MACRO ENGINE

user guide

No Magic, Inc.

2015

CONTENTS

MACRO ENGINE

1. Introduction

Macro Engine (previously called Script Engine) in MagicDraw allows you to create your own macro (script) by using BeanShell, Groovy, JRuby, JavaScript (Nashorn and Rhino), or Jython. With Macro Engine, you can control everything that is allowed in Open API, for example, transforming and manipulating models. You can find MagicDraw Open API in **MagicDraw OpenAPI UserGuide.pdf** in the manual directory and sample macros in **<MagicDraw>/samples/product features/macros**.

Macro Engine comes with Professional, Architect, and Enterprise Editions starting from MagicDraw version 16.6 and greater.

MagicDraw 18.1 Macro Engine supports two types of Javascript: (i) Javascript Nashorn and (ii) Javascript Rhino). This support is intended to prevent some incompatibility issues in the language syntax with macros that were created using Java 7 or earlier (Javascript Rhino), because Oracle changed the built-in Javascript engine from Javascript Rhino to Javascript Nashorn since Java 8 was released.

When you refer to Javascript, Macro Engine will refer to the default Javascript that comes with the JRE, which is Javascript Nashorn. For example, if you use Java 8, the default Javascript will be Javascript Nashorn. If you find any incompatibility issues, Macro Engine provides the migration capability to downgrade the Javascript engine for all of the existing macros to use Javascript Rhino.

To migrate to Javascript Rhino:

- Click **Tools** > **Macros** > **Migrate to Javascript Rhino**.

NOTE	If you are using MagicDraw 18.1 or greater with Java 8 and created a macro with the new language syntax in Nashorn, and then downgraded your Java to Java 7, this will cause an execution problem for your macro.

2. Working with Macro Engine

2.1 Selecting a Default Macro Language

Use the **Environment Options** dialog to select a default macro language.

To select a default macro language:

1. Click **Options** > **Environment** on the MagicDraw main menu (Figure 1). The **Environment Options** dialog will open (Figure 2).

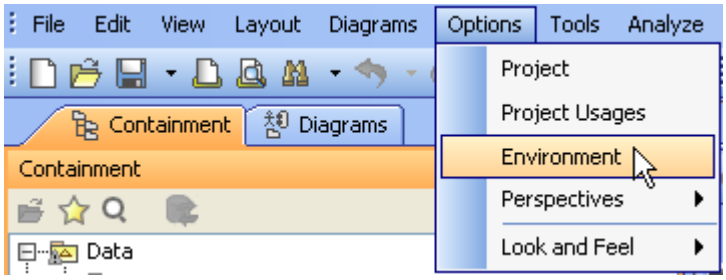

Figure 1 -- Environment Options Dialog Menu

2. Select the **Macros** node (Figure 2).

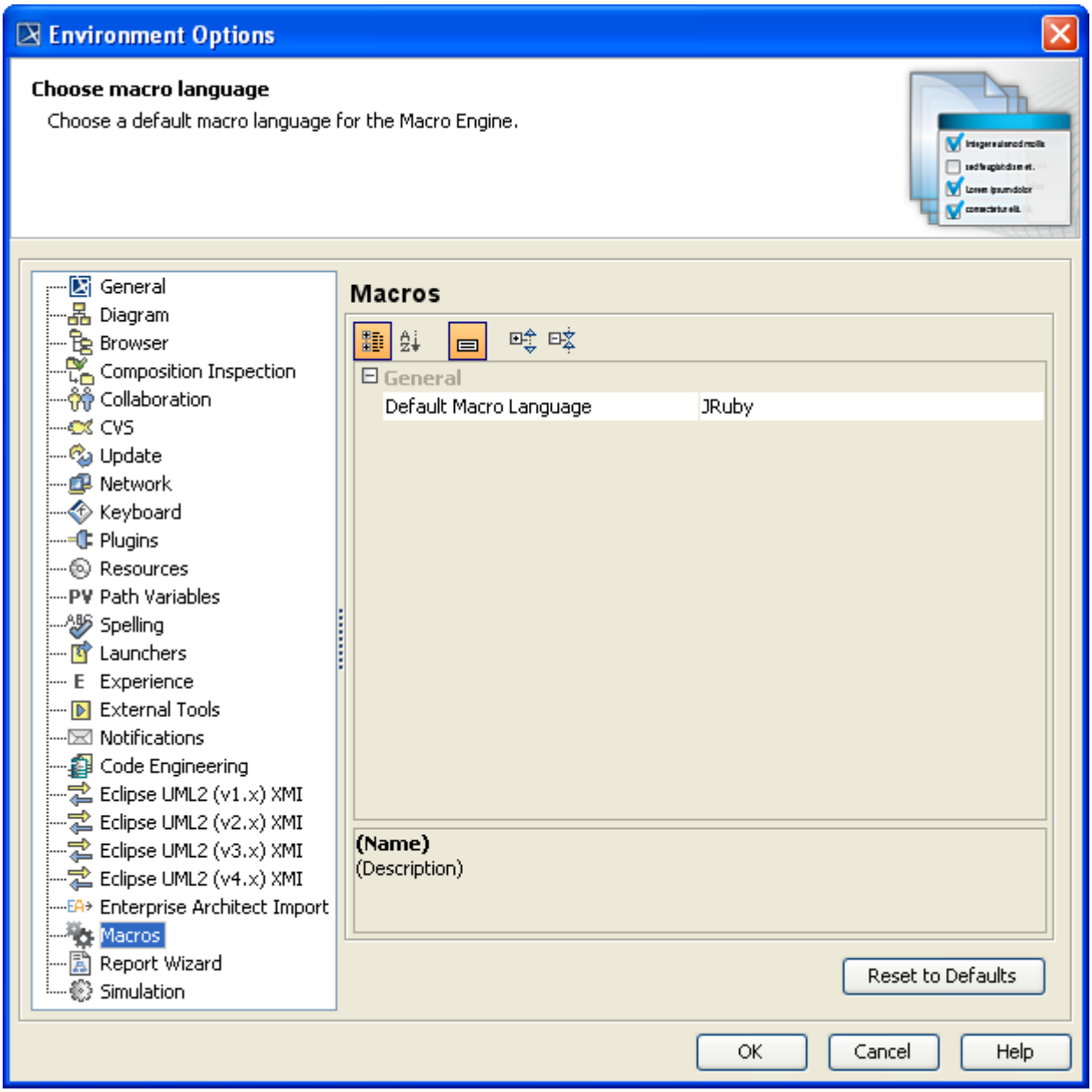

Figure 2 -- The Environment Options Dialog

3. Click the box next to the **Default Macro Language** box to see a list of supported programming languages (Figure 3).

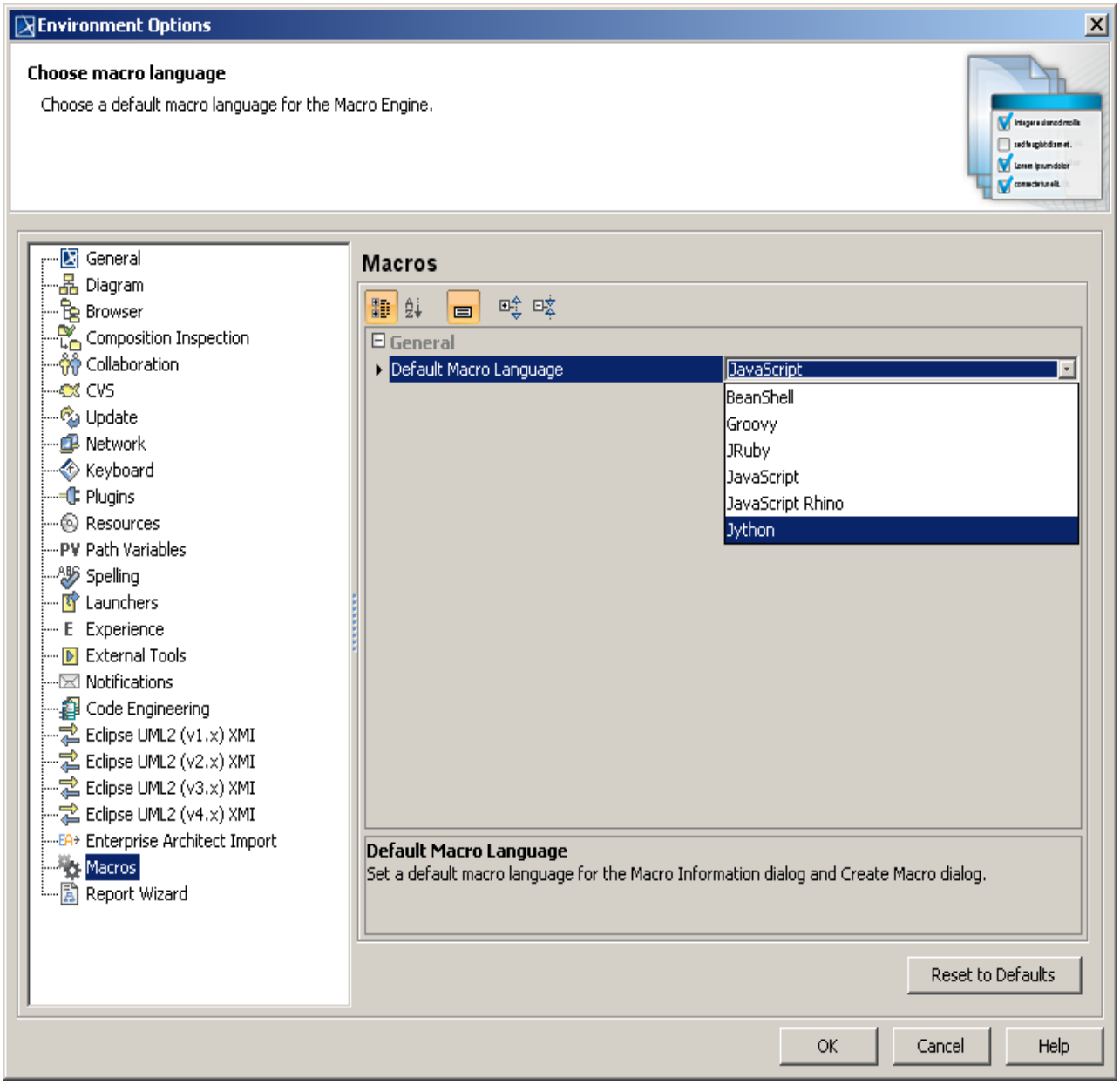

Figure 3 -- Selecting Macro Language

4. Select **Jython**, **BeanShell**, **Groovy**, **JRuby**, or **JavaScript**.
5. Click **OK** to save the selected language as the default macro language.

NOTE:	• JavaScript is the default macro language. • Macro Engine currently supports BeanShell, Groovy, JRuby, JavaScript (Nashorn and Rhino), and Jython only.

2.2 Creating a Macro

Once you have selected a default macro language (see **2.1 Selecting a Default Macro Language**), you can create a new macro by using the **Create Macro** dialog (Figure 4). The dialog allows you to specify a macro language, enter source code, and save it.

To create a new macro:

1. Click **Tools** > **Macros** > **Create Macro...** on the MagicDraw main menu. The **Create Macro** dialog will open (Figure 4).

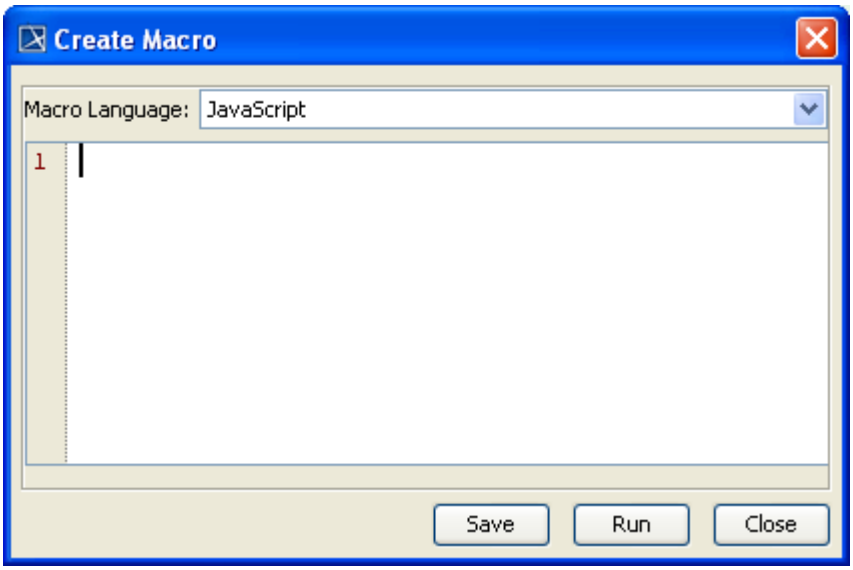

Figure 4 -- The Create Macro Dialog

2. Select a macro language in the **Macro Language** box.
3. Enter source code in the text box.
4. Click **Run** to test and make sure that the source code works properly.
5. Click **Save**. The **Macro Information** dialog will open (Figure 7). Follow the steps described in section **2.3.2 Adding a Macro and Its Information** below.
6. After clicking **OK** in the **Macro Information** dialog (Figure 7), the new macro will be saved in the location you have specified in the **File** text box.

2.3 Adding a Macro and Editing Macro Information

You can add a new macro and enter all necessary information about it in MagicDraw by following the steps described in sections **2.3.1 Opening Macro Information Dialog** and **2.3.2 Adding a Macro and Its Information** below.

2.3.1 Opening Macro Information Dialog

You can add or modify macro information such as the macro name and description, in the **Macro Information** dialog. To open the **Macro Information** dialog, you need to open the **Organize Macros** dialog first.

To open the **Organize Macros** dialog:

1. Click **Tools** > **Macros** > **Organize Macros...** on the MagicDraw main menu (Figure 5). The **Organize Macros** dialog will open (Figure 6).

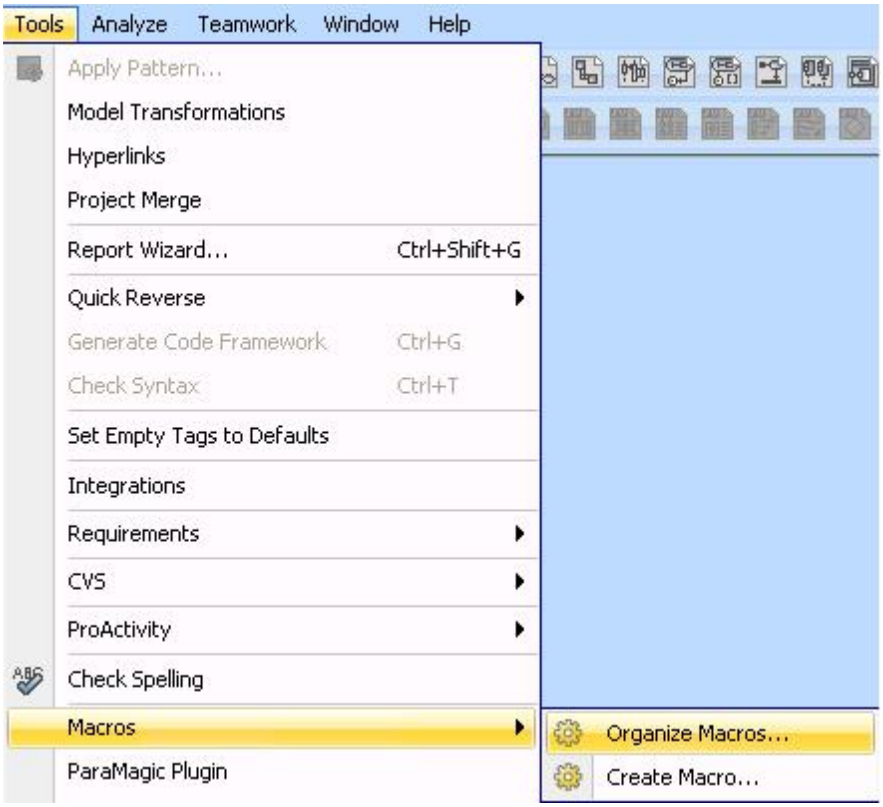

Figure 5 -- Organize Macros Menu

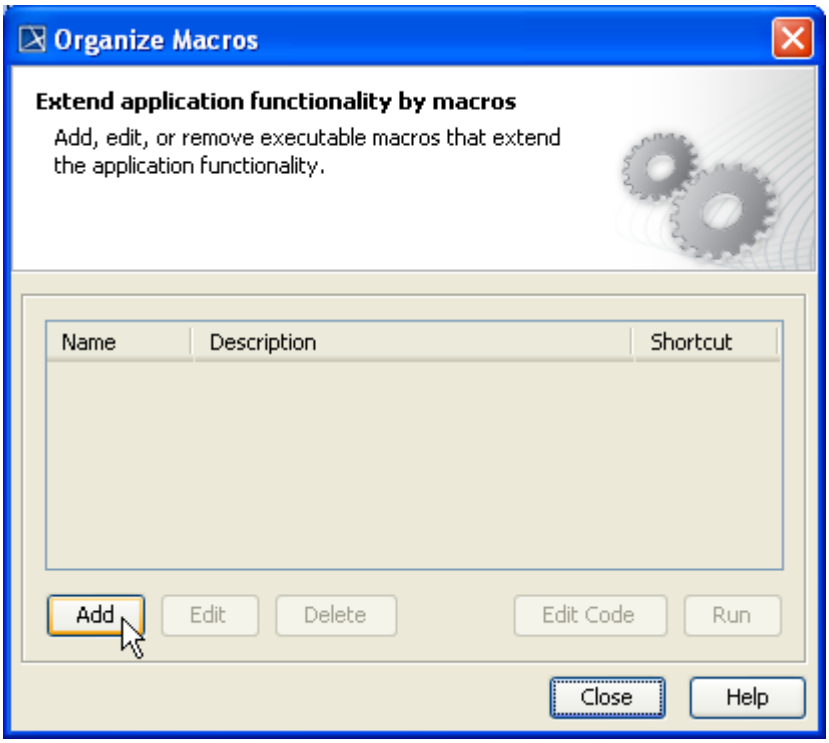

Figure 6 -- The Organize Macros Dialog

To open the **Macro Information** dialog:

1. Click **Tools** > **Macros** > **Organize Macros...** on the MagicDraw main menu to open the **Organize Macros** dialog.
2. Click **Add** (Figure 6). The **Macro Information** dialog will open.

2.3.2 Adding a Macro and Its Information

Use the **Add** or **Edit** button in the **Organize Macros** dialog to add or edit a macro and its information in the **Macro Information** dialog. You can also press the mnemonic keys to add or edit a macro (see **2.4.3 Organizing Macros Dialog Mnemonic Keys**).

To add a macro and enter macro information in the **Macro Information** dialog:

1. Open the **Organize Macros** dialog (click **Tools** > **Macros** > **Organize Macros...** on the MagicDraw main menu).
2. Click **Add**. The **Macro Information** dialog will open (Figure 7).

NOTE:	The **Edit**, **Delete**, **Edit Code**, and **Run** buttons in the **Organize Macros** dialog will be disabled if there is no macro in the macro table or if you do not select any macro from the table.

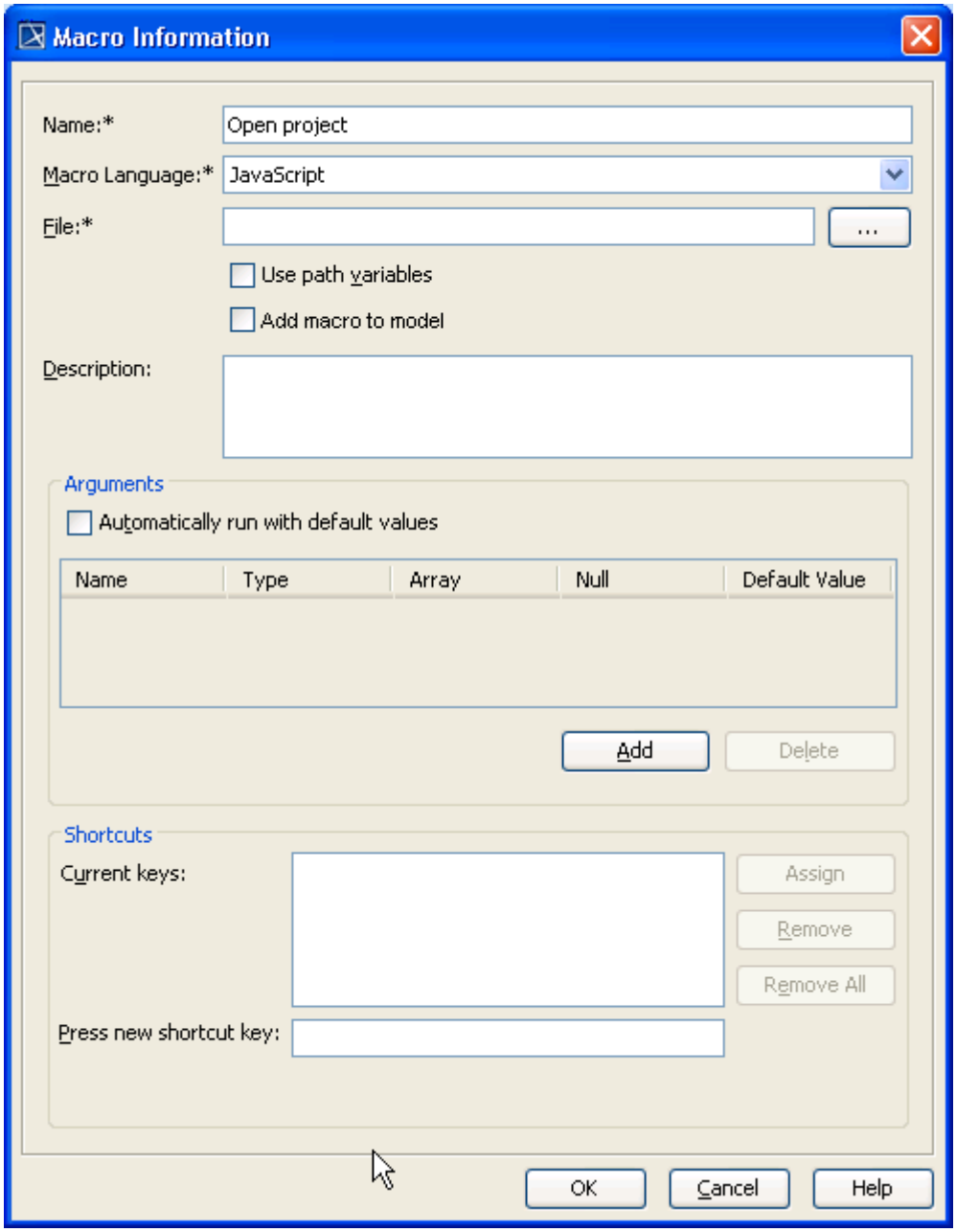

Figure 7 -- The Macro Information Dialog

3. Type the macro name in the **Name** box.
4. The default macro language you have previously selected (see **2.1 Selecting a Default Macro Language**) will appear in the **Macro Language** box (Figure 7).
5. Click the **...** button to locate a macro file. The **Open file** dialog will open (Figure 8).
6. Select the file and its type (there are 5 types of file filter: ***.bsh**, ***.groovy**, ***.rb**, ***.js**, or ***.py**) (Figure 9).

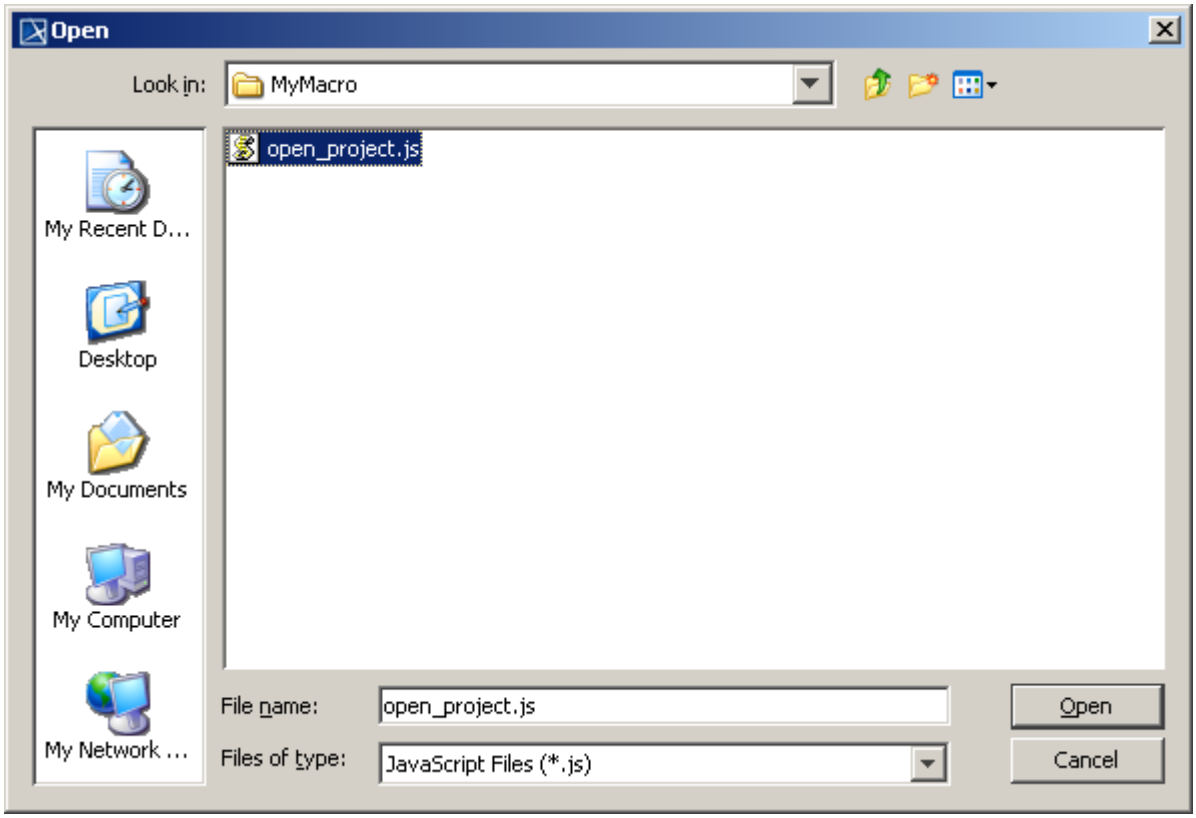

Figure 8 -- Use Path Variables Check Box

7. Click **Open** (Figure 9). The selected pathname will appear in the **File** box in the **Macro Information** dialog (Figure 10).

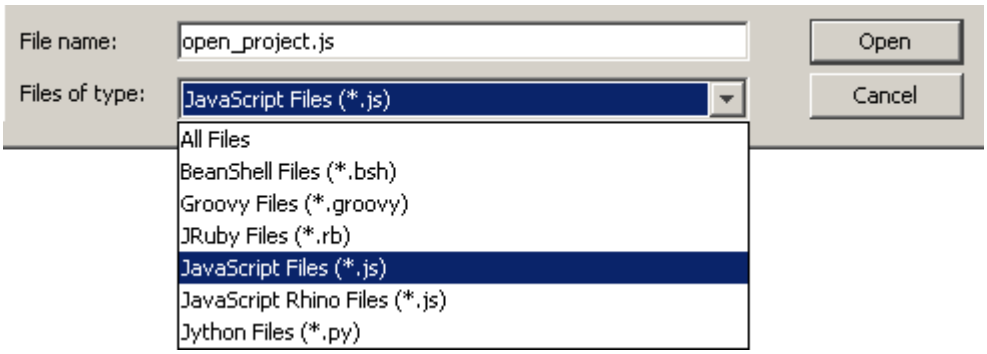

Figure 9 -- Types of File Filter

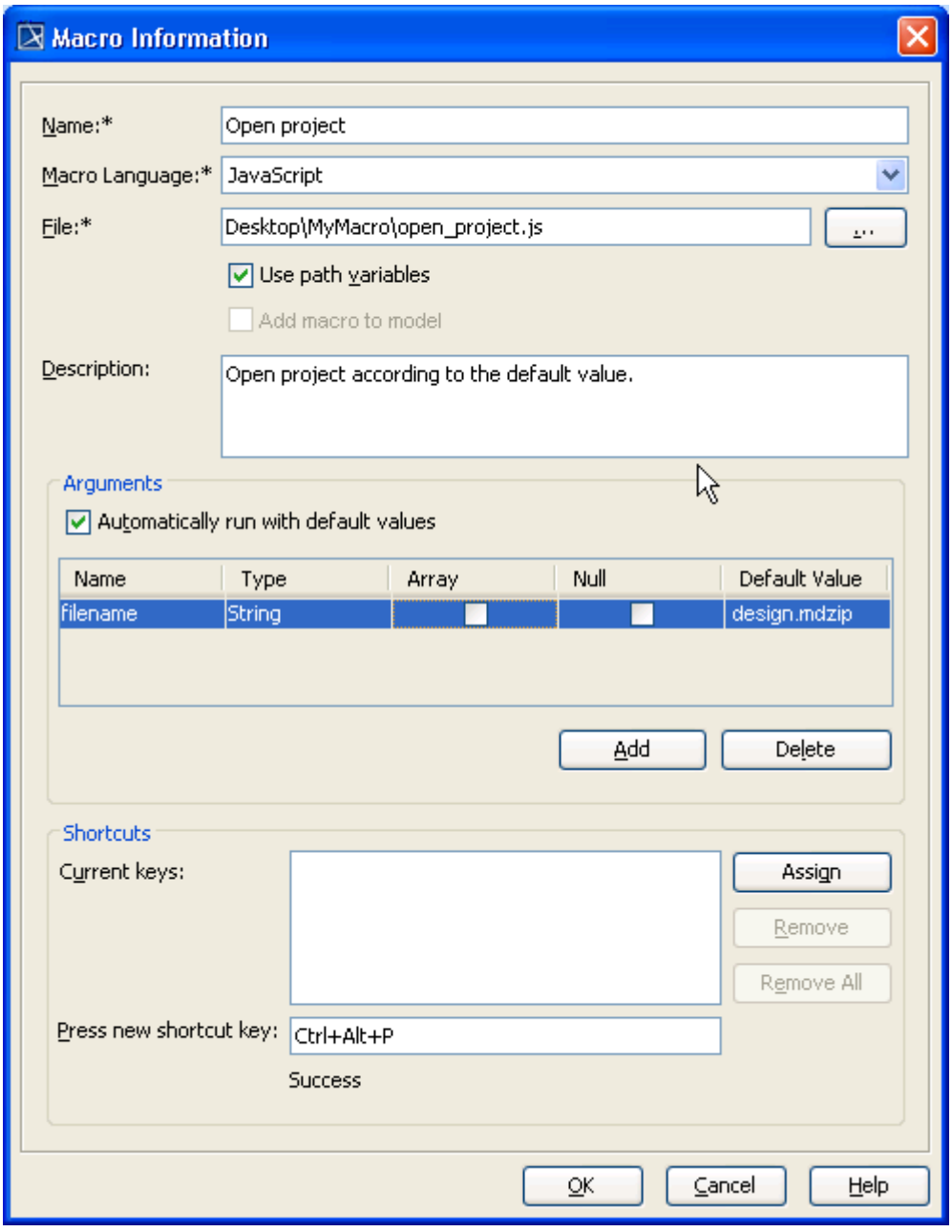

Figure 10 -- Macro Information Dialog

8. Select either (i) the **Use path variables** check box or (ii) the **Add macro to model** check box (Figure 10).

NOTE:	• If you have specified the file or network path in the **Environment Options** dialog by clicking **Options** > **Environment** > **Path Variables** and selected the **Use path variables** check box in **Macro Information** dialog, the `<Path Variable name>` will show in front of the file pathname. This field is the [Required] field, for example, `<mypath>/<macro_name>.js`. • If you select the **Use path variables** check box, the full pathname will not be saved. • If you select the **Add macro to model** check box, your source code will be imported from the file to the model. The location to keep the model is `Data::MacroEngine`. • You can open the **Macro Information** by using Ctrl+Alt+m as a shortcut key.

9. Type the macro description in the **Description** box (Figure 10).

NOTE:	A macro name must be unique and cannot be duplicated.

10. The function of the **Automatically run with default values** check box (Figure 10) is to allow you to run the default values automatically. If you select the check box, the system will not open a dialog to prompt you to input the value.

NOTE:	If you select the **Automatically run with default values** check box, you need to enter the valid default value of each parameter.

11. Click **Add** to specify the arguments of the macro. The arguments specified in the **Arguments** table will be the global variables of a specific macro.

    - **Name** column contains the name of a parameter
    - **Type** column contains the type of a parameter
    - **Array** check box is to identify if an array is the parameter
    - **Null** check box is to identify if null is the parameter value
    - **Default Value** column contains an initial value to run the macro

NOTE:	• A parameter type can be a String, Integer, Long, Double, Date, or ElementPath. • If the **Null** check box is selected, you cannot enter the default value of that particular parameter. • An empty value in the **Default Value** column does not necessarily mean a null value, for example an empty string value is an empty string. • If you input an invalid default value or you do not enter the argument name, the system will display the following error message when you run the script: **The following argument(s) are invalid: <List of the invalid argument...>**.

12. Type a keyboard shortcut that will be used to run the macro in the **Press new shortcut key** box and click **Assign**. The newly assigned keyboard shortcut will appear in the **Current keys** box.
13. Click **OK**. The **Organize Macros** dialog will open, showing the newly added macro name, description, and keyboard shortcut.

NOTE:	• Macro names, filenames, and languages are required. • Macro description and keyboard shortcuts are optional. • If any of the required fields are not entered or duplicate macro name is entered, the following message will open: **The following field(s) are invalid: <List of the problems>**. • If the **Automatically run with default values** check box is selected and at least one value is empty or not valid, then the following error message will open: **The following value(s) are invalid. <List of the invalid value...>** • If the **Automatically run with default values** check box is selected, all variables must have valid values or are set to null.

14. Click **Close** to close the **Organize Macros** dialog.

2.3.3 Editing Macro Information

You can see macro information such as names and descriptions, as well as the macro keyboard shortcuts in the **Organize Macros** dialog.

To edit macro information:

1. Click **Tools** > **Macros** > **Organize Macros...** . The **Organize Macros** dialog will open (Figure 11).

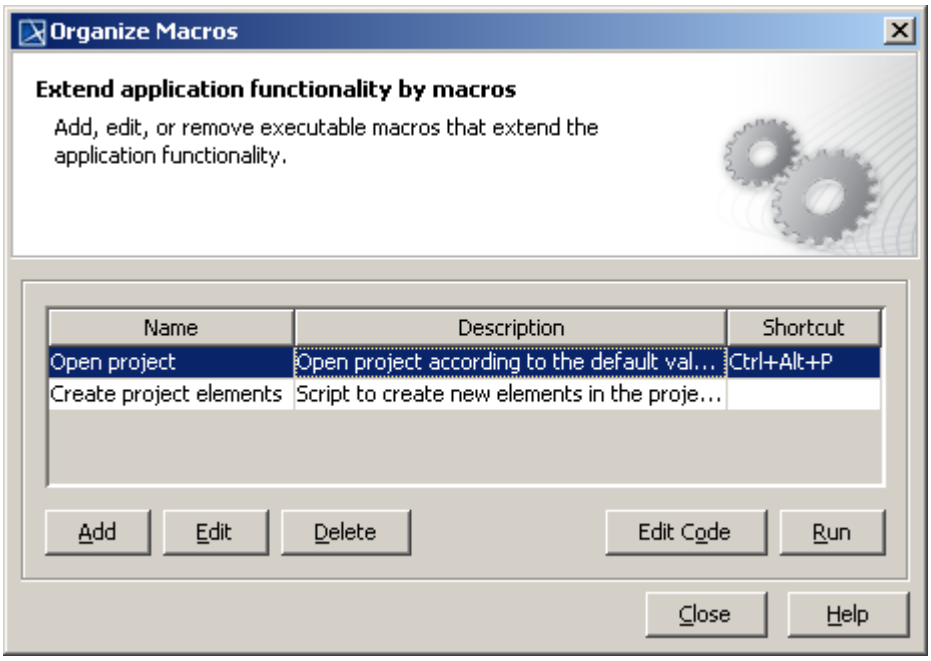

Figure 11 -- Editing a Macro

2. Select a macro from the table and either click **Edit** or press **Alt** + **E**. The **Macro Information** dialog will open.
3. Edit the macro information.
4. Click **OK** to save it. You will see the altered information in the **Organize Macros** dialog.
5. Click **Close**.

2.3.4 Macro Information Dialog Mnemonic Keys

Macro Engine provides mnemonic keys for you to perform some operations, for example, highlight a textbox and click a specific button in the **Macro Information** dialog. Table 1 below lists the **Macro Information** dialog mnemonic keys and their function.

Table 1 -- Macro Information Dialog Mnemonic Keys

Mnemonic keys	Function
Alt + N	To place the pointer in the **Name** box.
Alt + M	To highlight the **Macro Language** box.
Alt + F	To place the pointer in the **File** box.
Alt + .	To open the **File** dialog.
Alt + D	To place the pointer in the **Description** box.
Alt + A	To add an argument in the **Arguments** area.
Alt + I	To delete an argument in the **Arguments** area.
Alt + U	To highlight the **Current keys** box.
Alt + G	To click the **Assign** button.
Alt + R	To click the **Remove** button.
Alt + E	To click the **Remove All** button.
Alt + P	To place the pointer in the **Press new Shortcut key** box.
Alt + O	To click the **OK** button.
Alt + C	To click the **Cancel** button.
Alt + H	To click the **Help** button.

2.4 Deleting and Executing Macros

You can click the **Delete** or **Run** button in the **Organize Macros** dialog to (2.4.1) delete or (2.4.2) execute a selected macro. You can also press the predesigned mnemonic keys to delete or run a macro (see section **2.4.3 Organizing Macros Dialog Mnemonic Keys** below).

2.4.1 Deleting a Macro

To delete a macro from the **Organize Macros** dialog:

1. Click **Tools** > **Macros** > **Organize Macros...** to open the **Organize Macros** dialog.
2. Select a macro from the table and either click **Delete** or press **Alt** + **D**. A dialog will open, asking whether you want to delete the macro (Figure 12).

Figure 12 -- Deleting Macro

3. Click **Yes** and the macro will be deleted from the **Organize Macros** dialog.

2.4.2 Executing a Macro

To execute a macro from the **Organize Macros** dialog:

1. Click **Tools** > **Macros** > **Organize Macros...**. The **Organize Macros** dialog will open (Figure 13).

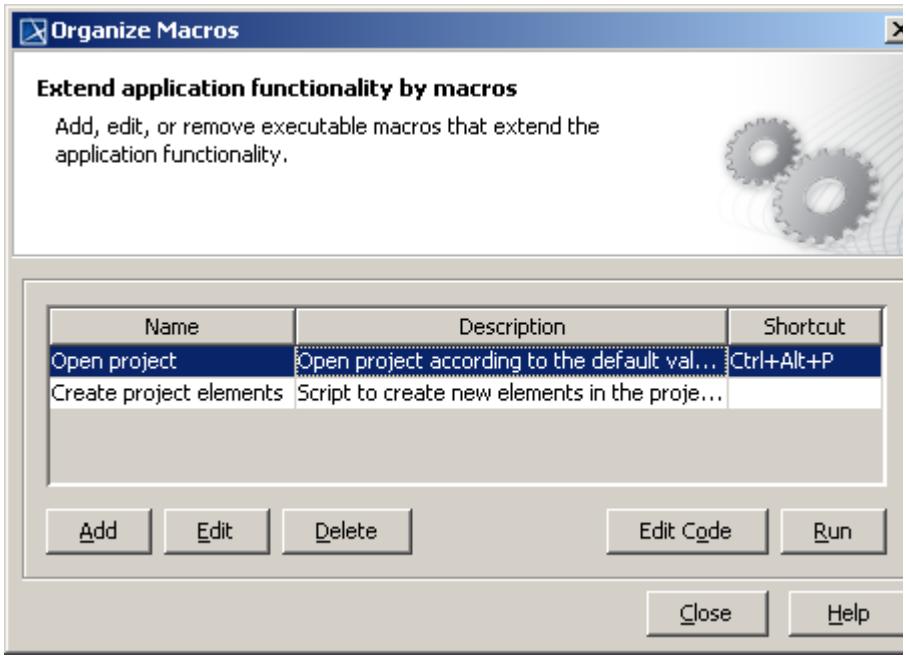

Figure 13 -- Executing Macro

2. Select a macro from the table and click **Run**. After the macro has been executed, a message will open: **The macro <macro name> has been executed**.
3. If you have the parameters in the **Macro Information** dialog, you need to specify the value in the **Macro Arguments** dialog (Figure 15) before running the macro.

NOTE:	• For an array datatype, you need to click the "**...**" button in the **Value** column in the **Macro Arguments** dialog and enter each value into each line. The value of the first line will be the value in array index 0. • An ElementPath is a Qualified Name. You can find this information in the specification dialog of each element (Figure 14). • The **Macro Arguments** dialog (Figure 15) will be displayed if the **Automatically run with default values** check box is not selected. • If you want to save argument values in the **Macro Arguments** dialog, you need to select the **Set as default values** check box before you click **OK**.

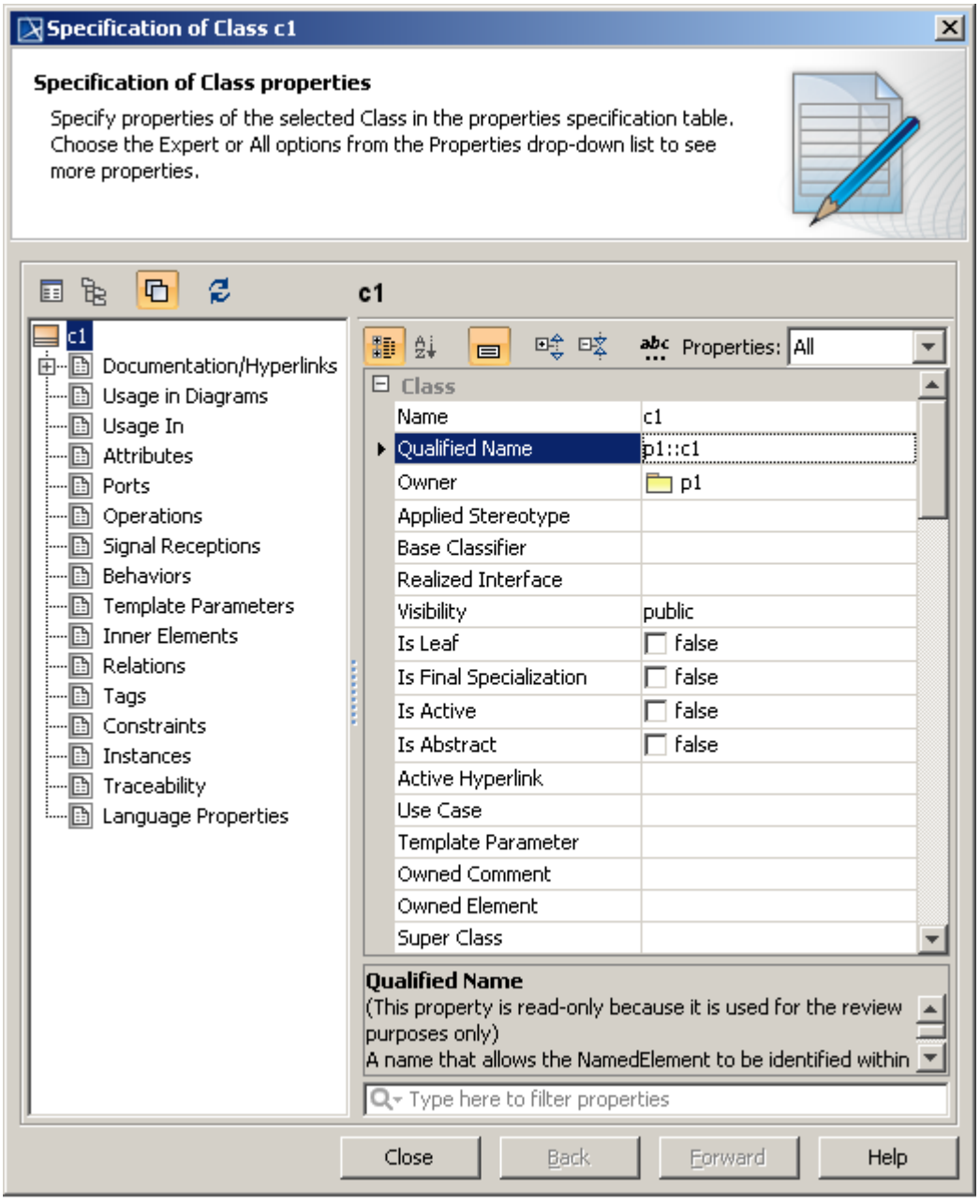

Figure 14 -- Specification Dialog Example

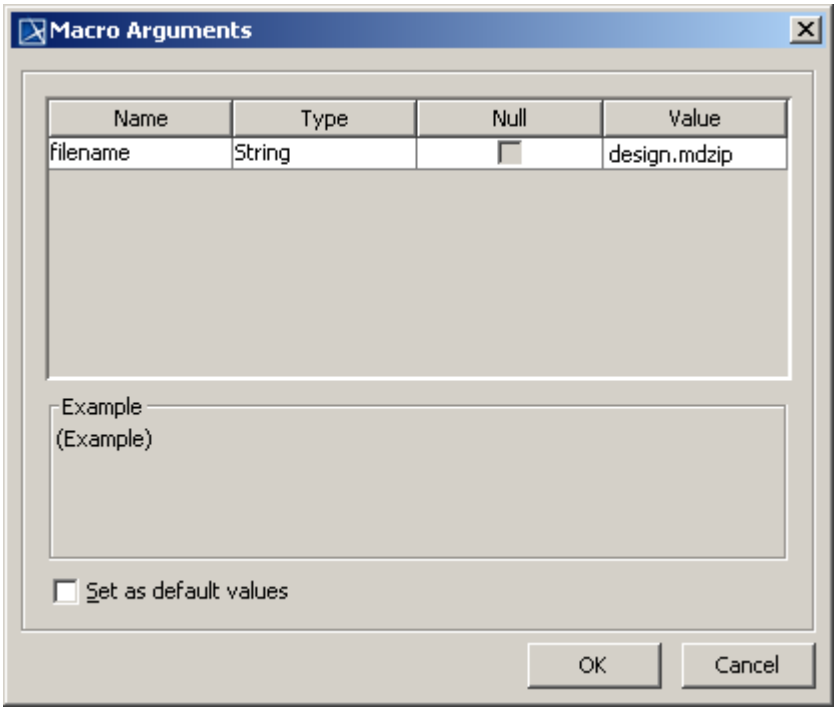

Figure 15 -- Macro Arguments Dialog

NOTE:	• You can also execute a macro from the main browser in MagicDraw by pressing the shortcut keys that you have defined in the **Organize Macros** dialog. • You can only run macro one at a time. • If there is an error while running a macro, for example, syntax error, the following message will open: **MagicDraw cannot execute the <macro language> macro, please make sure that <path, filename, extension> is correct. <error description>**. • If MagicDraw cannot find a macro file in the location that you have specified in the **Open** dialog, the following message will open: **MagicDraw cannot find the macro: <path, filename, extension>**.

2.4.3 Organizing Macros Dialog Mnemonic Keys

Macro Engine also provides mnemonic keys to add, edit, delete, and run a macro from the **Organize Macros** dialog. Table 2 shows the **Organize Macros** dialog mnemonic keys and their function.

Table 2 -- Organize Macros Dialog Mnemonic Keys

Mnemonic keys	Button	Function
Alt + **A**	**Add**	To add a macro in the **Macro Information** dialog.
Alt + **E**	**Edit**	To edit a macro in the **Macro Information** dialog.
Alt + **D**	**Delete**	To delete a macro from the **Organize Macros** dialog.
Alt + **O**	**Edit Code**	To edit source code in **Macro Editor**.
Alt + **R**	**Run**	To run a macro from the **Organize Macros** dialog.
Alt + **C**	**Close**	To click the **Close** button.
Alt + **H**	**Help**	To click the **Help** button.

NOTE:	
	• You can click the **Edit Code** button in the **Organize Macros** dialog (Figure 13) to edit and save source code in the **Macro Editor** dialog (Figure 16). • You can click **Save** to save the source code or click **Run** to run the macro in the **Macro Editor** dialog (Figure 16).

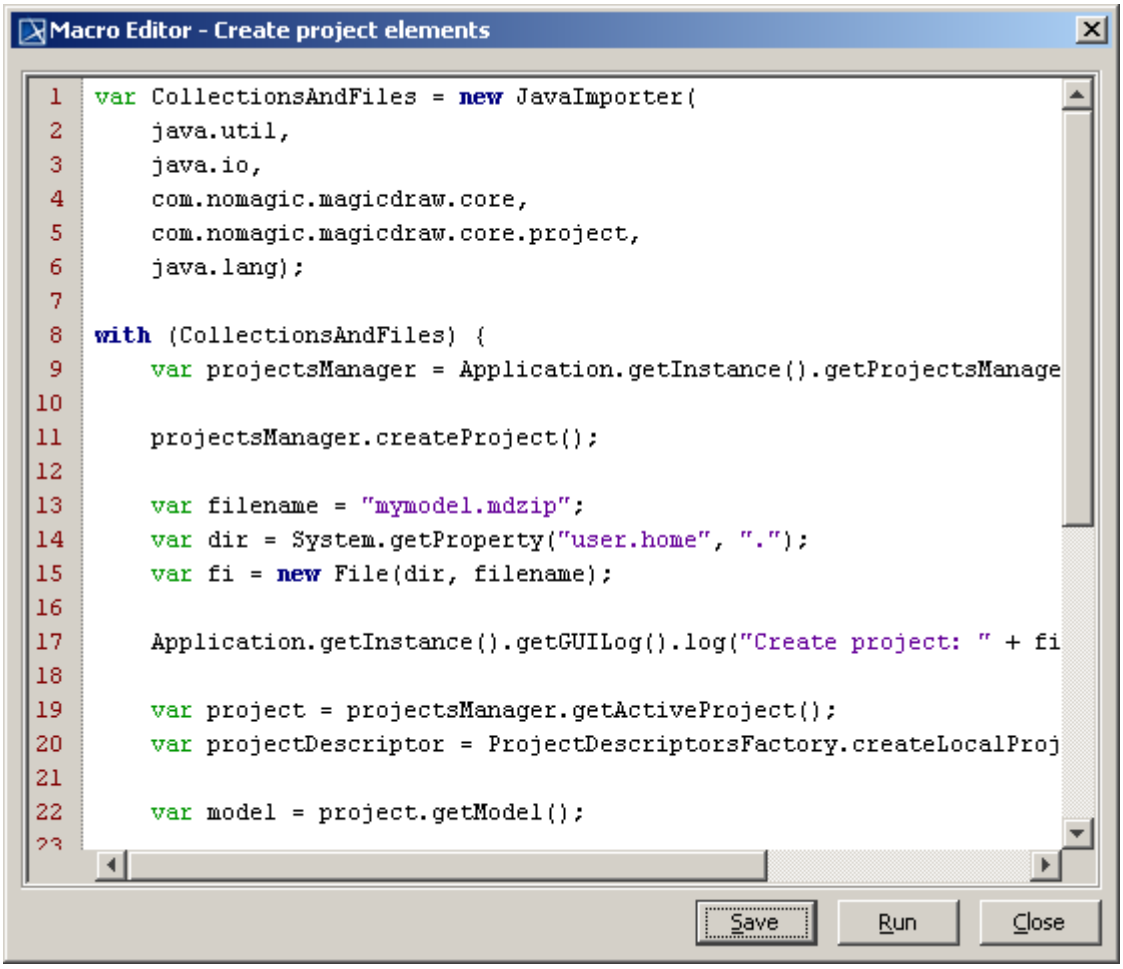

Figure 16 -- Macro Editor

2.5 Macro Keyboard Shortcuts

Macro Engine allows you to assign keyboard shortcuts to the macros that you have created using either the **Environment Options** (Figure 17) or **Macro Information** dialog (Figure 18). You can later press the keyboard shortcuts to execute or run the macros in MagicDraw.

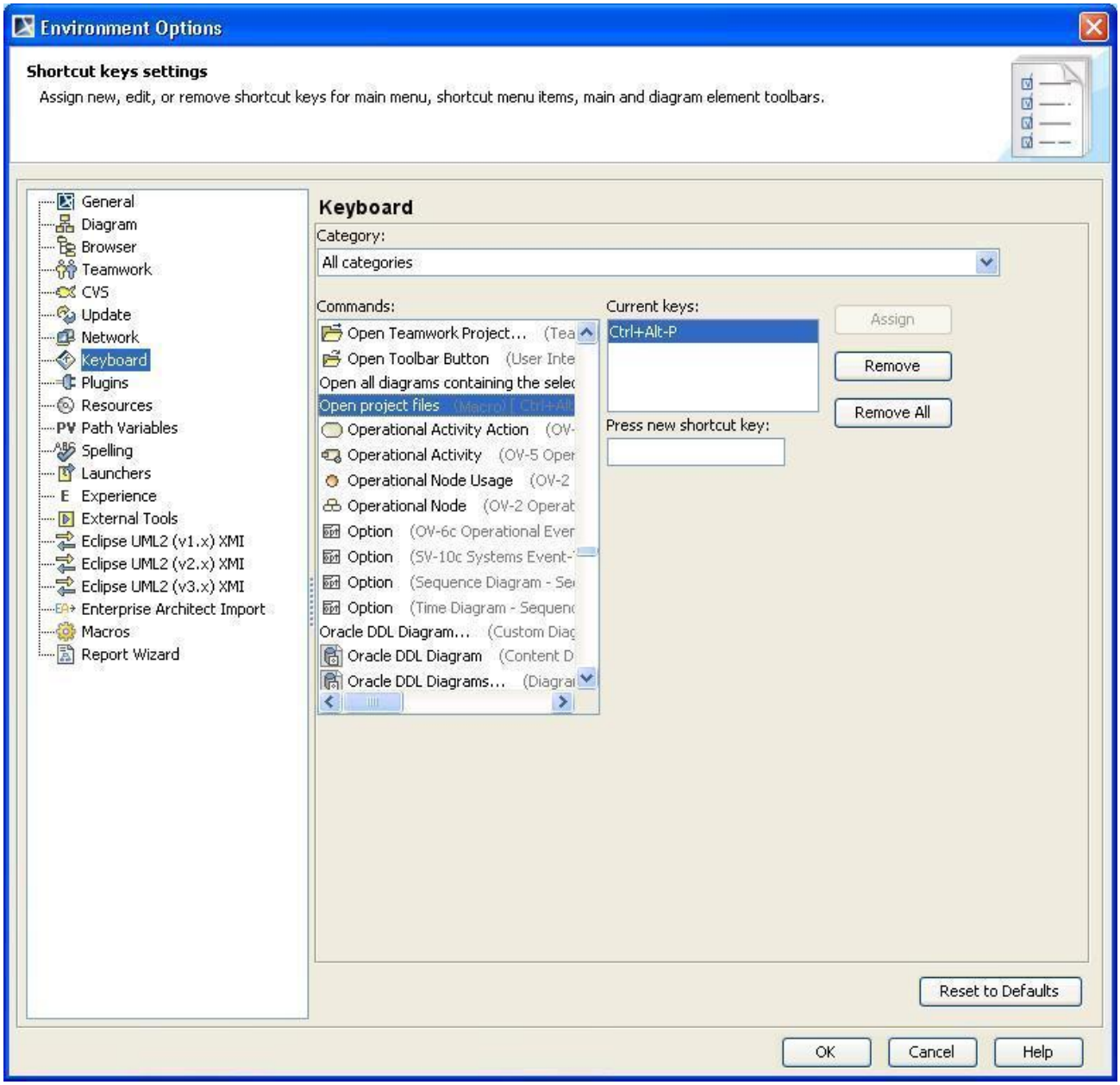

Figure 17 -- Environment Options Dialog

To open the keyboard shortcuts pane in the **Environment Options** dialog:

1. Click **Options** > **Environment** on the MagicDraw main menu. The **Environment Options** dialog will open.
2. Click **Keyboard** (Figure 17).

The macro information and keyboard shortcuts that are saved either in the **Environment Options** or **Macro Information** dialog will be stored as a MagicDraw environment. They will not be kept in a specific project file [*.mdzip].

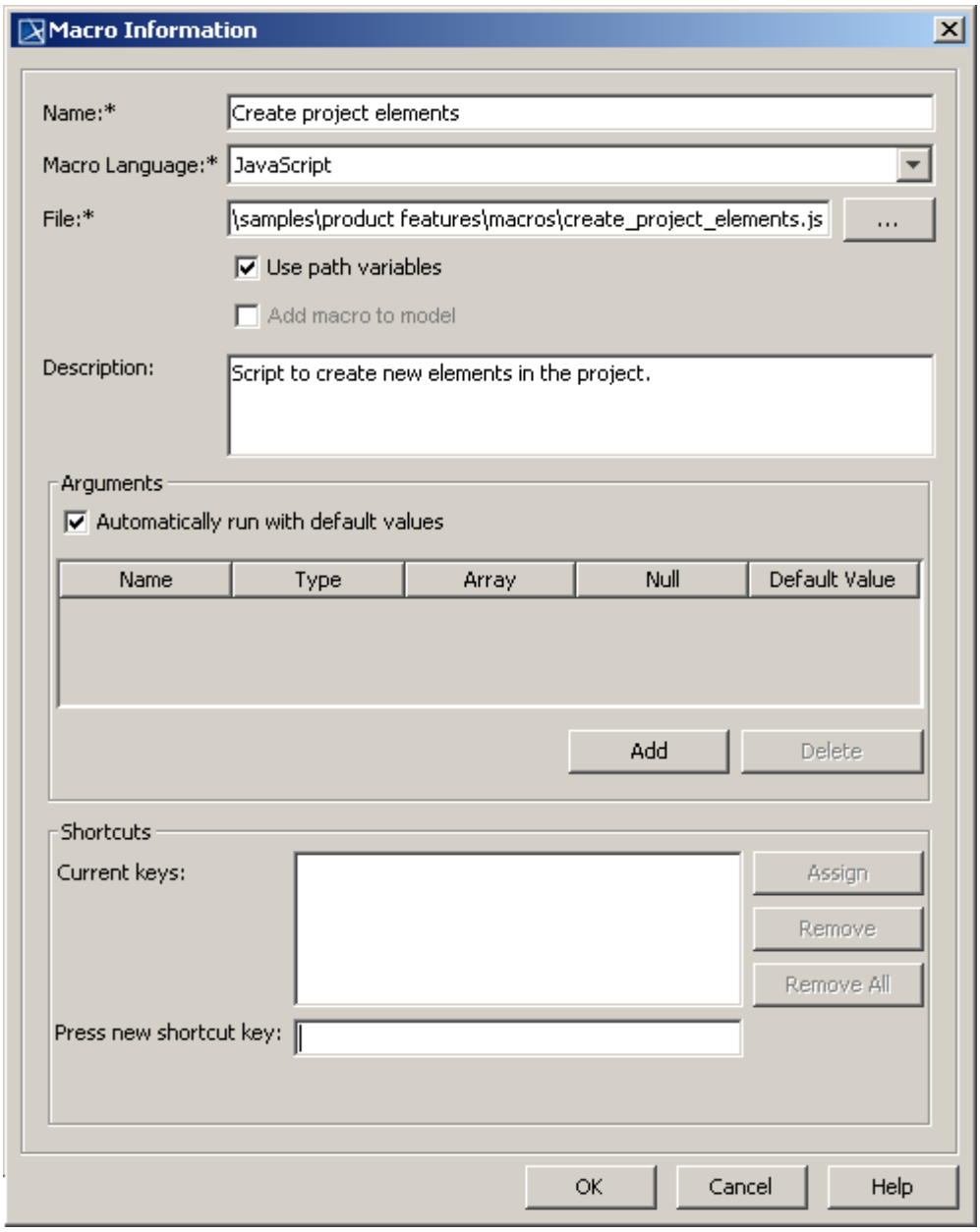

Figure 18 -- Macro Information Dialog

NOTE:	• The **Assign** button in the **Macro Information** or **Environment Options** dialog will be disabled until you type a keyboard shortcut in the **Press new shortcut key** box.

Table 3 -- Keyboard Shortcut Buttons

Button	Function
Assign	To assign a new keyboard shortcut to a macro.
Remove	To remove the keyboard shortcuts from the selected item in **Current keys** box.
Remove All	To remove all keyboard shortcuts from a macro.

Table 4 -- Keyboard Shortcut Text Boxes

Field	Function
Current keys	To store a list of keyboard shortcuts currently assigned to a macro.
Press new shortcut key	To type a new keyboard shortcut to be assigned to a macro.

2.5.1 Assigning a Keyboard Shortcut to a Macro

To add a new keyboard shortcut to a macro:

1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Type a keyboard shortcut in the **Press new shortcut key** box.
3. Click **Assign** to assign the keyboard shortcut to a macro.
4. Click **OK**.

NOTE:	If a keyboard shortcut key entered in the **Press new shortcut key** box has already been assigned to another macro, the following message will appear under the **Press new shortcut key** box: `Currently assigned to <the command name>`.

2.5.2 Removing Keyboard Shortcuts from Macro

To remove a keyboard shortcut:

1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Select a keyboard shortcut from the **Current keys** box.
3. Click **Remove**. The selected keyboard shortcut will be removed from the **Current keys** box.

NOTE:	You can also remove a keyboard shortcut through the **Environment Options** dialog by clicking **Options** > **Environment** > **Keyboard** on MagicDraw main menu.

To remove all keyboard shortcuts:

1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Click **Remove All**. All keyboard shortcuts will be deleted from the **Current keys** box.

NOTE:	• The **Assign**, **Remove**, and **Remove All** buttons in the **Macro Information** or **Environment Options** dialog will be disabled if there is no keyboard shortcut either in the **Press new shortcut key** or **Current keys** box. • The **Assign** button will be enabled if the new keyboard shortcut entered has not been assigned to any other macro. • You can assign more than one keyboard shortcut to a macro.

2.6 Opaque Objects

Macro Engine creates opaque objects to represent the elements in MagicDraw. Through these opaque objects, you can access the elements, retrieve, or assign values to them instead of using MagicDraw OpenAPI to do it.

NOTE:	All examples given in this section is written in Javascript.

2.6.1 Getting an Opaque Object

You can get an opaque object of an existing MagicDraw element by using either:

(i) `AutomatonMacroAPI.getOpaqueObjectByPath(String path)`

(ii) `AutomatonMacroAPI.getOpaqueObject(Element element)`

If the above methods cannot find the element, they will return null.

(i) getOpaqueObjectByPath(String path)

To use `getOpaqueObjectByPath(String path)`, for example, type:

```
AutomatonMacroAPI.getOpaqueObjectByPath ("PackageA::Element2");
```

(ii) getOpaqueObject(Element element)

To use `getOpaqueObject(Element element)`, for example, type:

```
var element = ModelHelper.findElementWithPath("PackageB::ClassB");
var a = AutomatonMacroAPI.getOpaqueObject(element);
```

You can also use two other methods to get an opaque object as follows:

(iii) `AutomatonMacroAPI.getModelData()`

(iv) `AutomatonMacroAPI.getSelectedElementFromContainmentTree()`

(iii) getModelData()

This method will obtain an opaque object of the model **Data** in the Containment tree.

(iv) getSelectedElementFromContainmentTree()

This method will obtain an opaque object of the selected element in the Containment tree.

Macro Engine uses methods (iii) **getModelData()** and (iv) **getSelectedElementFromContainmentTree()** to retrieve opaque objects in order to identify the defined scope in its recording mechanism.

2.6.2 Getting Element Property Values

Once you have obtained an opaque object, you can get the property value of the element by using any of the following methods:

(i) `<variableName>.get<PropertyName>()`

(ii) `<variableName>.<PropertyName>`

(iii) `<variableName>._automatonGetValue(String realPropertyName, String stereotypePath)`

The `<variableName>` is the name of a macro variable that stores the opaque object. The `<PropertyName>` is the name of the property that appears in the Specification dialog.

You need to capitalize the first letter of `<PropertyName>` and replace the whitespace with an underscore. If a duplicate property name occurs, you can refer to the right property name by using the following additional information: `<SterotypeName>_<PropertyName><RunningNumber>`.

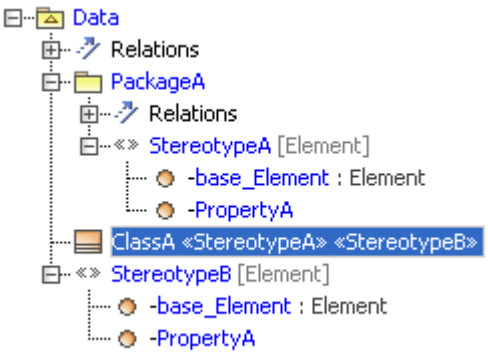

Figure 19 -- A Duplicate Property in Stereotype

If there are two stereotypes applied to the same element (Figure 19), you can differentiate one from the other, for example, by specifying `<PropertyName>` as `StereotypeA_PropertyA1` and `StereotypeA_PropertyA2` in the macro.

You can also use the method `_automatonGetValue` to get a property value. If you want to get the value of a PropertyA from SterotypeA in PackageA., for example, you can use `_automatonGetValue("PropertyA", "PackageA::StereotypeA")`.

A `realPropertyName` is the real property name that is used in MagicDraw openAPI. A `stereotypePath` is the path of a stereotype that contains the property. This property will not be needed if it is in the Element itself.

If you refer to a property that does not exist, Macro Engine may or may not throw an error, depending on which language library you use. For example, if you use Javascript to call the property that does not exist, Macro

Engine will not throw an error. But if you use JRuby, it will throw an exception to report the error condition: `org.jruby.exceptions.RaiseException`.

If the value of a property is an element, Macro Engine will convert it to an opaque object and you can call it, for example, by typing `classA.Owner.Name`.

If a property has more than one value, Macro Engine will convert the values to a list of opaque objects. If you need to get a value from the list, you can call the method of the class `java.util.List`, for example, by typing:

(i) `classA.appliedStereotype.get(<index>).Name`

or

(ii) `classA.appliedStereotype.add(anotherOpaque)`

If a property is read-only, an exception will be thrown.

2.6.2.1 Getting Element Property Value Examples

The following are some examples of how to get an element property value using the methods given in section 2.6.2 above:

- to use `get<PropertyName>`, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath ("MyClass");
Application.getInstance().getGUILog().log(classA.getName());
```

- `to use <PropertyName>`, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("MyClass");
Application.getInstance().getGUILog().log(classA.Name);
```

- to use `_automatonGetValue`, for example, type:

```
var reqA = AutomatonMacroAPI.getOpaqueObjectByPath("MyRequirements");
Application.getInstance().getGUILog().log(reqA._automatonGetValue("Pr
opertyA", "PackageA::StereotypeA");
```

- to use a SysML Element, for example, type:

```
var reqA = AutomatonMacroAPI.getOpaqueObjectByPath("MyRequirements");
Application.getInstance().getGUILog().log(reqA.getID());
```

2.6.3 Setting Element Property Values

You can assign values to a MagicDraw element by using any of the following methods:

(i) `<variableName>.set<PropertyName>(Object value)`

(ii) `<variableName>.<PropertyName> = value;` and then call `persist()`

(iii) `<variableName>._automatonSetValue(String realPropertyName, String stereotypePath, Object value)`

The value of an element can be a primitive data type, an opaque object, or an element. If you use a setter to set the value, for example, `_automatonSetValue()` or change the value on a list, it will be saved in the MagicDraw element automatically.

If you use `<variableName>.<PropertyName> = value` to set the value, you must call `persist()` to persist the change to the MagicDraw element. You need to first set the data, call `persist()`, and finally call a getter method in order to set the data and retrieve them. This process will force an opaque object to retrieve the current value from a MagicDraw model and overwrite the value that you have just specified in the opaque object.

NOTE:	If you use JRuby, do not capitalize the first letter of `<PropertyName>` in `<variableName>.<PropertyName>`.

2.6.3.1 Setting Element Property Value Examples

The following are some examples of how to set an element property value by using the methods given in section 2.6.3 above.

- to use `set<PropertyName>(value)`, for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB.setName("NewElementName")
```

- to use <PropertyName> = value), for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB.Name = "NewElementName";
classB.Is_Abstract = true;
classB.persist();
```

- to use _automatonSetValue, for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB._automatonSetValue("PropertyA", "PackageA::StereotypeA", "Demo String value");
```

- to set an opaque object to another opaque object, for example, type:

```
var ele1 = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
var ele2 = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
ele1.setPackaged_Element(ele2);
```

Table 5 below lists the supported element properties in Macro Engine.

Table 5 -- Supported Element Properties

Property	Type	Support Operation
Active Hyperlink	String	Read
All General Classifiers	List<AutomatonOpaqueObject>	Read
All Realizing Elements	List<AutomatonOpaqueObject>	Read
All Specific Classifiers	List<AutomatonOpaqueObject>	Read
All Specifying Elements	List<AutomatonOpaqueObject>	Read
Applied Stereotype	List<AutomatonOpaqueObject>	Read
Applied_Stereotype_Instance	N/A	Read
Attribute	List<AutomatonOpaqueObject>	Read
Base Classifier	List<AutomatonOpaqueObject>	Read
Class	AutomatonOpaqueObject	Read
Classifier Behavior	AutomatonOpaqueObject	Read
Client Dependency	List<AutomatonOpaqueObject>	Read
Collaboration Use	List<AutomatonOpaqueObject>	Read
Element ID	N/A	-
Extension	List<AutomatonOpaqueObject>	Read
Feature	List<AutomatonOpaqueObject>	Read
Generalization	List<AutomatonOpaqueObject>	Read
Image	N/A	-
Imported Member	List<AutomatonOpaqueObject>	Read
Inherited Member	List<AutomatonOpaqueObject>	Read
Interface Realization	List<AutomatonOpaqueObject>	Read
Is Leaf	Boolean	Read/Write
Is Final Specialization	Boolean	Read/Write
Is Active	Boolean	Read/Write
Is Abstract	Boolean	Read/Write
Member	List<AutomatonOpaqueObject>	Read
Name	String	Read/Write
Name Expression	AutomatonOpaqueObject	Read
Namespace	AutomatonOpaqueObject	Read
Nested Classifier	List<AutomatonOpaqueObject>	Read
Owned Attribute	List<AutomatonOpaqueObject>	Read
Owned Connector	List<AutomatonOpaqueObject>	Read
Owned Comment	List<AutomatonOpaqueObject>	Read
Owned Diagram	List<AutomatonOpaqueObject>	Read
Owned Element	List<AutomatonOpaqueObject>	Read
Owned Member	List<AutomatonOpaqueObject>	Read
Owned Operation	List<AutomatonOpaqueObject>	Read

Owned Port	List<AutomatonOpaqueObject>	Read
Owned Reception	List<AutomatonOpaqueObject>	Read
Owned Rule	List<AutomatonOpaqueObject>	Read
Owned Template Signature	AutomatonOpaqueObject	Read
Owned Trigger	List<AutomatonOpaqueObject>	Read
Owned Use Case	List<AutomatonOpaqueObject>	Read
Owning Package	List<AutomatonOpaqueObject>	Read
Owning Template Parameter	N/A	-
Owner	AutomatonOpaqueObject	Read/Write
Package	AutomatonOpaqueObject	Read
Package Import	List<AutomatonOpaqueObject>	Read
Part	List<AutomatonOpaqueObject>	Read
Participates In Activity	List<AutomatonOpaqueObject>	Read
Participates In Interaction	List<AutomatonOpaqueObject>	Read
Powertype Extent	List<AutomatonOpaqueObject>	Read
Qualified Name	List<AutomatonOpaqueObject>	Read
Realized Interface	N/A	-
Realizing Component	List<AutomatonOpaqueObject>	Read
Realizing Element	List<AutomatonOpaqueObject>	Read
Redefined Classifier	List<AutomatonOpaqueObject>	Read
Redefined Element	List<AutomatonOpaqueObject>	Read
Redefinition Context	List<AutomatonOpaqueObject>	Read
Representation	N/A	-
Role	List<AutomatonOpaqueObject>	Read
Specific Classifier	List<AutomatonOpaqueObject>	Read
Specifying Component	List<AutomatonOpaqueObject>	Read
Specifying Element	List<AutomatonOpaqueObject>	Read
Supplier Dependency	List<AutomatonOpaqueObject>	Read
Template Binding	List<AutomatonOpaqueObject>	Read
Template Parameter	N/A	-
To Do	String	Read/Write
Use Case	List<AutomatonOpaqueObject	Read/Write
Visibility	String	Read/Write

2.6.4 Getting the Child of an Element

You can get the child of an Element by typing the following:

- `<variableName>._getChild(String childElementName)`

If the above method cannot find the childElementName, it will throw an error.

To get the child of an element named ClassB from ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var classB = classA._getChild("ClassB");
```

2.6.5 Getting the Owner of an Element

You can get the owner of an element by typing the following:

- `<variableName>._getOwner()`

If the above method cannot find the owner, for example Model Data, it will throw an error.

To get the owner of an element named ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var classB = classA._getOwner();
```

2.6.6 Creating a New Element

You can create a new element by using its Meta-class name, such as a Class in Standard UML or a Requirement in SysML by using the following method:

- `AutomatonMacroAPI.createElement(String metaClassName)`

This method will return an opaque object of the created element. If the method cannot find the Meta-class, it will throw an exception.

To create a new Class & Requirement element (Javascript), for example, type:

```
AutomatonMacroAPI.createElement("Class");
AutomatonMacroAPI.createElement("Requirement");
```

2.6.7 Creating a Relationship Between Elements

You can create a relationship between elements by typing:

- `AutomatonMacroAPI.createRelationship(AutomatonOpaqueObject opque1, AutomatonOpaqueObject opque2, String relationName)`

To create a Copy & Abstraction relationship between Element1 and Element2 (Javascript), for example, type:

```
var a = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
var b = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
AutomatonMacroAPI.createRelationship(a, b, "Copy");
AutomatonMacroAPI.createRelationship(a, b, "Abstraction");
```

2.6.8 Removing an Element

You can remove an Element from MagicDraw by typing the following:

- `AutomatonMacroAPI.removeElement(AutomatonOpaqueObject opaqueObj)`

To remove an Element1 (Javascript), for example, type:

```
var a = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
AutomatonMacroAPI.removeElement(a);
```

2.6.9 Adding a Stereotype to an Element

You can apply a stereotype to an element by typing either:

(i) `AutomatonMacroAPI.addStereotype(AutomatonOpaqueObject opaque, AutomatonOpaqueObject opaqueStereotype)`

(ii)`AutomatonMacroAPI.addStereotypeByStereotypeName(AutomatonOpaqueObject opaque, String stereotypeName)`

To add a StererotypeA to ClassA (Javascript), for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var stereotypeA = AutomatonMacroAPI.getOpaqueObjectByPath("StereotypeA");
AutomatonMacroAPI.addStereotype(classA, stereotypeA);
```

2.6.10 Removing a Stereotype from an Element

You can remove a Stereotype from an element by typing either:

(i) `AutomatonMacroAPI.removeStereotype(AutomatonOpaqueObject opaque, AutomatonOpaqueObject opaqueStereotype)`

(ii)`AutomatonMacroAPI.removeStereotypeByStereotypeName(AutomatonOpaqueObject opaque, String stereotypeName)`

To remove a StereotypeA from ClassA (Javascript), for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var stereotypeA = AutomatonMacroAPI.getOpaqueObjectByPath("StereotypeA");
AutomatonMacroAPI.removeStereotype(classA, stereotypeA);
```

2.6.11 Printing Element Details

If you want to know the method that is used in the opaque object of an element, you can print the element details by typing the following:

- `AutomatonMacroAPI.printElementDetail(AutomatonOpaqueObject opaque);`

This method will print the field, constructor, and method that belong to the opaque object in the MagicDraw **Message** dialog.

To print the details of ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
AutomatonMacroAPI.printElementDetail(classA);
```

2.7 Recording Macros

Macro Engine has the capability to record changes in a model. It uses opaque objects to generate macros and record them. This capability is especially useful when you want to redo some of your repetitive tasks.

The following is a list of actions that you can record:

- Creating UML, Stereotype, and DSL elements
- Creating relationships between UML, Stereotype, or DSL elements

NOTE:	You cannot move the element defined as a record scope during recording.

To record a macro:

1. Click **Tools** > **Macros** > **Record Macro...** on the MagicDraw main menu. The **Record Macro** dialog will open (Figure 20).

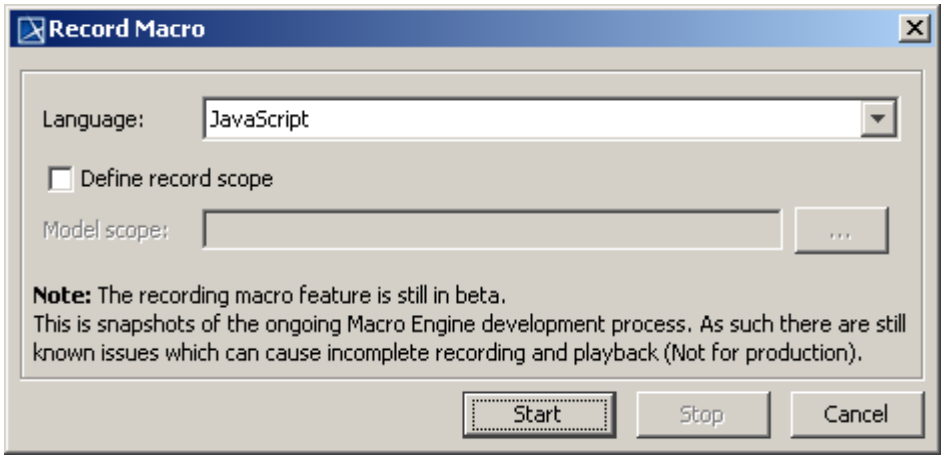

Figure 20 -- Record Macro Dialog

2. Select a macro language in the **Language** box (you will see the default macro language that you have previously selected in that particular box (see **2.1 Selecting a Default Macro Language**).
3. Select the **Define record scope** check box and click the **Model Scope** "**...**" button to locate a scope in the Containment tree. The generated macros will later record the change in the element by using a relative path that refers to the defined scope.
4. Click **Start** to start recording.
5. Work with the model in the scope you have defined.
6. Click **Stop** to stop recording. The **Record Macro** dialog will close and the **Create Macro** dialog will open, showing the recorded macros (Figure 21).

NOTE:	• If you do not open a project, the menu **Tools** > **Macros** > **Record Macro...** will be disabled.
	• You can select a record scope only before you start recording
	• You cannot change a record scope during recording.
	• If you do not define a record scope, the model **Data** will become the record scope.
	• If you click the **Cancel** button, the **Record Macro** dialog will be closed.
	• If you click the **Start** button, it will be changed to **Pause** and the **Stop** button will enabled.
	• You cannot alter the **Language**, **Define record scope**, or **Model Scope** options after you click the **Start** button.
	• If you click the **Pause** button, the recording will pause and the button will be changed to **Resume**.
	• If you click the **Resume** button, the following things will happen:
	  ◦ the recording will continue
	  ◦ the button will be changed to **Pause**
	• The recording mechanism of Macro Engine can generate code for Beanshell, Groovy, Javascript (Nashorn and Rhino), JRuby, and Jython. Unsupported languages will be filtered out of the **Language** box.

7. Click either **Save** or **Run**.

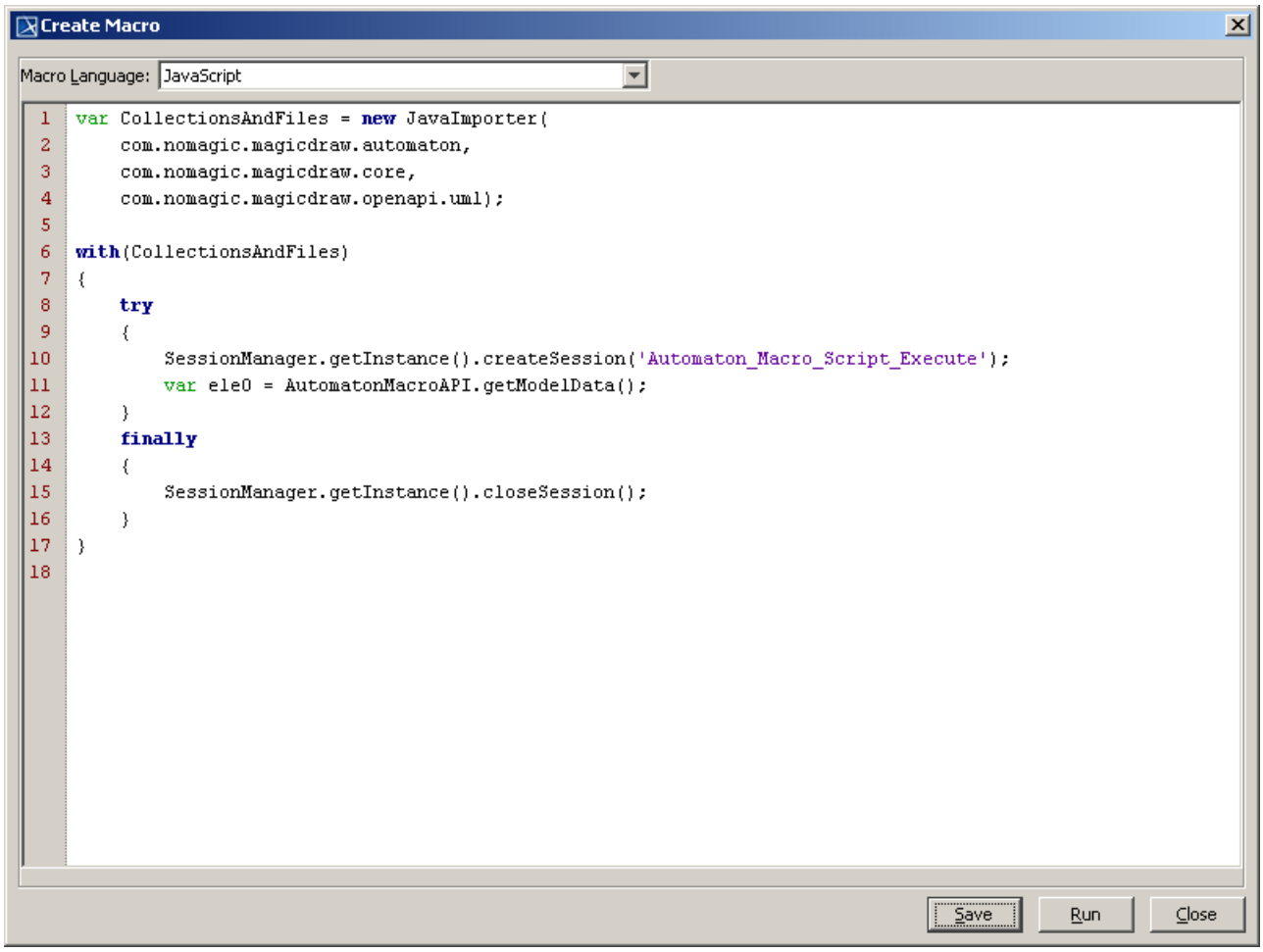

Figure 21 -- The Create Macro Dialog after Successful Recording

3. Appendix

Macro Engine supports five scripting languages: (i) BeanShell, (ii) JavaScript (Nashorn and Rhino), (iii) JRuby, (iv) Jython, and (v) Groovy.

(i) BeanShell

BeanShell is a lightweight scripting language for Java. The shipped version is 2.1.7. The advantage of using BeanShell is that its syntax is compatible with Java; therefore, you can use the code assistant feature in most Java IDE. The BeanShell syntax documentation is available at `http://www.beanshell.org/docs.html`.

(ii) JavaScript

JavaScript is a scripting language based on Java syntax. The documentation is available at:

`http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/index.html` and `https://developer.mozilla.org/en/Rhino_documentation`.

(iii) JRuby

JRuby is a 100% pure-Java implementation of the Ruby programming language. Macro Engine uses JRuby 1.7.11 The documentation is available at `http://kenai.com/projects/jruby/pages/Home`.

(iv) Jython

Jython, a successor of JPython, is a Python programming language implemented in Java. The documentation is available at `http://wiki.python.org/jython`. Besides automating MagicDraw by using Jython scripting, the plugin development in the Jython programming language is also supported in MagicDraw. For more information on Jython, see JPython Scripting in MagicDraw OpenAPI User Guide.

(v) Groovy

Groovy is an agile and dynamic language for the Java Virtual Machine. Macro Engine uses Groovy 2.0.1. The advantage of using Groovy is that its Java-like syntax seamlessly integrates with the existing Java source code and libraries. It supports many IDEs that provide code completion and debugging. BeanShell scripts can be easily moved to Groovy, with some modifications. For more information on Groovy, go to `http://groovy.codehaus.org`.

3.1 Using Code Completion to Develop BeanShell Scripts

You can use any text editor to develop a scripting language. However, a standard text editor lacks of code assistant features. Most scripting languages are loose type. For example, to define a variable in JavaScript, you need to type:

```
var a;
```

It is difficult to determine what type of "`a`" later in the source code. With BeanShell, you can use a variable without declaring it, for example:

```
a = new File("file.txt");
```

Or you can declare it first:

```
File a = new File("file.txt");
```

Java IDE does not officially support code completion for scripting languages. However, there is a workaround if you use BeanShell. First you need a Java IDE. If you do not have one, you can select NetBeans because it has the smallest file size. You can download the latest version of NetBeans at `http://www.netbeans.org/downloads/index.html`. The **NetBeans Java SE** package is enough. Second you need to set up a MagicDraw classpath.

To set up a classpath point in the MagicDraw library in NetBeans:

1. Click **Tools** > **Libraries** on the main menu. The **Library Manager** dialog will open.
2. Click the **New Library** button. The **New Library** dialog will open.
3. Specify a library name, for example, MD16.6. The Library type must be **Class Libraries**.
4. Click **OK** to close the **New Library** dialog.
5. Select your new library in the Libraries tree.
6. Click the **Add JAR/Folder...** button and add all the JAR files in **<MagicDraw>/lib**.
7. Click **OK** to close the **Library Manager** dialog.

To develop a BeanShell script in NetBeans:

1. Click **File** > **New Project** on the main menu to create a Java application project. The **New Project** dialog will open.
2. Select **Java** in the **Categories** box and **Java Application** in the **Projects** box, and then follow the instructions.
3. Expand your project node in the **Project** window. The Libraries node will appear.
4. Right-click the Libraries node and select **Add Library** from the shortcut menu.
5. Select the MagicDraw library that you have previously created (see "To set up a classpath point in the MagicDraw library in NetBeans:") and click **Add Library**.
6. Click **File** > **New File** on the main menu to add a new Java file.
7. Select **Java** in the **Categories** box and **Empty Java File** in the **File Types** box, and then follow the instructions until finish.

You need to create a public static method in a Java file, for example, `main()method`, to follow the standard Java programming language. At the end of the file, insert a statement to call the static method. See the example in **create_project_elements.bsh** in the **<MagicDraw>/samples/product features/macros** directory.

NOTE:	The official filename extension for BeanShell is **.bsh**. However, you can add a **.java** file to the BeanShell scripting language.

3.2 Using NetBeans IDE to Develop Groovy Scripts

You will need NetBeans IDE 6.7.1 or 6.8 with Groovy Plugin to develop Groovy scripts. The Groovy support comes with NetBeans "Java" and "All" package. If your NetBeans does not have Groovy, download the plugin through NetBeans Plugin Manager. For more information about adding a new plugin, click **IDE Basics** > **Plugins** > **Updating the IDE** on the NetBeans **Help** menu.

Use the following three simple steps to develop and run a Groovy script in NetBeans:

1. Set up a classpath.
2. Develop a Groovy script.

1. To set up a classpath:

1. Click **Tools** > **Libraries** on the main menu. The **Library Manager** dialog will open.
2. Click the **New Library** button. The **New Library** dialog will open.
3. Specify a library name, for example, MD16.8. The Library Type must be Class Libraries.
4. Click **OK** to close the **New Library** dialog.
5. Select your new library in the Libraries tree.
6. Click the **Add JAR/Folder...** button and add all the JAR files in <MagicDraw>/lib.
7. Repeat steps 2 to 6 to add the Groovy library that is in <MagicDraw>/plugins/com.nomagic.magicdraw.automaton/engines/groovy-2.0.1/embeddable/groovy-all-2.0.1.jar.
8. Click **OK** to close the **Library Manager** dialog.

2. To develop a Groovy script:

1. Click **File** > **New Project** on the main menu to create a Java application project. The **New Project** dialog will open.
2. Select **Java** in the **Categories** box and **Java Application** in the **Projects** box, and then click **Next**.
3. Choose a project location. Be sure that you do not select **Create Main Class**.
4. Click **Finish**.
5. Expand your project node in the **Project** window. The Libraries node will appear.
6. Right-click the Libraries node and select **Add Library** from the shortcut menu.
7. Click **File** > **New File** on the main menu to add a new Groovy file.
8. Select Groovy in the **Categories** box and Groovy Script in the **File Types** box.
9. Follow the instructions until finish.

3.3 Using Eclipse to Develop Groovy Scripts

Use the following three simple steps to develop and run a Groovy script in NetBeans:

1. Install Groovy-Eclipse Plugin.
2. Develop a Groovy script.

1. To install Groovy-Eclipse Plugin:

1. Go to `http://groovy.codehaus.org/Eclipse+Plugin`.
2. Follow the instructions for installing Groovy-Eclipse Plugin.

2. To develop a Groovy Script:

1. Create a Groovy project.
2. Right-click the project in Package Explorer and click **Properties**.
3. Click Java Build Path in the tree on the left-hand side. Click the **Libraries** tab on the right pane.
4. Click **Add External JARs** to add all the JAR files in <MagicDraw>/lib and in <MagicDraw>/plugins/com.nomagic.magicdraw.automaton/engines/groovy-2.0.1/embeddable/groovy-all-2.0.1.jar.
5. Click **OK** to close the dialog.
6. Create a new Groovy class in the project. Remove a class declaration, and then put the script there.

3.4 Installing Gems for JRuby

To install a gem for Ruby engine inside MagicDraw:

1. Open command line.
2. Go to folder <MagicDraw folder>\plugins\com.nomagic.magicdraw.automaton\engines").
3. Enter the following command to install a gem.

```
java -jar jruby-complete-<version>.jar -S gem install [--user-
install] <gem name1> <gem name2> ...
```

NOTE:	• <version> is the version of the JRuby, for example, 1.5.1 or 1.6.7.2. • <gem name> is the name of a gem. • The three dots (...) mean that you can type a list of gem names there.

Once the gem has been installed, you can use it in MagicDraw Macro Engine, for example:

```
require 'java'
require 'rubygems'
require 'uuid'

Application = com.nomagic.magicdraw.core.Application
uuid = UUID.new
Application.getInstance().getGUILog().log(uuid.generate);
```

The above example shows how to create a macro that can generate a unique ID by using a gem.

To install a gem for the existing Ruby environment on your machine and use it in Macro Engine:

1. Add the following properties to specify the home and library of JRuby in <MagicDraw>/bin/mduml.properties under JAVA_ARGS section (assuming you have JRuby installed in **C:/jruby-1.5.3** on your machine).

```
-Djruby.home\="C:/jruby-1.5.3" -Djruby.lib\="C:/
jruby-1.5.3/lib”
```

For example:

```
...
JAVA_ARGS=-Xmx800M -XX\:PermSize\=40M -
XX\:MaxPermSize\=150M -Djruby.home\="C:/jruby-1.5.3"
-Djruby.lib\="C:/jruby-1.5.3/lib”
```

2. Change the JRuby library path to navigate to your jruby.jar in <MDdir>/plugins/com.nomagic.magicdraw.automaton/plugin.xml.

```
...
<library name="c:/jruby-1.5.3/lib/jruby.jar"/>
```

NOTE:	• The command in this section should run in <MDdir>/plugins/com.nomagic.magicdraw.automaton/engine. • If you have a whitespace in the property file path, you need to type a double quote to wrap both the property and its value, for each property, for example: "-Djruby.home=C:/my white space/jruby-1.5.3" "-Djruby.lib=C:/my white space/jruby-1.5.3/lib"

3.5 Adding a Scripting Language to MagicDraw

You can add a new scripting language that supports JSR-223 to MagicDraw by editing the Macro Engine's plugin.xml file.

To add a new scripting language:

1. Open the plugin.xml file in **<MagicDraw>/plugins/com.nomagic.magicdraw.automaton/plugin.xml**.
2. Add the following jar file tags to the runtime section and save the file.

```
...
<library name="<path to jar file>"/>
```

The path to the jar file is relative to **<MagicDraw>/plugins/com.nomagic.magicdraw.automaton** or the absolute pathname can be used, for example, if you want to add Sleep programming language, add the following tag to the runtime section:

```
...
<library name="c:/sleep_2.1.jar"/>
```

A slash (/) is used as a directory separator for all platforms, including Windows. The new language will be displayed in most of the Macro Engine dialogs, for example, in the Macros section in the **Environment Options** dialog, the **Create Macro** dialog, and in the **Macro Information** dialog. The language name is automatically configured from the information inside the jar file. This version of Macro Engine does not allow you to modify the language name.

3.5.1 Script Filename Extension Filter

When you open an **Open** file dialog to browse for a script file in Macro Engine, you can find the filename extension for the script file in the dialog file filter. If you have added a new scripting language to MagicDraw, for

example, Sleep programming language, Macro Engine will add the filename extension (*.sl)" in the **Open** file dialog (Figure 22).

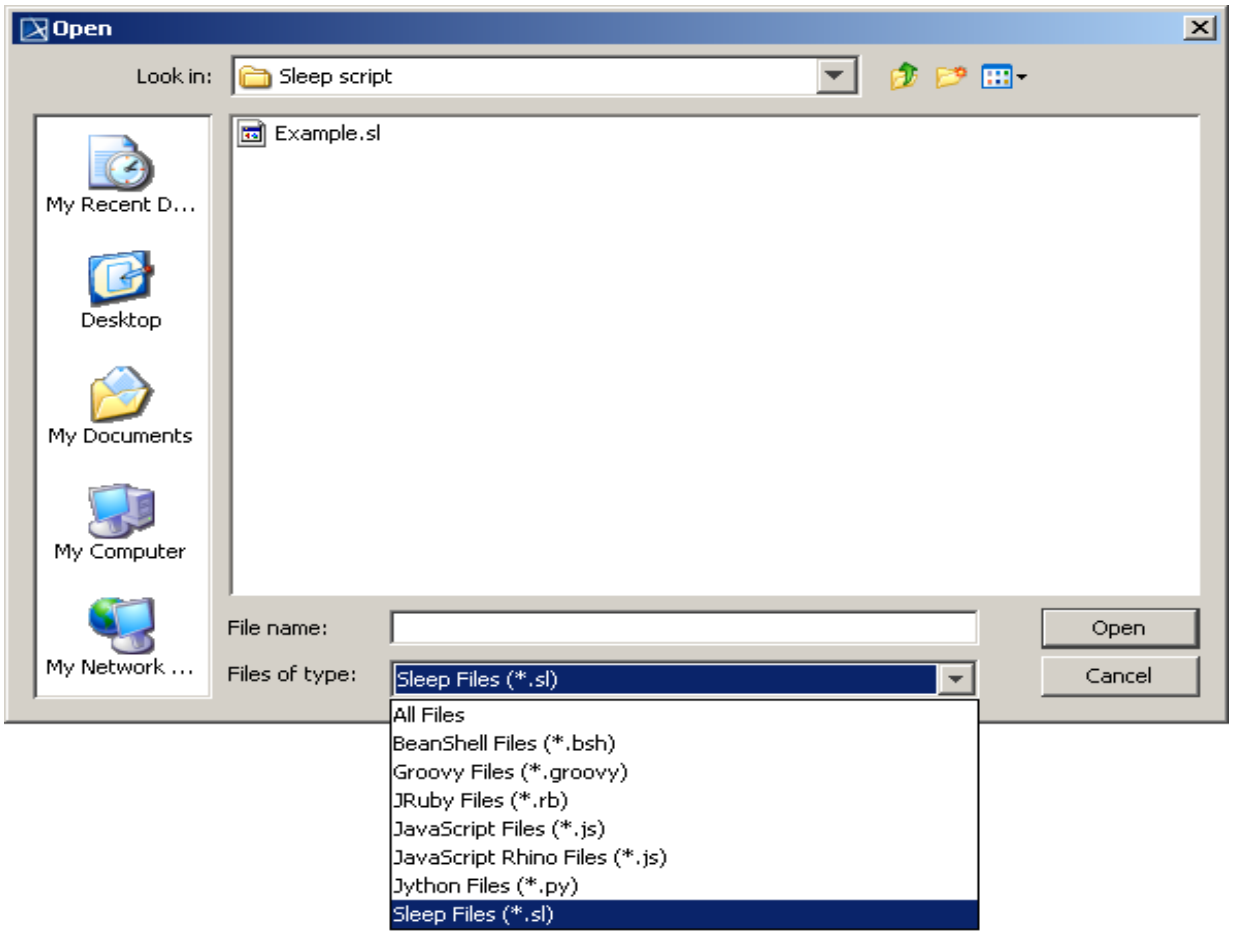

Figure 22 -- New Script Extension in File Type Filter

The filename extensions are derived from jar files. There can be more than one filename extension for each engine. However, the current Macro Engine version only shows the first filename extension that is queried from the engine jar file.

NOTE:	• You can remove a scripting language from Macro Engine by removing the associated library tag in the plugin.xml file. Once the tag has been removed, the language will be removed from all of the dialogs in Macro Engine. However, the script configured to be used with the language will not be removed from MagicDraw. The language of the script will be reset to the default language, which has been previously configured in the MagicDraw **Environment Options** dialog.