

# Writing scripts



The following instructions apply to BeanShell, Groovy, JRuby, JavaScript, and Jython scripts only.

## How to access the arguments and other values from the script body?

To access an argument from the script body, you should refer to the corresponding parameter name.

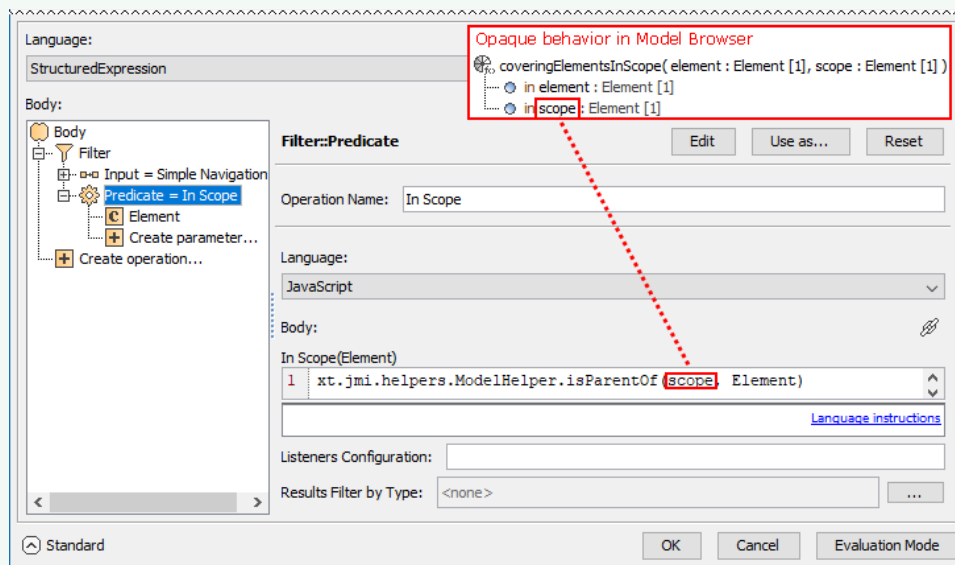
The screenshot displays the Script Editor interface. On the left, a tree view under 'Body' shows a 'Collection Size Calculation' operation. An orange circle highlights 'Context' in 'Context = Find', with an arrow pointing to the 'Script' tab. In the 'Script' tab, the 'Operation Name' is 'Collection Size Calculation'. The 'Language' is set to 'Groovy'. The 'Body' text area contains the code '1 Context.size()', where 'Context' is circled in orange and an arrow points to it from the 'Context' in the tree view. Below the code is a 'Language instructions' link. At the bottom, there are 'Listeners Configuration' and 'Results Filter by Type' fields, and 'OK', 'Cancel', and 'Evaluation Mode' buttons.

A script body can access the following values:

- Arguments passed to this script as parameters, such as the Context parameter in the preceding figure.
- Values passed to the structured expression, inside of which the script operation is defined.

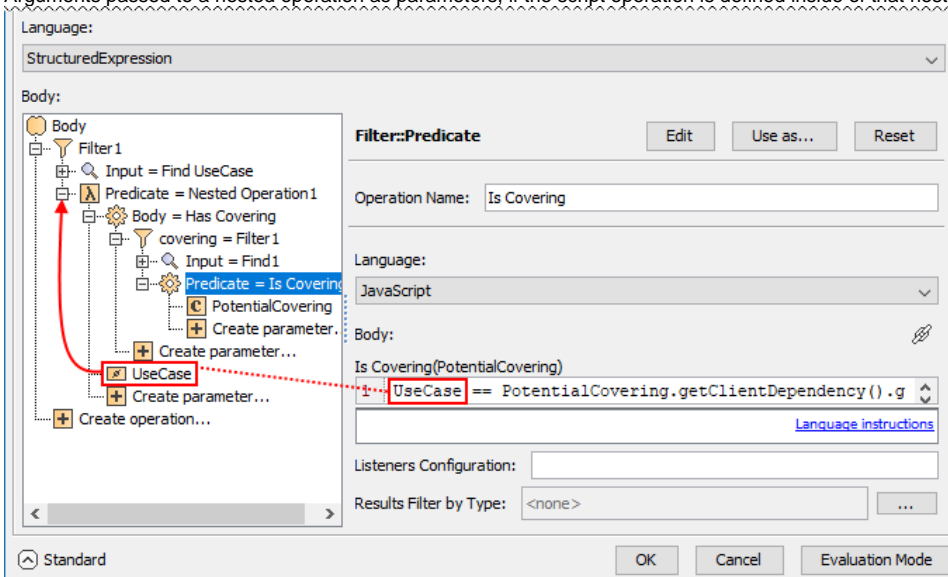


If you have an opaque behavior with a couple of parameters, you can refer to them from a script operation, which is inside of the structured expression - the body of this opaque behavior.



If you have a structured expression specifying a derived property, validation rule, relation criterion, etc., use *THIS* to refer to the [Contextual element](#).

- Arguments passed to a nested operation as parameters, if the script operation is defined inside of that nested operation.



- Globally defined values:
  - project (current project)
  - application

## What MagicDraw functionality can a script use?

The script can call MagicDraw Open API.



For the list of available [MagicDraw Open API methods](#), refer to javadoc.zip, which can be found in <MagicDraw installation directory>\openapi\docs.

Language: Groovy

Body:

Script(ctx)

```

1 boolean isCovered(e1) {
2     return !e1.getSupplierDependency().isEmpty();
3 }
4 ctx.findAll{e1->isCovered(e1)}.size

```


[Language instructions](#)

Listeners Configuration:

Results Filter by Type: <none> ...

OK Cancel Help

More complex model access operations are available in *ModelHelper* and *StereotypesHelper*.

 Use import statements to shorten java class names as shown in the following figure.

Language: Groovy

Body:

filteringPredicate(context)

```

1 import com.nomagic.uml2.ext.jmi.helpers.ModelHelper
2 ModelHelper.isParentOf(scope, context)

```

Short name of java class

[Language instructions](#)

Listeners Configuration:

Results Filter by Type: <none> ...

OK Cancel Help

## How to simplify scripts?

To simplify the script, move complex sub calculations out of the script. Use other operations of the structured expression to specify these sub calculations and then pass the results to the script through parameters.



Let's say we need a script operation, which counts the number of nested packages. You can use MagicDraw Open API to navigate through the model and find all the nested packages, but it would be complicated.

To simplify the script:

1. Use the Find operation to list these nested packages.
2. Use the script operation for nothing else than counting the number of the list items. Pass the result of the Find operation to this script through the PackageList parameter as the following figure shows.

The screenshot displays the MagicDraw configuration window for a script operation. On the left, a tree view under 'Body' shows a 'Count Nested Packages' operation. A red circle highlights the 'PackageList' parameter in the tree, and a red circle highlights the 'PackageList' parameter in the script body. A red dotted arrow points from the 'PackageList' parameter in the tree to the 'PackageList' parameter in the script body. The 'Script' panel on the right shows the 'Operation Name' as 'Count Nested Packages', the 'Language' as 'JavaScript', and the 'Body' as 'Count Nested Packages(PackageList)' followed by a script line '1 PackageList.size()'. The 'Results Filter by Type' is set to '<none>'. The bottom of the window shows 'Standard' mode, 'OK', 'Cancel', and 'Evaluation Mode' buttons.

Language: StructuredExpression

Body:

- Count Nested Packages
  - PackageList = Find Nested Packages
    - Search String = ""
    - Scope = THIS
    - Types = Collection
    - Include Subtypes = false
    - Filter Properties = ""
    - Text Filter Properties = ""
    - Regular Expression = false
    - Case Sensitive = false
    - Match Anywhere = true
    - Include Elements From Modules = false
    - Include Elements From Additional Content = false
    - Search Data Unused In Diagrams = false
    - Create parameter...
    - Create operation...

Script

Operation Name: Count Nested Packages

Language: JavaScript

Body:

Count Nested Packages(PackageList)

1 PackageList.size()

Listeners Configuration:

Results Filter by Type: <none>

Standard OK Cancel Evaluation Mode