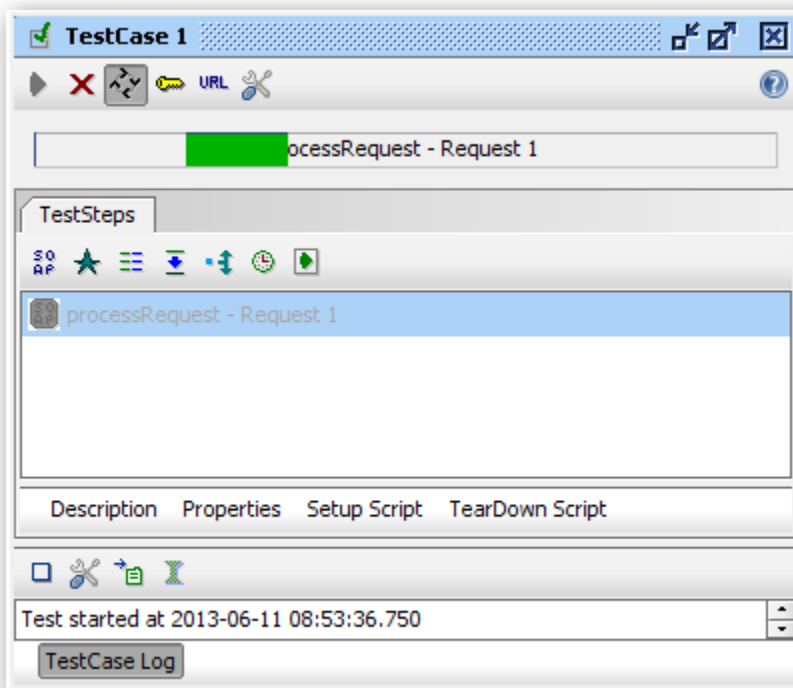
Should look like this:



Run once to test it:



Now set it to loop continuously and start it.

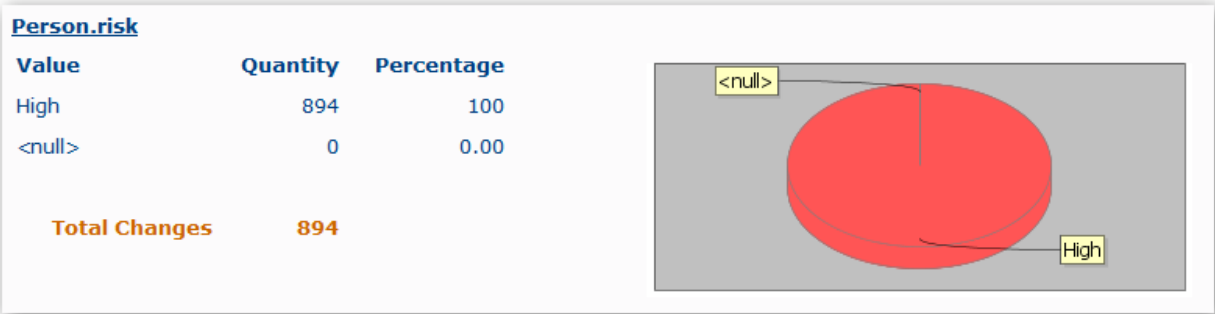


Let it run for about a minute and then click on the X to stop it (you might have to click a couple of times to get its attention)

Now if you look in the Web Console you will see something like:

Skydiver	1.0	<input checked="" type="checkbox"/>	No/Remove	No	894
----------	-----	-------------------------------------	-----------	----	-----

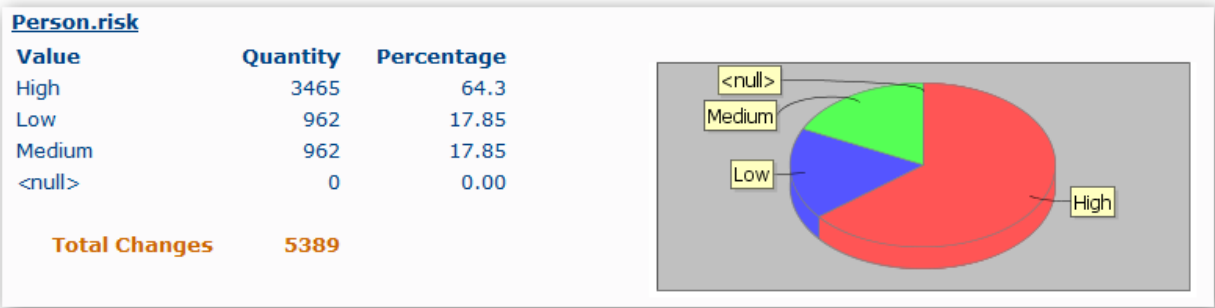
And



If you add some other test cases then the pie chart may be more interesting.

For example

```
<urn: Person id="1"><urn: age>24</urn: age><urn: isSkydiver>True</urn: isSkydiver><urn: name>Tom</urn: name></urn: Person>
<urn: Person id="2"><urn: age>24</urn: age><urn: isSkydiver>False</urn: isSkydiver><urn: name>Dick</urn: name></urn: Person>
<urn: Person id="3"><urn: age>34</urn: age><urn: isSkydiver>True</urn: isSkydiver><urn: name>Harry</urn: name></urn: Person>
<urn: Person id="4"><urn: age>44</urn: age><urn: isSkydiver>False</urn: isSkydiver><urn: name>Jane</urn: name></urn: Person>
```



How to Execute Decision Services from Open Edge

Basically this is accomplished by using OE to make web services calls to the Corticon engine just like calling any other standard web service

Essentially you will create datasets in OE that are the inputs to the Corticon request and get back the results in the Corticon response.

You will use ABL to populate the request, invoke the web service and extract the results from the response.

Since the structure of the request and response are pretty much the same, you can use a single dataset in OE for both purposes if you want.

In OE 11.2 and prior you will use the proenv.exe to generate ABL code from the decision service WSDL.

In OE 11.3 we provide some macros that can automate this process. In particular we have the ability to generate the Corticon vocabulary from the OE datasets.

For more detail see separate documents entitled

1. [Business_Rules_In_OpenEdge_With_Corticon \(pre-OE11.3\).pdf](#)
2. [OE 11.3-OECorticonIntegration-280513-0108-8.pdf](#)
3. [OE113BETA-Corticon End-to-End Workflow.pdf](#)
4. [OE113BETA-OpenEdge Business Rules White Paper.pdf](#)

Sample Projects

1. [Corticon PUG Skydiver Project.zip](#)
2. [OE PUG Skydiver Project.zip](#)
3. [SkyDiver-WebServicesPROC.zip \(combined Corticon and OE\)](#)

Create an Open Edge Project

1. Create a suitable User Interface to enter some data and display the results
2. Make sure the data elements correspond to those defined in the rule model
3. Create a submit button that will trigger the call to Corticon

Create the Corticon Decision Service

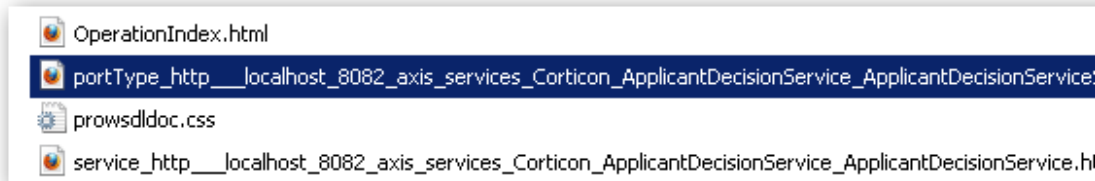
1. Create a decision service
2. Make sure the vocabulary matches the data in Open Edge
3. Deploy it to the server
4. Generate the WSDL (save to the Open Edge Project workspace)

Convert the WSDL into ABL

1. Start proenv in a command prompt
2. Execute the command bprowsddoc

```
proenv>bprowsddoc skydiver.wsdl
Documentation written to
file:///C:/OpenEdge/WRK/Skydiver/index.html
```

3. Open the generated html file



4. Copy the relevant sections into your ABL code (see Appendix for an example)
 - i. TEMP-TABLES
 - ii. DATASET

Add ABL code to control the GUI

1. Insert code for the submit button. For example

```
prepareCorticonRequest().
invokeCorticon().
processCorticonResponse().
```

2. Create the code for each of these procedures (see Appendix for examples)

Appendix

A Skydiver Client GUI in Open Edge

The screenshot shows a window titled "Skydiver Client Application (OE)". The window contains a form with the following elements:

- Name: Tom
- Age: 25
- Skydiver
- Submit button
- RiskRating: High

At the bottom of the window, there is a status bar with the text: ".Applicants who skydive have a High Risk rating."

The GUI is created by dragging and dropping and the code for it is generated automatically. You need to provide the code for the submit button.

ABL Code (Overall Structure)

```
/*-----  
File      : Skydiver Client Application  
Purpose   : Demonstrate how to call Corticon from OE  
Syntax    :  
Description : Simple Skydiver Client  
Author(s) : mparish  
Created    : Mon May 06 08:15:33 PDT 2013  
Notes     : 1. Make sure the namespace matches the WSDL  
           : 2. Make sure the WSDL is in the project workspace  
           : 3. Make sure you have the right WSDL for the decision by generating it from the  
             : Corticon Deployment Console  
           : 4. Make sure the decision service is deployed to the Corticon Server  
           : 5. Make sure the Corticon Server is running  
           : 6. Use the web console to determine if the decision service was executed  
           : 7. Make sure the decision service name is what you deployed  
           : 8. make sure to include CorticonResponse.decisionServiceTargetVersion = ?.  
             : or provide an effective date. Call fails without it.  
-----*/  
  
USING Progress.Lang.*.  
USING Progress.Windows.Form.  
  
CLASS SkydiverDataEntryCOPY INHERITS Form:  
  
    DEFINE PRIVATE VARIABLE age           AS System.Windows.Forms.TextBox NO-UNDO.  
    DEFINE PRIVATE VARIABLE ageLabel      AS System.Windows.Forms.Label NO-UNDO.  
    DEFINE PRIVATE VARIABLE ApplicantName AS System.Windows.Forms.TextBox NO-UNDO.  
    DEFINE PRIVATE VARIABLE components    AS System.ComponentModel.IContainer NO-UNDO.  
    DEFINE PRIVATE VARIABLE Explanation   AS System.Windows.Forms.TextBox NO-UNDO.  
    DEFINE PRIVATE VARIABLE nameLabel     AS System.Windows.Forms.Label NO-UNDO.  
    DEFINE PRIVATE VARIABLE Skydiver     AS System.Windows.Forms.CheckBox NO-UNDO.  
    DEFINE PRIVATE VARIABLE Result        AS System.Windows.Forms.TextBox NO-UNDO.  
    DEFINE PRIVATE VARIABLE label1       AS System.Windows.Forms.Label NO-UNDO.  
    DEFINE PRIVATE VARIABLE SubmitButton  AS System.Windows.Forms.Button NO-UNDO.  
  
    DEFINE TEMP-TABLE CorticonResponse NO-UNDO.  
    DEFINE TEMP-TABLE WorkDocuments NO-UNDO.  
    DEFINE TEMP-TABLE Person NO-UNDO.  
    DEFINE TEMP-TABLE Messages NO-UNDO.  
    DEFINE TEMP-TABLE Message1 NO-UNDO.  
    DEFINE TEMP-TABLE entityReference NO-UNDO.  
    DEFINE DATASET CorticonResponseDset .  
  
    CONSTRUCTOR PUBLIC SkydiverDataEntryCOPY ( ):.  
    METHOD PRIVATE VOID InitializeComponent( ):.  
  
    METHOD PRIVATE VOID SubmitButton_click( INPUT sender AS System.Object, INPUT e AS System.EventArgs ):.  
    METHOD PRIVATE VOID prepareCorticonRequest():.  
    METHOD PRIVATE VOID invokeCorticon():.  
    METHOD PRIVATE VOID processCorticonResponse():.  
    END METHOD. /* processCorticonResponse */.  
    DESTRUCTOR PUBLIC SkydiverDataEntryCOPY ( ):  
    END DESTRUCTOR.  
  
END CLASS.
```

The Submit Button

```
METHOD PRIVATE VOID SubmitButton_click( INPUT sender AS System.Object,
                                           INPUT e AS System.EventArgs ):

    prepareCorticonRequest().
    invokeCorticon().
    processCorticonResponse().
    RETURN.
```

PrepareCorticonRequest

```
METHOD PRIVATE VOID prepareCorticonRequest():
    /* Maps the OE GUI data elements to the Applicant data object */
    DATASET CorticonResponseDset:EMPTY-DATASET().
    CREATE CorticonResponse.
    ASSIGN
        CorticonResponse.decisionServiceName = 'Skydiver'.
        CorticonResponse.decisionServiceTargetVersion = ?. /* needs this */
    /* CorticonResponse.decisionServiceEffectiveTimestam =*/
    CREATE WorkDocuments.
    ASSIGN
        WorkDocuments.CorticonResponse_id = RECID(CorticonResponse).
    CREATE Person.
    ASSIGN
        Person.WorkDocuments_id = RECID(WorkDocuments)
        Person.name              = THIS-OBJECT:ApplicantName:Text
        Person.isSkydiver        = THIS-OBJECT:Skydiver:Checked
        Person.age               = INTEGER(THIS-OBJECT:age:Text)
    .
    RETURN.
END METHOD.
```

InvokeCorticon

```
METHOD PRIVATE VOID invokeCorticon():
    /* Invokes Corticon as a web service */
    DEFINE VARIABLE hWebService AS HANDLE NO-UNDO.
    DEFINE VARIABLE hSkydiverDecisionServiceSoap AS HANDLE NO-UNDO.
    CREATE SERVER hWebService.
    hWebService:CONNECT("-WSDL 'Skydiver_Person.wsdl'").
    RUN SkydiverDecisionServiceSoap SET hSkydiverDecisionServiceSoap
        ON hWebService.
    RUN processRequest IN hSkydiverDecisionServiceSoap(INPUT DATASET
        CorticonResponseDset, OUTPUT DATASET CorticonResponseDset).
    DELETE OBJECT hSkydiverDecisionServiceSoap.
    RETURN.
    CATCH oError AS Progress.Lang.Error:
        MESSAGE oError:GetMessage(1)
            VIEW-AS ALERT-BOX.
    END CATCH.
    FINALLY:
        hWebService:disconnect().
        DELETE OBJECT hWebService NO-ERROR.
    END FINALLY.
END METHOD.
```

processCorticonResponse

```
METHOD PRIVATE VOID processCorticonResponse():
/* Maps the Corticon response (Applicant object) back to the GUI elements in OE */
  DEFINE VARIABLE messageString AS CHARACTER INITIAL " " NO-UNDO.
  FOR FIRST Person:
    THIS-OBJECT:Result:Text = STRING(Person.risk).
  END.
  FOR EACH Message1:
    messageString = messageString + Message1.text1 + " ".
  END.
  THIS-OBJECT:Explanation:Text = messageString.

  RETURN.
END METHOD. /* processCorticonResponse */
```

Update this to match your data payload if necessary