# Evaluate a Site for Solar Energy Potential

Integrate data from external REST APIs to enrich your decision data

## First Launch of Corticon Studio

Workspaces, projects, and folders provide a way to organize rule-modeling components. A workspace is a ‘root’ directory that acts as a container for rule projects created in Corticon Studio. When you first launch Corticon Studio, you’ll be prompted for a workspace—this is different than the folders that were assigned during the Corticon installation wizard. A workspace is where all of the things created by you, the end user, are stored. The work directory—the one from the installation wizard—is where all of the things that Corticon itself needs to work as an application will live. If you choose to later uninstall Corticon, make sure to do so using the uninstall wizard in your start menu, not just deleted these folders.

You can feel free to leave the default workspace folder (it will be created automatically), or point to a different folder if you prefer.

## Rule Projects

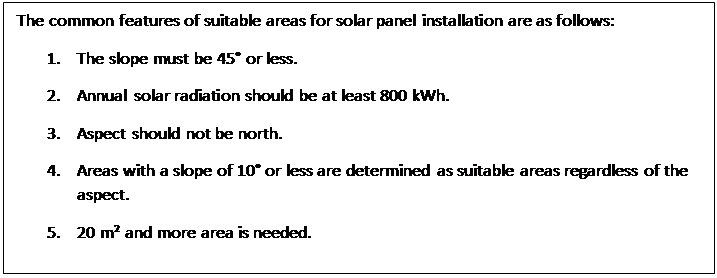
A Corticon rule project is a ‘container’ for rule-modeling files. A Corticon rule project can further contain folders to segregate rule-modeling components. A ‘template’ rule project can be downloaded from here—download the file, but don’t unzip it.

To import that template:

1. Click ‘File’ on the main toolbar, then click ‘Import…’
2. Choose the ‘Select archive file:’ option, then browse to the downloaded zip file
3. Click Finish

## Rule Vocabulary

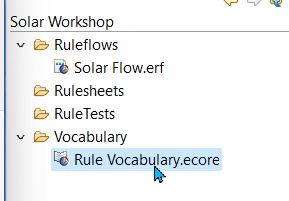
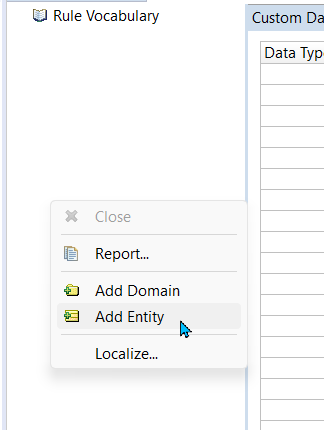
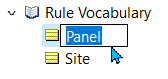
A Corticon Vocabulary is a rule-modeling component that enables you to define all the business terms that you require in your rules. For example, let’s take a look at how we might formulate a rule vocabulary from some common solar site evaluation rules.



An installation Site will have a size, annual radiation, and aspect. The Site will be where we install some number of Panels, each of which will have various values like length, width, and efficiency. Let’s define this is as a rule vocabulary, with Site and Panel being entities, each having their respective attributes, and associated to one another as one to many—one site to any number of panels.

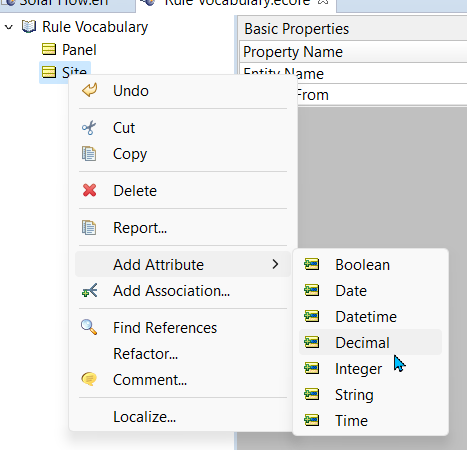
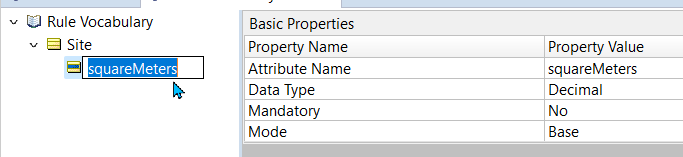
### Entities

Follow these steps to add an entity in the Vocabulary editor:

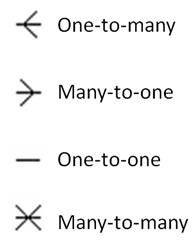
1. Double click the vocabulary file in the project explorer—it is in the Vocabulary folder and has the extension ‘.ecore’: 
2. In the Vocabulary editor, right-click and select Add Entity, and type in ‘Site’. If you don’t enter a name, it will default to ‘Entity\_1’, but it can be changed by double clicking on the name of the entity. 
3. Add one more attribute, and give it the name ‘Panel’: 

### Attributes

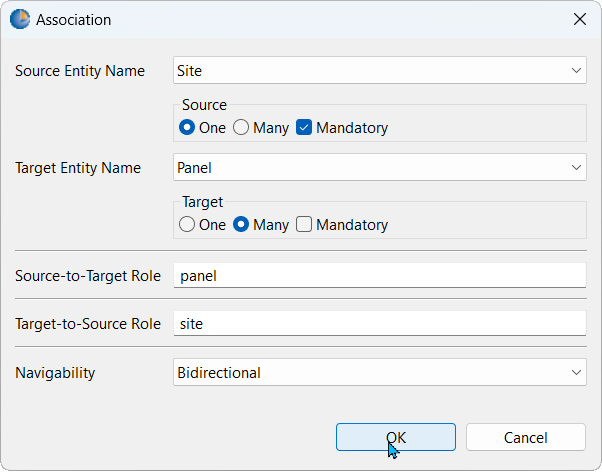
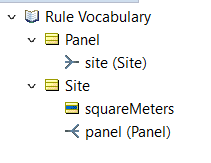
After you create an entity, you can add attributes to it. An attribute is like an adjective, whose values are populated at runtime.

1. Right-click the entity name and then select Add Attribute. 
2. You’re given a choice of the attributes’ data type—Boolean (True or False), Decimal, DateTime, Date, Integer, String, Time. For this example, select decimal. Enter a name, “squareMeters” for the attribute, or edit the name in the same way you did with the entity. After you add an attribute, the property editor panel of the Vocabulary editor displays all the properties for the attribute with default values. 

### Associations

An association describes the relationship between two entities. In a Corticon Vocabulary, you can define the association under either entity. By default, the association that you define appears under both entities in the Vocabulary tree. 

You add an association to an entity just as you add an attribute.

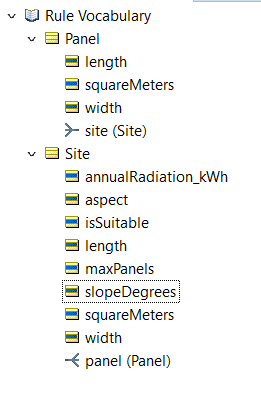
1. Right-click Site and select Add Association.
2. The default selections should be what we want—one site to many associations. Click OK.  Your vocabulary will now look like this: 

### Complete the Vocabulary

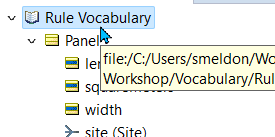
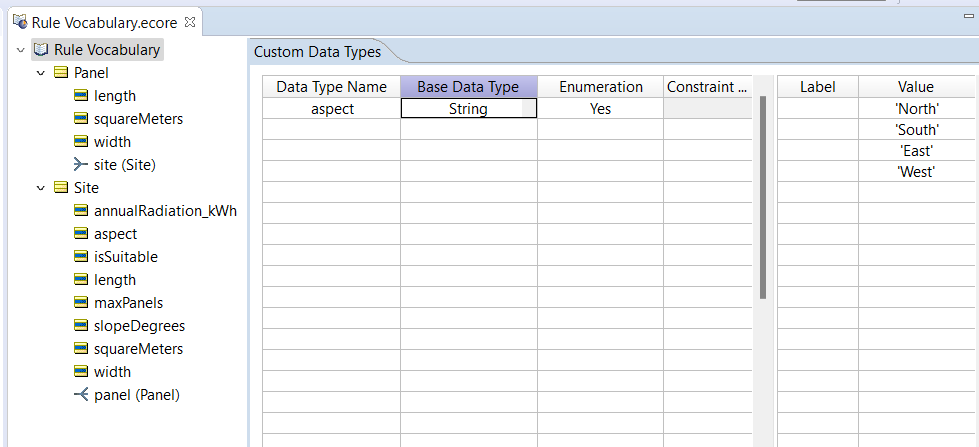
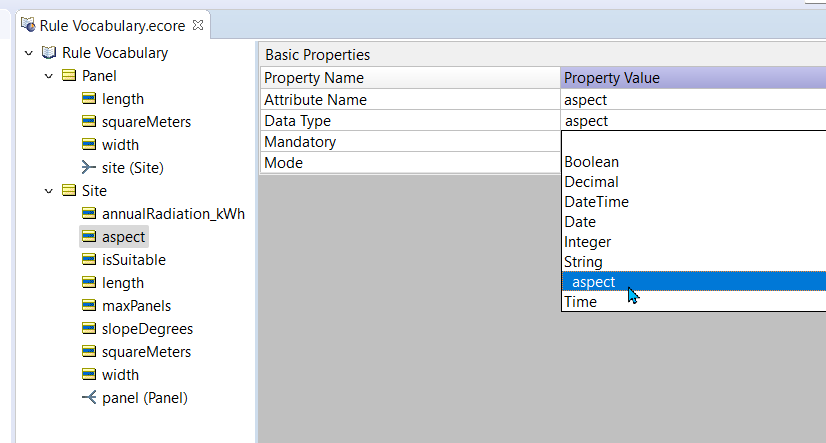
Now, add the rest of the vocabulary elements we’ll need. Add these attributes to ‘Site’:

* length (decimal)
* width (decimal)
* maxPanels (integer)
* annualRadiation\_kWh (decimal)
* aspect (string)
* isSuitable (boolean)
* slopeDegrees (decimal)
* Add these attributes to ‘Panel’:
* length (decimal)
* width (decimal)
* squareMeters (decimal)

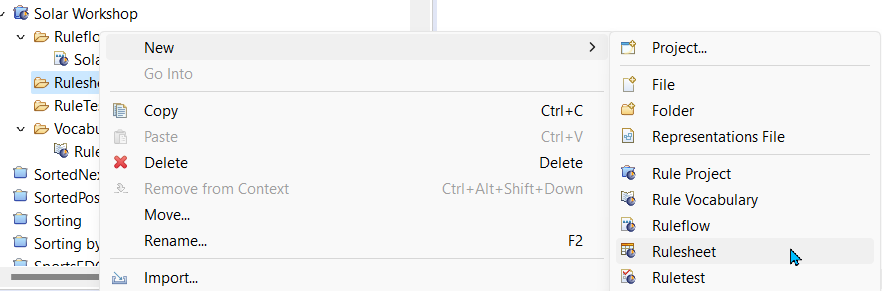
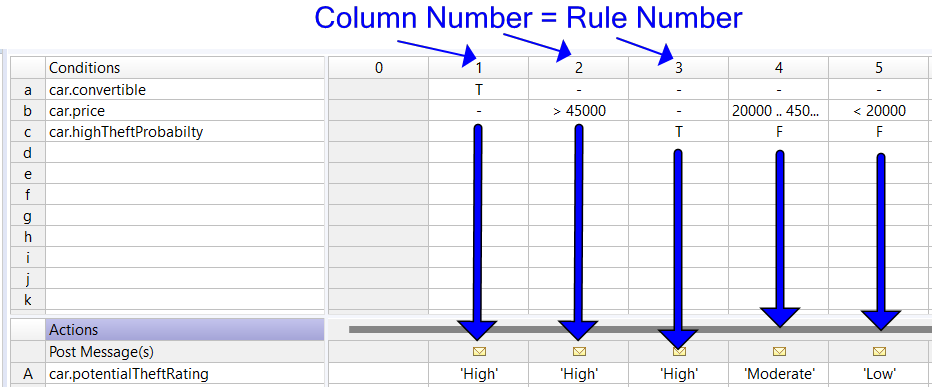
Your vocabulary should now look like:

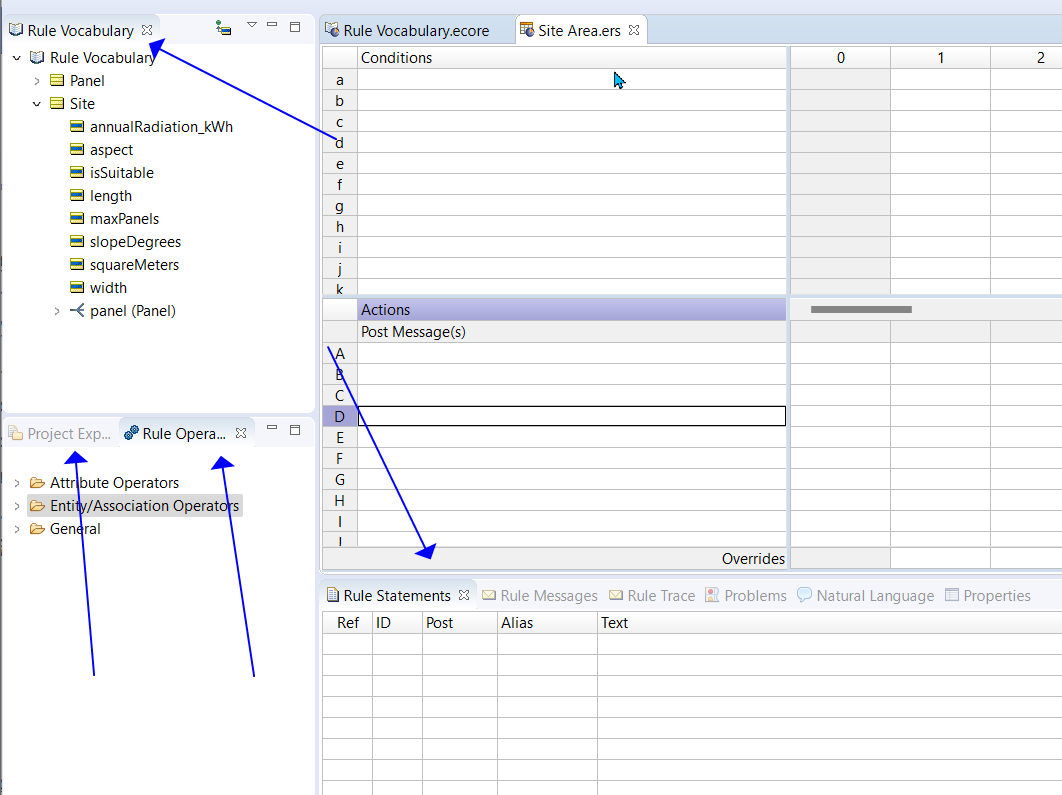


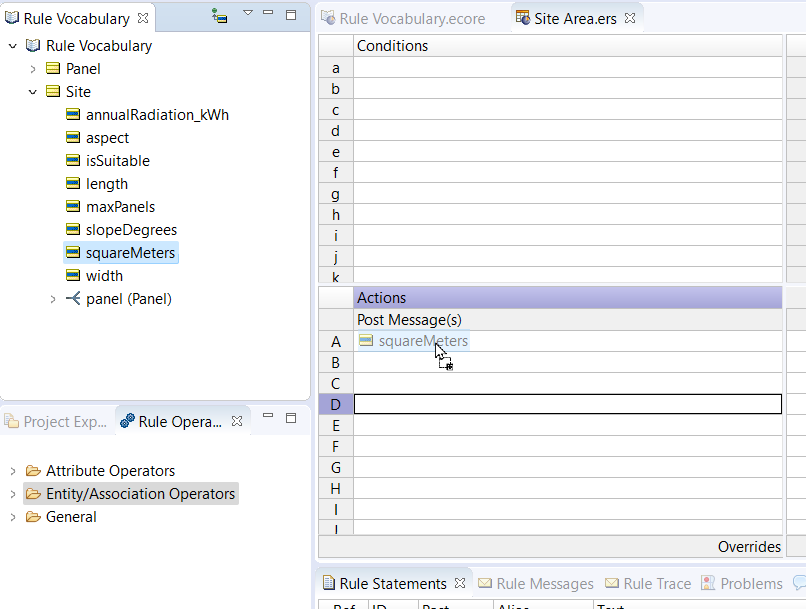
As you likely deduced from the rule, “Aspect should not be north,” the word ‘aspect’ is used here to mean compass direction (north, south, east, west). Each of the four compass directions are ‘string’ (alphanumeric) datatypes, but there is a specific list of potential values for those strings—we want to make sure we don’t deviate from those specific values when modeling rules. For this situation, we’re going to define a ‘Custom Data Type’:

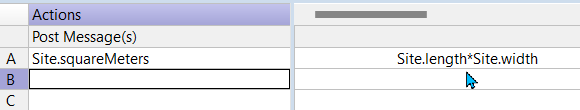
1. Click the name of the vocabulary at the top of the vocabulary tree, as shown: 
2. In the righthand side of the vocabulary editor, you’ll see a pane come into focus called ‘Custom Data Types’. In the table, type ‘aspect’ on the first row under Data Type Name, then select ‘String’ from Base Data Type. The default selection under ‘Enumeration’ will be ‘Yes’, don’t change this. Lastly, on the right hand side, under ‘Value’, type the four compass directions on the first four rows. Your custom data types table will look like this once complete: 
3. Lastly, click the ‘aspect’ attribute underneath the Site entity that you created previously. On the righthand side, a panel labeled ‘Basic Properties’ should appear. Next to the row called ‘Data Type’, which will currently say ‘String’, select the dropdown button and select the newly created data type ‘aspect: . You’ll see why we defined this custom data type in the next step.

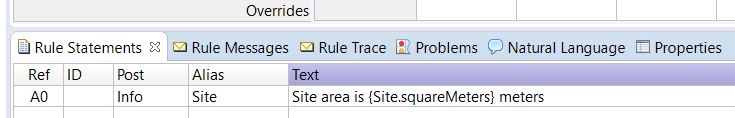
## Create your first rulesheet

Let’s start with a rule that will solve for a site’s area in square meters. First, right click on the ‘Rulesheets’ folder and select New > Rulesheet:  Give it the name ‘Site Area’, and then click Finish. You define your rule logic in a Corticon Rulesheet. A rule is like an ‘if-then’ statement. Each rule consists of one or more conditions (if) that are associated with one or more actions (then). Here is an example of a Rulesheet with five rules:  The Rulesheet editor has the following parts: Conditions rows—for defining the specific value(s) of a given vocabulary attribute which should trigger this rule should fire. For example, car.price> 45000. The condition value could be a single value (45000), a set of values (45000, 60000), or a range of values (20000 .. 45000). Actions—where you assign the value(s) to assign to vocabulary attribute(s) when the conditions are met. For example, car.potentialTheftRating= 'High'. But what about unconditional rules? For example, let’s say our Solar site is always a rectangle, so square meters can always be solved for by ‘area=length \* width’ with no conditions involved. This is called an ‘action-only’ rule—it will always fire. We define these in column 0. Let’s try it out. Open your newly created ‘Site Area.ers’ rulesheet. When defining rules, it is easiest to arrange the different panels of the screen in the way shown here:



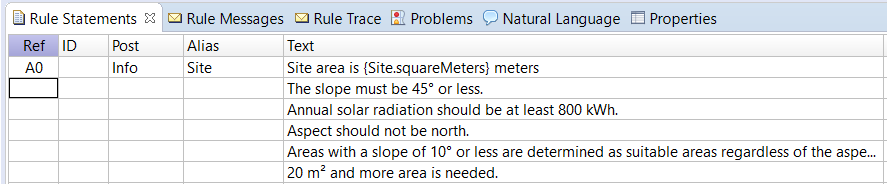
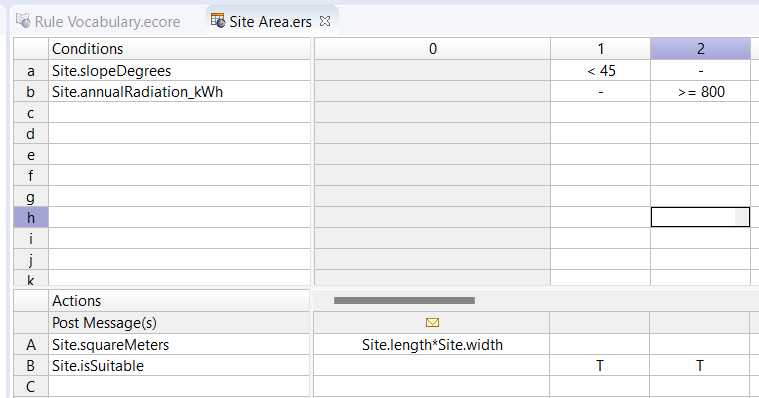
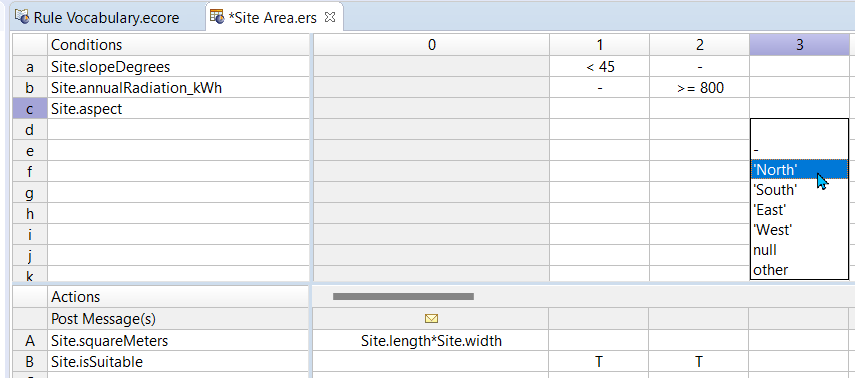
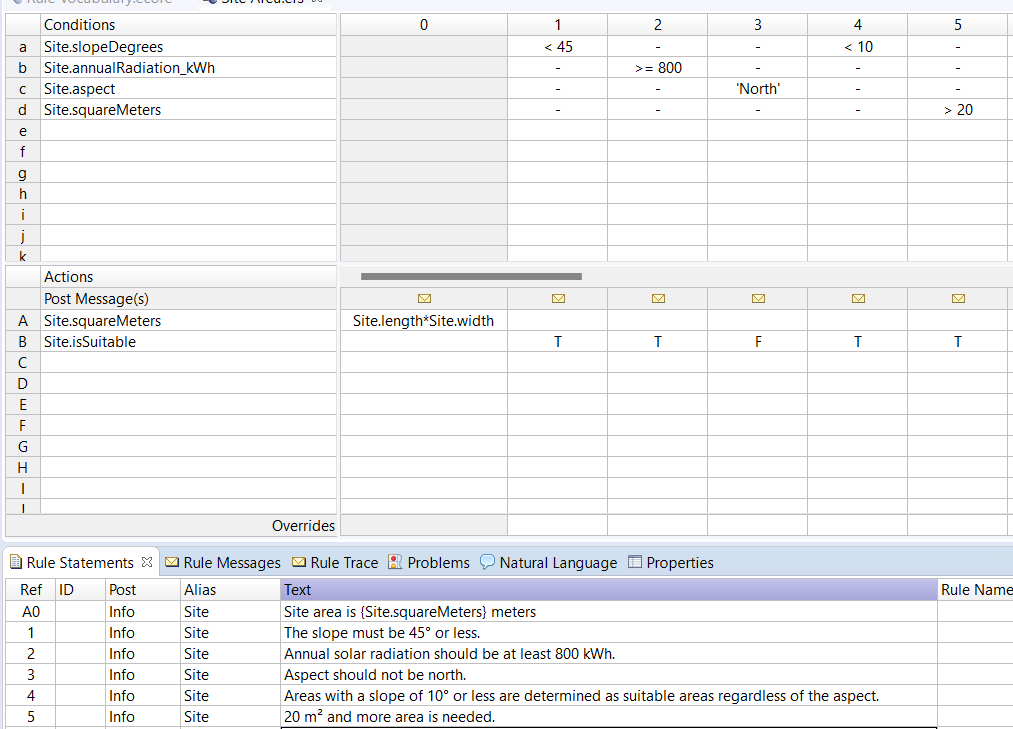
1. From vocabulary panel on the left, select the attribute ‘squareMeters’ from underneath the Site entity, and drag it onto the first action row: 
2. In the 0 column, we’re going to define the expression which will be used to assign the value for the ‘squareMeters’ attribute. Drag and drop (or just type in) the two other attributes that will be multiplied-- Site.length\*Site.width:



1. Now, lets define a rule statement for this rule. Rule statements are ‘explainers’ of a sort for each rule that fires during the execution of the decision service, to help document the rules which contributed to the final output. In the Rule Statement section, copy the text below into the Text column in the first row: Site area is Site.squareMeters meters
2. Add the other values shown in the picture below to your rule statement: 

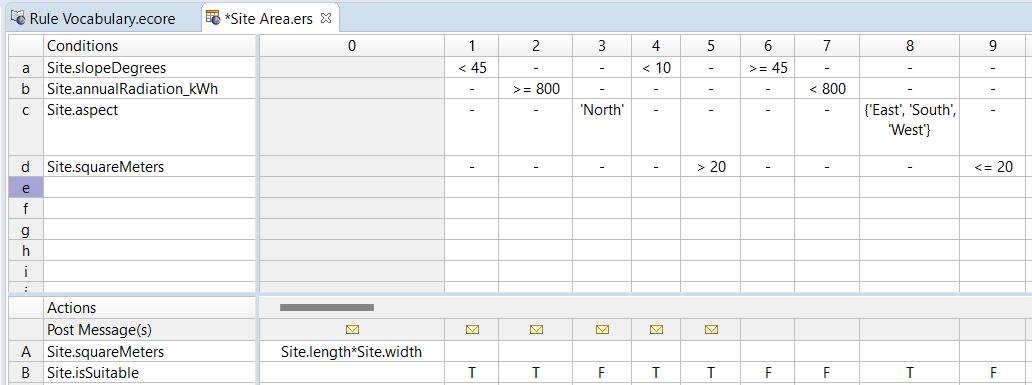
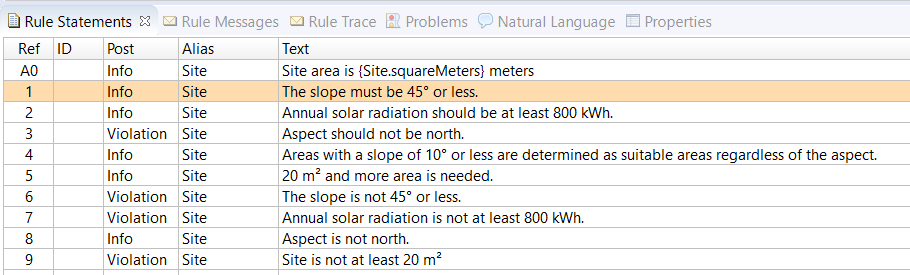
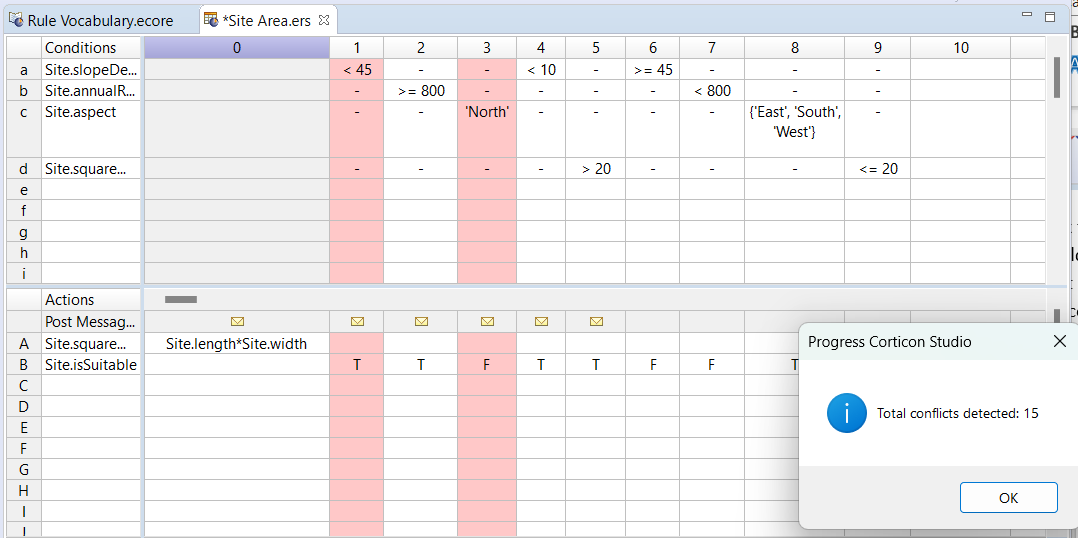
Let’s now define the conditions and actions for our initial set of rules. We will set the boolean attribute ‘isSuitable’ based upon whether the Site meets these conditions:

* The slope must be 45° or less.
* Annual solar radiation should be at least 800 kWh.
* Aspect should not be north.
* Areas with a slope of 10° or less are determined as suitable areas regardless of the aspect.
* 20 m² and more area is needed.

1. Copy and paste the five rows above into your Rule Statements underneath the one we’ve just defined: 
2. Drag Site.slopeDegrees onto the first Condition row. In the first row of the ‘1’ column, type '< 45'
3. Drag Site.isSuitable to the second Action row, then ‘T’ from the dropdown.
4. Drag Site.annualRadiation\_kWh to the second Condition row. In the second row of the ‘2’ column, type >=800. Set the corresponding action to also be Site.isSuitable = T
5. Drag Site.aspect to the third condition row. In column 3, select the third condition row. A dropdown button will be available on the right side of the cell—select North from it: 
6. Set the corresponding action to set isSuitable to F.
7. Implement the next two rules and then update the Rule Statements as shown below. 

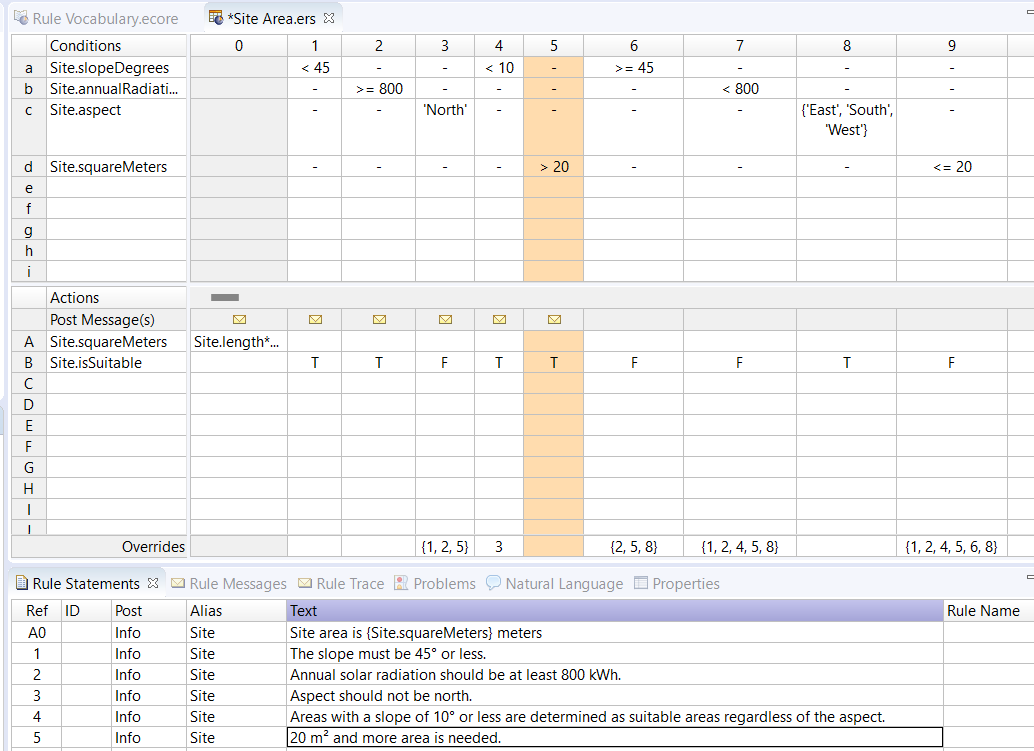
### Logical Integrity Checks

Before we test these rules with test data, Corticon can analyze them for logical issues we may have introduced. Look for these three buttons  at the top of the rulesheet editor screen (if they’re greyed out, then click anywhere in the rulesheet to bring it into focus).

1. Click them from left to right, start with the Logical Loop Checker. As the name implies, if we were introducing any circular logic, Corticon would point us to it automatically with this check. We shouldn’t have any loops.
2. Next is the Completeness Checker—are there more scenarios implicitly possible based upon the rules we’ve authored so far? Corticon will condense all possibilities into a new column. This can be helpful, but some of the new conditions should be evaluated independently, and we’re not going to worry about null checks. You can select this column and click the delete button.
3. Instead, let’s implement the missing rules in a few additional columns such that your rulesheet looks like this:  Since we have new rules, it is a good idea to add our additional rule statements for the rules.  You can copy and paste the italicized text below directly into your rule statements pane to save yourself some time.
4. Finally, let’s run the Conflict Checker. Here, we can really see the value of business users that are well-acquainted with the rules being the implementers of the logic versus developers, because they will know best how to address logical mistakes in the rules:  Corticon has identified 15 conflicts—situations where the conditions we’ve defined could presumably overlap. Currently, any of the cells with a ‘ – ‘ in them will evaluate that rule without considering the vocabulary attribute on the left hand side. For example, the first conflict which Corticon shows us is a conflict between 1 and 3, because Corticon has inferred there may be scenarios ‘North’ could be the value in cell c2, or a value < 45 could be in the cell a3—if both conditions are met, which matters more?

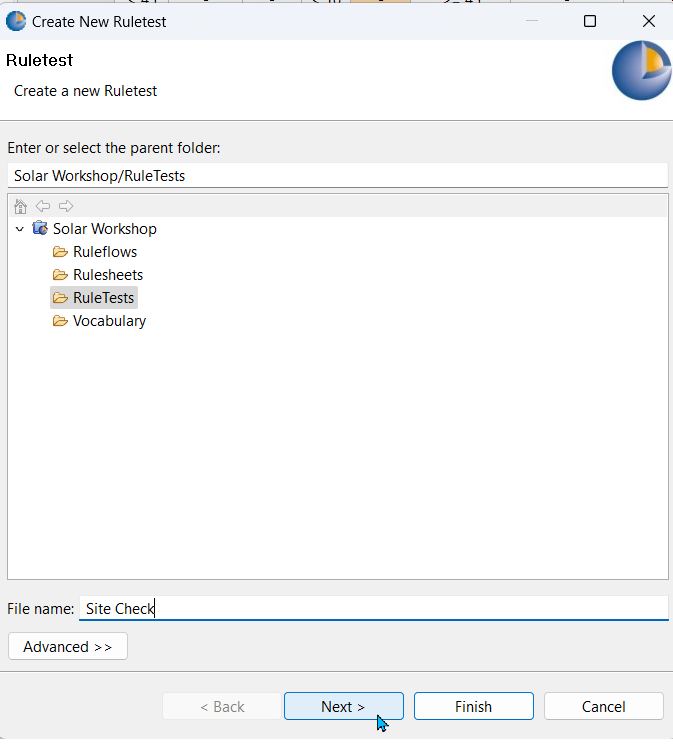
We can resolve this in a few ways—explicitly define every combination of values in every rule, or simply override one with another in cases of conflicts. We’re going to evaluate these rules such that if any aspect of the Site makes is unsuitable for solar, then that it will take precedent over any condition which resolves to isSuitable=T. However, recall that the condition for rule #4—when the slope is less than 10 degrees—will resolve to isSuitable=T even when the Site.aspect= ‘North’.

We’ll define overrides to tell Corticon which action to defer to in cases of conflict. Other than rule #4 overriding number #3 for the reason mentioned above, all other rules that lead to a determination of isSuitable=F should override those which set isSuitable=T. At the bottom of each rule column, you’ll see an override row where you can specify which rule(s) that particular rule will override in cases of conflict. You can type them in or select them from the dropdown (hold down control to select multiple). Try this on your own, then check your work based on the screenshot below.

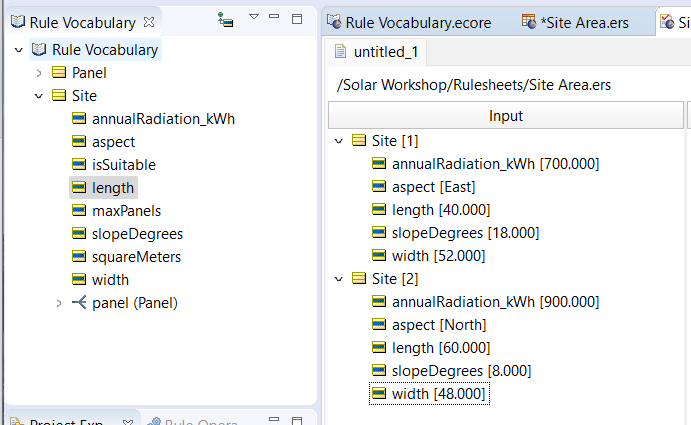


## Testing Rules

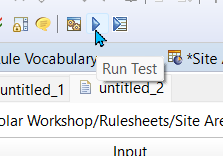
Next, let’s set up a rule test to run test scenarios against this rulesheet. A Ruletest simulates a business scenario where the rules are applied to input data. If the data satisfies all the conditions in a rule, the rule fires and some output containing the results of the rule execution is produced. You can define different sets of input data to test how the rules behave in different scenarios. You can also use a Ruletest to compare the output of a rule execution with expected results. A Ruletest stores this information in a Ruletest file, enabling you to save use-cases that are of interest, change rules, and run the test again to see how the modified rules behave when applied to the same use-cases.

1. Right click on the Ruletests folder > New > Ruletest. ![](images/Solar Workshop\_image072.png)
2. Give it a name like ‘Site Check’, and select the RuleTests folder if it isn’t already. 
3. Hit next, and you should see the ‘Site Area’ rulesheet already selected. This is the ‘test subject’, i.e. the specific rule asset file we’re testing against. Click Finish.

The Ruletest editor has four parts: **Test Subject**—specifies which Rulesheet or Ruleflow is being tested **Input**—where you define input data to be processed by the rules in the Rulesheet. **Output**—where Corticon Studio displays the result of a Ruletest execution. **Expected**—where you can optionally define the result that you expect. In the real world, a Corticon Decision Service may receive input in different formats such as XML, JSON, Java, or .NET objects. However, in a Ruletest, you just specify input data using the same Rule Vocabulary we’ve used for the rules.

1. You can specify different entities as well as multiple instances of the same entity. Let’s try it—drag in two instances of the ‘Site’ entity, into the input pane. This adds all the attributes in the entity to the Input pane. However, you can delete attributes that you do not need.
2. Double click the attributes to specify values. The syntax for specifying values is similar to specifying values for attributes in Rulesheets, with one difference—you must not enter any values in quotes, even if the data type of the attribute is String, DateTime, Date or Time. If you enter quotes, the Ruletest treats it as part of the value. Copy the values as shown below for your input: 

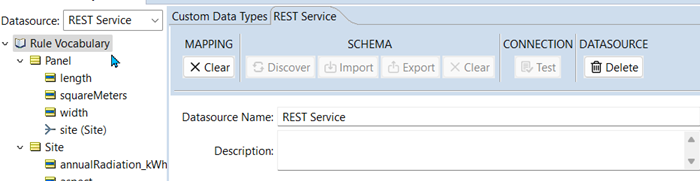
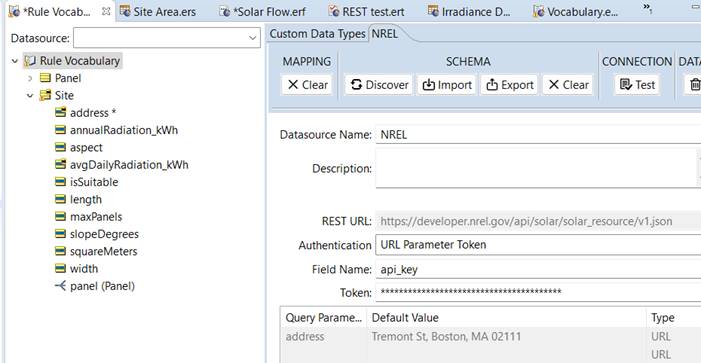
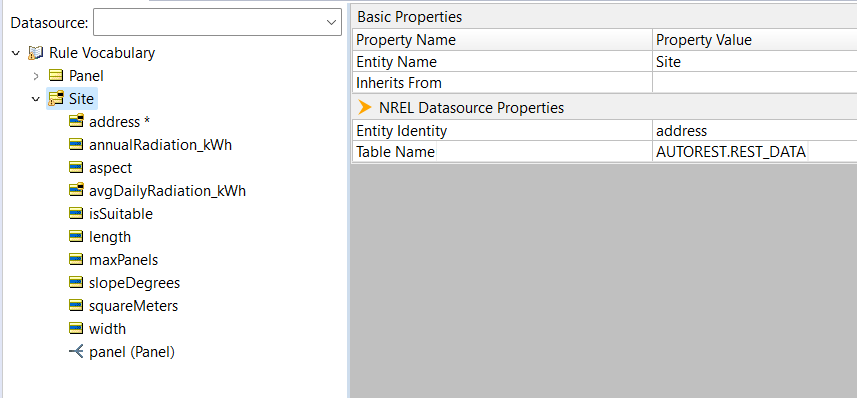
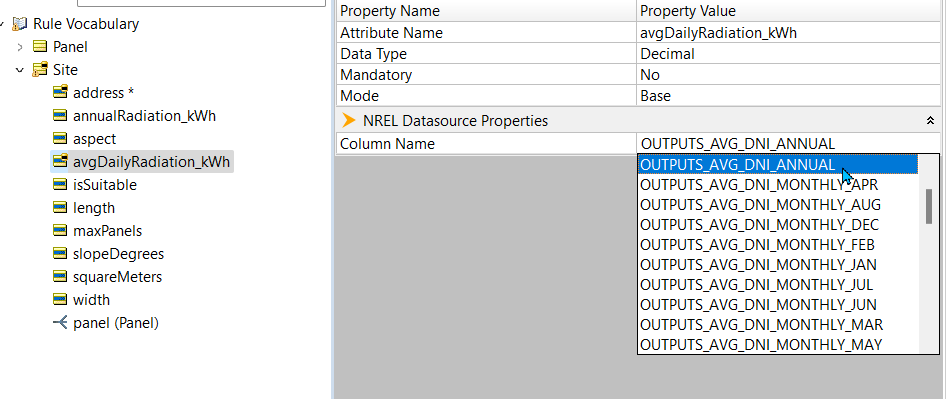
Above the ruletest, click the Run Test button:



The rules will be compiled into a Decision Service and executed against a test instance of the Corticon Server. In other words, the ruletest enables us to replicate the precise behavior we would expect from the rules if they were deployed as is. You should see the output populated in the Output pane, and Rule Messages sent back for the rules that fired.

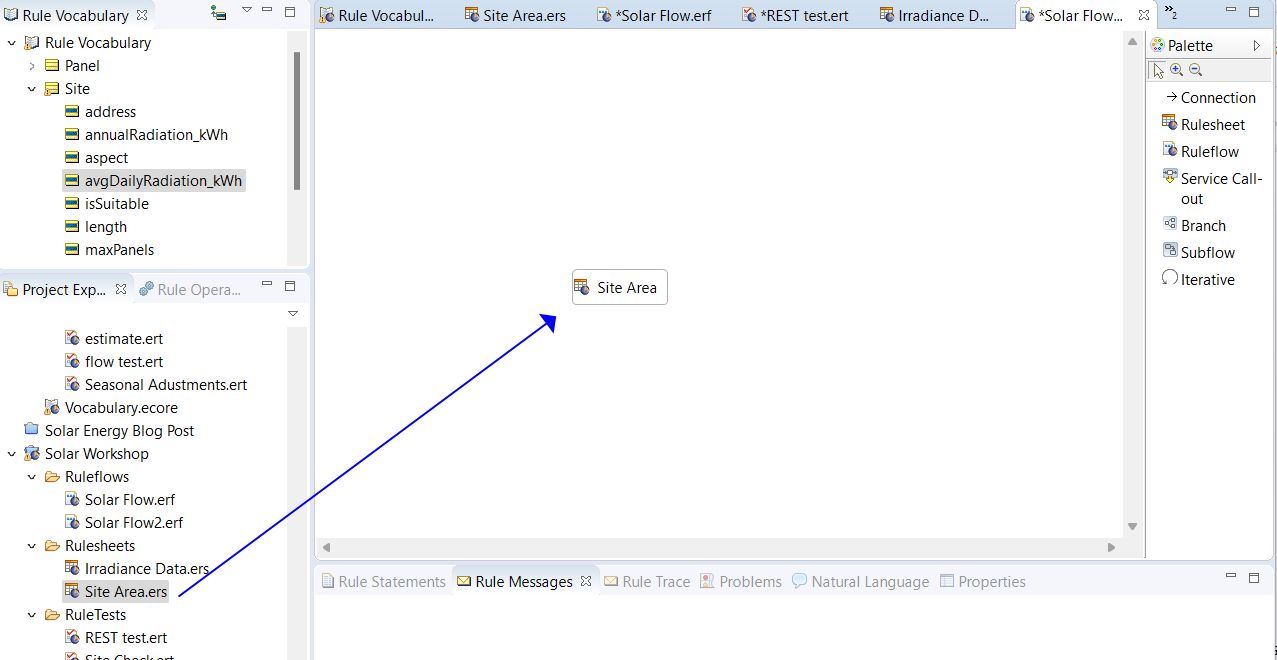
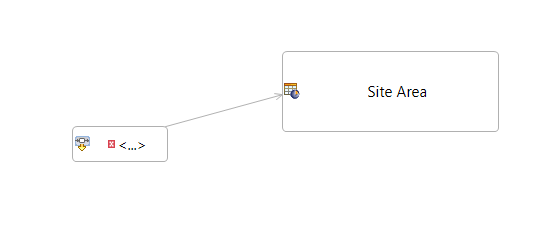
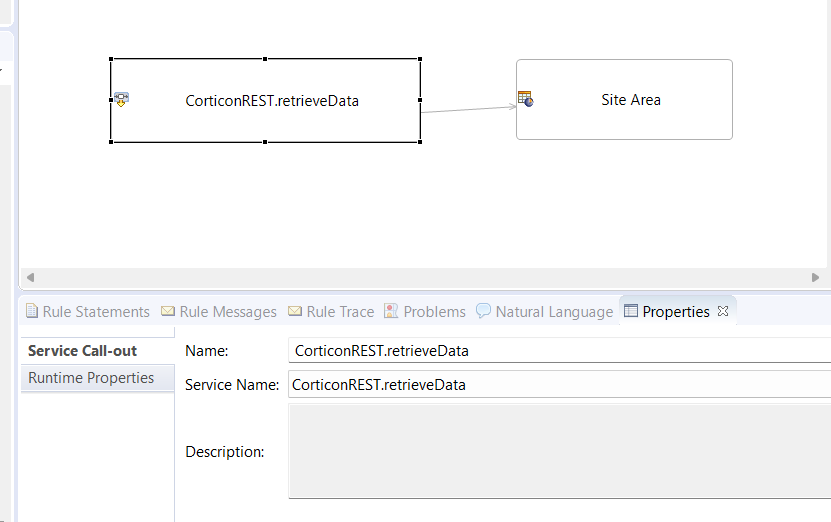
## Data Integration

So far, we’ve built a rulesheet and a test case in which we define the test data. Corticon is not limited to only evaluating data provided on the initial request to the Decision Service (like the data we defined by double clicking the attributes in the ruletest). Let’s incorporate some external data retrieved from a REST API. We’ll use the API provided by the US National Renewable Energy Lab to retrieve data about the solar radiation for a specific address. We’ll be using an API key that will expire after today’s workshop, but they are free to generate at NREL.gov.

1. Reopen your rule vocabulary
2. Add two attributes to Site—address (string), and avgDailyRadiation\_kWh (decimal)
3. At the top of the screen, click the Vocabulary menu > Add Datasource > REST Datasource
4. Click the name of the rule vocabulary to bring the datasource configuration pane into focus: 
5. Rename the Datasource Name from the default, “REST Service”, to “NREL”.
6. In the REST URL field, paste the URL: https://developer.nrel.gov/api/solar/solar\_resource/v1.json?address=Tremont%20St,%20Boston,%20MA%2002111
7. On the Authentication dropdown, select URL Parameter Token
8. Under field name, enter ‘api\_key’
9. Under token, paste your API key 
10. Click ‘Test Connection’. If everything is in the right place, you should get an alert that the connection was successful.
11. Click ‘Discover’. This will import the various fields at that datasource endpoint, which we will then ‘map’ to our entities/attributes.
12. Click your ‘Site’ entity. On the right hand side, select AUTOREST.REST\_DATA from the dropdown next to Table Name, then select ‘address’ from the entity identity dropdown. 
13. Next, click the address attribute. This time on the righty hand side, select ADDRESS from the column name dropdown.
14. Finally, do the same for the new avgDailyRadiation\_kWh attribute, selecting OUTPUTS\_AVG\_DNI\_ANNUAL from the column name dropdown. 

### Incorporate the REST Datasource

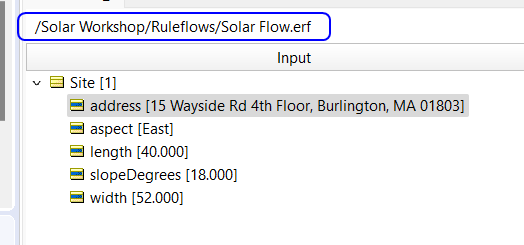
The one file we haven’t worked with yet is the Ruleflow. A Ruleflow enables you to connect two or more Rulesheets in a sequence. When the Ruleflow is processed at runtime, the Rulesheets are executed one by one in that sequence. The output of one Rulesheet becomes the input of the next Rulesheet. Note that a Rulesheet can be used in multiple Ruleflows. This enables you to use Rulesheets as reusable modules of rule logic. Any change in the rule logic only has to be made once in the Rulesheet and it is propagated across all Ruleflows that refer to it. Besides defining a sequence of rulesheet executions, we can specify behavior like branching to different rulesheets at specific points in the decision execution based upon attribute values. We also specify the REST callout within the ruleflow.

1. Right click the Ruleflows folder > New > Ruleflow
2. Name it Solar Flow and click finish
3. When we were defining rules in a rulesheets, we dragged and dropped from the Rule Vocabulary. For the ruleflow, we’ll be defining the order to rule execution by dragging and dropping our rulesheets onto the Ruleflow canvas, and then connecting them using the Connection tool on the righthand side of the canvas. Drag the Site Area rulesheet onto the ruleflow 
4. Click ‘Service Callout’ on the panel to the right, then click above the Site Area rulesheet to place the callout node. Connect the callout to the rulesheet by click Connection in the righthand pallet and dragging a connection from the callout to Site Area. 
5. Click on the service callout, and bring the ‘Properties’ pane into view. You should see a dropdown for Service Name—select ‘Retrieve Data’.
6. Next, click ‘Runtime Properties’ on the tab to the left, underneath the ‘Service Call-out’ tab. Here, you’ll set options for 6 different cells by choosing from each cell’s dropdown. Mirror the content shown below: 

### Update the rules

The last step we need to perform is to annualize the solar radiation data because we need the sum total of kWh. We’ll solve for this by multiplying the daily kWh value which will be retrieved from the data source by 365. We can do this by just adding another action only rule to the Site Area rulesheet. Reopen that rulesheet, and define the new rule to annualize the data: 

### Test the Updated Rules

Let’s give this thing a whirl. Open your ruletest back up. Change the test subject to the ruleflow from the rulesheet by double clicking the folder path above the input pane, highlighted below. Change the inputs to match the input shown in the same photo:  When you execute this test, Corticon will retrieve the data value for avgDailyRadiation\_kWh from the REST endpoint, incorporate it into the data used for the rule processing, and evaluate the site’s suitability based upon the existing rules and the annualized kWh value.

