Specific use cases

On this page

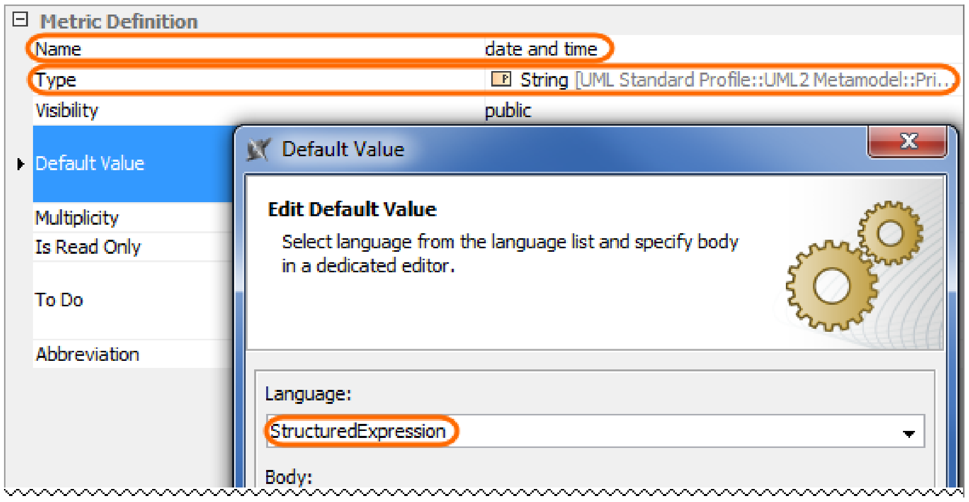
How to build a metric definition for date and time tracking?

If your metric suite isn't created using the *BaseMetricSuite*, you must build the metric definition for date and time tracking on your own.

To build the metric definition for date and time tracking

---

1. Add a new metric definition with the following characteristics to your metric suite:
   - *date and time* as name
   - **String** as the result type
   - **StructuredExpression** as the language of the metric definition body



Metric Definition

Name	date and time
Type	String [UML Standard Profile::UML2 Metamodel::Pri...
Visibility	public
Default Value	
Multiplicity	
Is Read Only	
To Do	
Abbreviation	

Default Value

Edit Default Value

Select language from the language list and specify body in a dedicated editor.

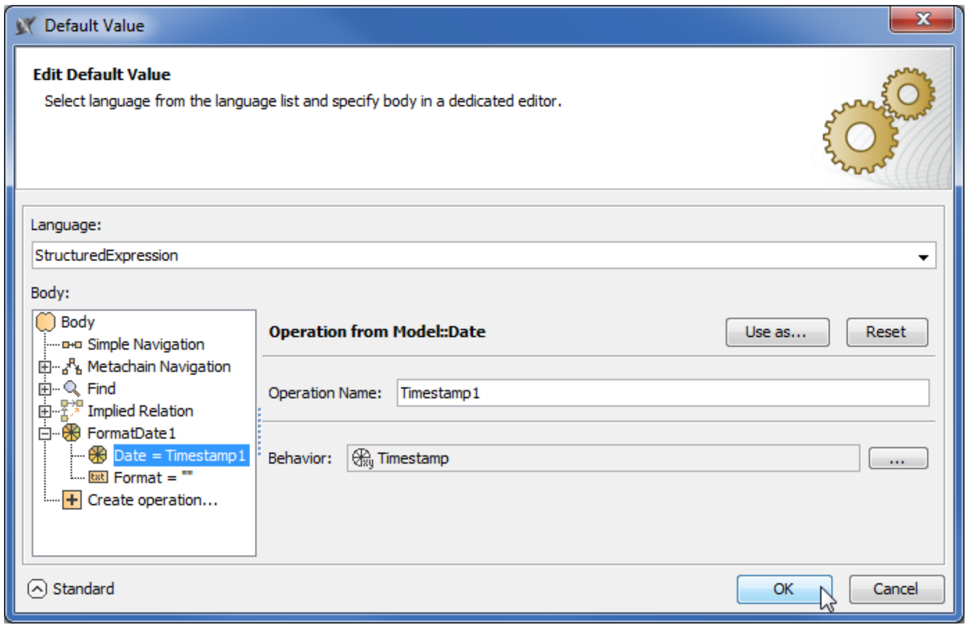
Language: StructuredExpression

Body:

2. In the **Body** area of the **Default Value** dialog, click **Create operation**, and then select **Operation from Model** on the right side of the dialog.
3. In the open dialog, select the **FormatDate** built-in operation which is stored in the *UML Standard Profile* and click **OK**.

> Make sure you clear the **Apply Filter** check box in the **Select Opaque Behavior** dialog.

4. In the **Body** area of the **Default Value** dialog, select the **Date** parameter of the **FormatDate** operation and click The **Reset** button.
5. Select **Operation from Model** on the right side of the dialog.
6. In the open dialog, select the **Timestamp** built-in operation which is stored in the *UML Standard Profile* and click **OK**.



7. In the **Body** area of the **Default Value** dialog, select the **Format** parameter of the **FormatDate** operation and in the **Value** box type "dd.MM.yyyy hh:mm aaa".
8. Click **OK**, and then close the metric definition Specification window.

Now you can track the date and time of each instance of the metric suite.

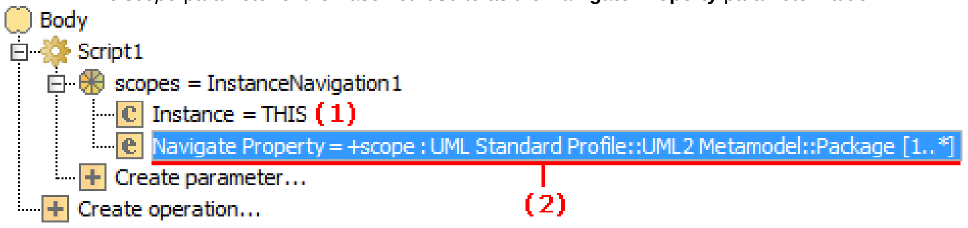
How to access parameter values in a metric definition?

If you need to use the values of a parameter for specifying calculations in a metric definition, you can easily access them with the help of the functionality provided by the StructuredExpression language. Let’s overview the case when you need to use the value of the *scope* parameter inherited from the *BaseM etricSuite* to calculate your metrics.

To access the value of the *scope* parameter inherited from the *BaseMetricSuite*

1. Add a new metric definition to your metric suite.
2. Select the **StructuredExpression** language for specifying the calculations.
3. In the structured expression, create a script operation with single parameter named *scopes*.
4. Select the **InstanceNavigation** built-in operation as the *scopes* parameter value.
5. Specify the following parameter values of that operation:

   - **Contextual Variable** as the **Instance** parameter value.
   - The *scope* parameter of the *BaseMetricSuite* as the **Navigate Property** parameter value.


Body
Script1
scopes = InstanceNavigation1
Instance = THIS
(1)
Navigate Property = +scope : UML Standard Profile::UML2 Metamodel::Package [1..*]
(2)
Create parameter...
Create operation...

6. In the script operation, refer to the *scopes* parameter simply by its name as the following Groovy script sample illustrates:

```
import static com.nomagic.magicdraw.modelmetrics.ScriptHelper.*;
import static com.nomagic.requirements.util.SysMLProfile.*;

getElementsRecursively(scopes)
findAll {el -> isRequirement(el)}
size()
```

An alternative way to access parameters is calling the *getValue()* method from MagicDraw Open API on the *valueContext* globally defined value, as the following Groovy script sample illustrates:

```
def scopes = valueContext.getValue(“scope”)
```

How to build a metric definition that takes the result of another metric definition?

Let’s say, you need to calculate the percentage of requirements that are covered by blocks. To do this, you must have the following values:

- Overall requirements count
- Requirements covered by blocks count

Let’s say, these values are the results of two already existing metric definitions: *requirementsCount* and *requirementsCoveredByBlocksCount*. You need another metric definition, which takes the results of the latter metric definitions and calculates the percentage. To access these values and make sure they are already calculated, we must call the *getOneValue()* method from MagicDraw Open API on the *valueContext* globally defined value for both metric definitions, as the following Groovy script sample illustrates:

```
int all = valueContext.getOneValue(“requirementsCount”);
int covered = valueContext.getOneValue(“requirementsCoveredByBlocksCount”);
com.nomagic.magicdraw.modelmetrics.ScriptHelper.calcPercentage(all, covered)
```

Related pages

- Metric Suites
    - Adding a new Metric Suite
    - Specify the target for the Metric Suite
    - Specifying parameter definitions
    - Building metric definitions
    - Building validation-based metric definitions