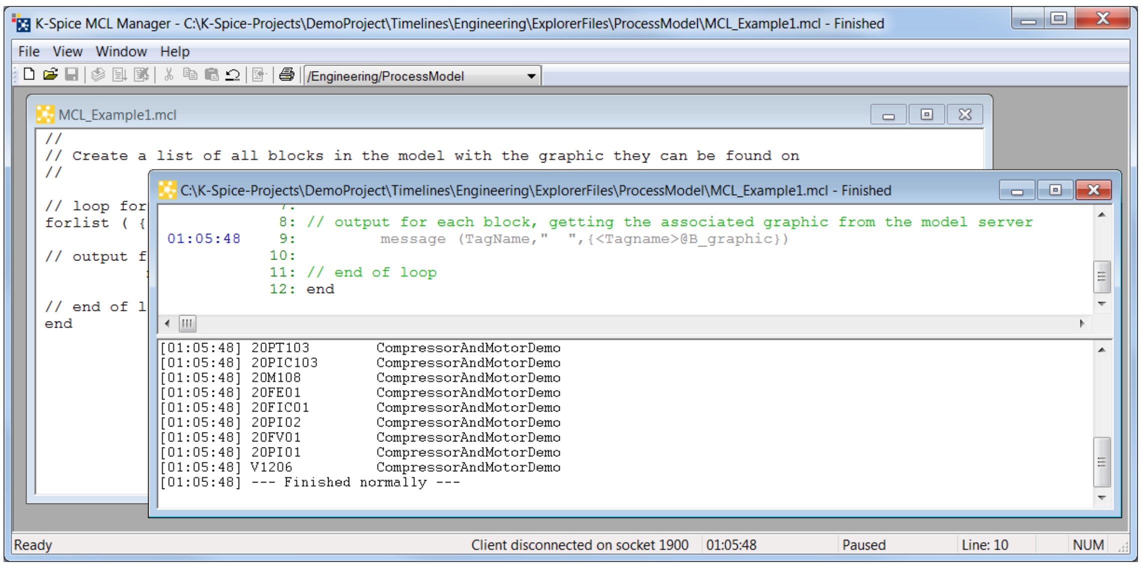
KONGSBERG

KONGSBERG K-Spice® MCL Manager Help

Release 3.2


K-Spice MCL Manager - C:\K-Spice-Projects\DemoProject\Timelines\Engineering\ExplorerFiles\ProcessModel\MCL_Example1.mcl - Finished

File View Window Help

/Engineering/ProcessModel

MCL_Example1.mcl

```
//
// Create a list of all blocks in the model with the graphic they can be found on
//

// loop for
forlist ( {

// output f

// end of l
end
```

C:\K-Spice-Projects\DemoProject\Timelines\Engineering\ExplorerFiles\ProcessModel\MCL_Example1.mcl - Finished

```
             // output for each block, getting the associated graphic from the model server
01:05:48            message (TagName,"   ",{<Tagname>@B_graphic})

             // end of loop
             end
```

```
[01:05:48] 20PT103          CompressorAndMotorDemo
[01:05:48] 20PIC103         CompressorAndMotorDemo
[01:05:48] 20M108           CompressorAndMotorDemo
[01:05:48] 20FE01           CompressorAndMotorDemo
[01:05:48] 20FIC01          CompressorAndMotorDemo
[01:05:48] 20PI02           CompressorAndMotorDemo
[01:05:48] 20FV01           CompressorAndMotorDemo
[01:05:48] 20PI01           CompressorAndMotorDemo
[01:05:48] V1206            CompressorAndMotorDemo
[01:05:48] --- Finished normally ---
```

Ready | Client disconnected on socket 1900 | 01:05:48 | Paused | Line: 10 | NUM

371574/I

Document history

Document number: 371574		
Rev. A	August 2008	First version.
Rev. B	August 2008	XML version from old documentation.
Rev. C	February 2011	Updated help file for K-Spice build 44.
Rev. D	March 2011	Update with new menu entries. Added tool bar and status bar details.
Rev. E	April 2012	Update to version 2.2.
Rev. F	July 2012	Added additional detail for Insert Statement. Updated screen shots, Minor edits.
Rev. G	December 2012	Updated for release 2.4
Rev. H	September 2014	Minor corrections, update for new version
Rev. I	October 2015	Update for release 3.2

The reader

This publication is intended as a guide for the K-Spice® MCL Manager user.

Note

Comments

To assist us in making improvements to the product and to this guide, we welcome comments and constructive criticism.

e-mail: kogt.documentation@kongsberg.com

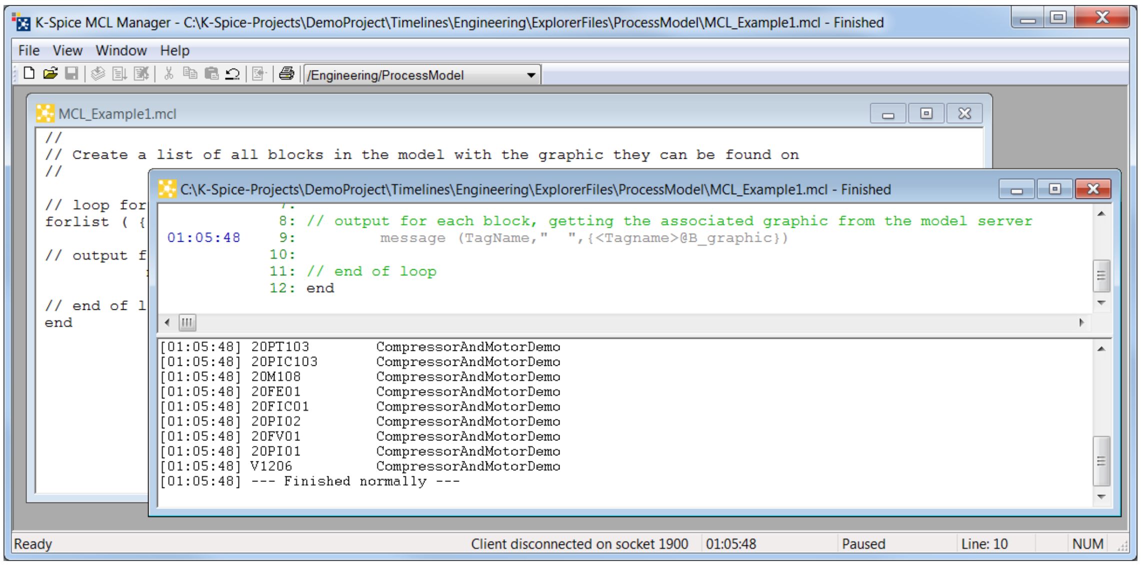
Table of contents

1 Introduction

Main Window

The MCL Manager is a configuration editor for building simple scenarios. The scenarios can be checked for correctness and saved for later use. The MCL Manager appears as a separate Windows application. See the example provided below:

Figure 1 MCL Manager – Scenario Configuration Editor


K-Spice MCL Manager - C:\K-Spice-Projects\DemoProject\Timelines\Engineering\ExplorerFiles\ProcessModel\MCL_Example1.mcl - Finished
File View Window Help
/Engineering/ProcessModel
MCL_Example1.mcl
C:\K-Spice-Projects\DemoProject\Timelines\Engineering\ExplorerFiles\ProcessModel\MCL_Example1.mcl - Finished

```
//
// Create a list of all blocks in the model with the graphic they can be found on
//

// loop for
forlist ( {

// output f

// end of l
end
```

```
             8: // output for each block, getting the associated graphic from the model server
01:05:48     9:         message (TagName,"   ",{<Tagname>@B_graphic})
            10:
            11: // end of loop
            12: end
```

```
[01:05:48] 20PT103         CompressorAndMotorDemo
[01:05:48] 20PIC103        CompressorAndMotorDemo
[01:05:48] 20M108          CompressorAndMotorDemo
[01:05:48] 20FE01          CompressorAndMotorDemo
[01:05:48] 20FIC01         CompressorAndMotorDemo
[01:05:48] 20PI02          CompressorAndMotorDemo
[01:05:48] 20FV01          CompressorAndMotorDemo
[01:05:48] 20PI01          CompressorAndMotorDemo
[01:05:48] V1206           CompressorAndMotorDemo
[01:05:48] --- Finished normally ---
```

Ready | Client disconnected on socket 1900 | 01:05:48 | Paused | Line: 10 | NUM

The screen shown above is the fully-featured MCL Manager that is available for use with K-Spice.

The application has two types of sub-windows:

- **The editor window** – This window contains the script code.
- **The monitor window** – This window shows the script progress.

2 Menu Commands

The following menus are available from MCL Manager:

- **File**
- **Edit**
- **View**
- **Window**
- **Help**

2.1 File

The following commands are available from the *File* menu:

`New` – Create a new script.

`Open` – Open an existing script.

`Close` – Close a script.

`Close Finished Monitors` – Closes all completed monitor windows.

`Save/Save as` – Save the current script.

`Run` – Run the currently selected script.

`Parse` – Parse the script to ensure that it is correct, but do not run it.

`Print` – Print the current script.

`Print Preview` – Preview the current script before printing.

`Print Setup` – Standard printer options.

`Recent MCL files` – Shortcuts for the most recent MCL files opened.

`Exit` – Close the program.

2.2 Edit

The following commands are available from the *Edit* menu:

Tip

This menu is only visible when an editor window is active.

`Undo` – Undo the last change.

`Cut` – Remove the currently selected text to the clipboard.

`Copy` – Copy the currently selected text to the clipboard.

`Paste` – Paste the clipboard contents at the current location.

`Select All` – Select the whole MCL script.

`Insert Data Item` – Insert an application data item using the Insert Data Item dialog.

`Insert Statement` – Insert an MCL statement using the Insert/Edit Statement dialog.

`Find` – Opens the find dialogue.

`Find Next` – Finds and selects the next 'find' match in the active script window.

`Replace` – Opens the replace dialogue.

2.3 View

The following commands are available from the *View* menu:

`Toolbar` – Show or hide the MCL Manager toolbar.

`Status Bar` – Show or hide the MCL Manager status bar.

`Options`

- *Statement editor:* Enter a data item list file for the Statement Editor

2.4 Window

The following commands are available from the *Window* menu:

Tip

This menu is only visible when an MCL script or script monitor window is open.

`New Window` – Open a new script window.

`Cascade` – Arrange the script and output windows so that they lie in a cascade down the screen.

`Tile` – Arrange the script and output windows so that they are tiled in the MCL Manager window.

`Arrange Icons` – Organise all minimised windows at the bottom of the main window.

2.5 Help

The following commands are available from the *Help* menu:

`Mcl Manager Help` – Displays this document.

`Mcl Program Help` – Displays the MCL script language documentation.

`About MCL Manager` – Displays MCL Manager version and copyright information.

3 MCL Manager Window

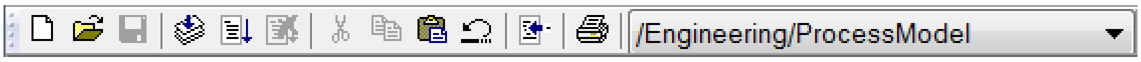
The following sections detail the functionality of the MCL Manager tool bar, windows and status bar.

- **Tool bar**
- **Main window**
- **Status bar**

3.1 Tool bar

The **tool bar** can be found at the top of the MCL Manager window.

Figure 2 MCL Manager – Status bar



The `New script` button creates a new blank MCL script.

The `Open script` button opens a standard Windows open file browser and allows the user to locate an MCL script to be opened.

The `Save script` button saves the currently selected script. If it is a new script the user is prompted to provide a filename and location using a standard Windows save file browser.

The `Parse script` button checks the syntax of the currently selected script. If an error is identified in the script the cursor is moved to the line containing the error and the line is high-lighted.

The `Run script` button executes the currently selected script. If necessary the script is parsed before it is executed.

The `Stop script` button stops the currently executing script.

The `Cut` button removes the currently selected text and places it in the Windows clipboard.

The `Copy` button copies the currently selected text and places it in the Windows clipboard.

The `Paste` button pastes the Windows clipboard contents at the current cursor location.

The `Undo` button undoes the last editing action. Repeatedly pressing the undo button causes earlier editing actions to be undone in the opposite order that they were executed.

The `Insert Statement` button opens the Insert Statement dialog to assist the user in creating more complex MCL script statements.

The `Print` button opens a standard Windows Print dialog, allowing the user to print the currently selected script.

/Engineering/ProcessModel The `Application` drop-down control allows the user to select the default application that the script will be run against.

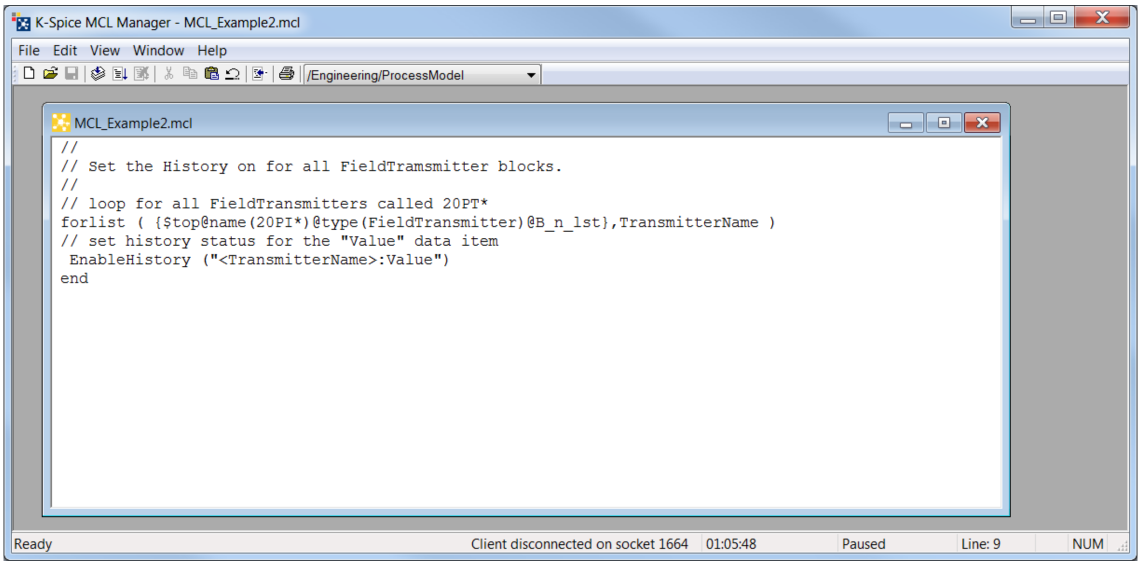
3.2 Main window

The **main window** area of MCL Manager is where scripts can be viewed and edited and the current execution state can be viewed.

Script window

The image below shows MCL Manager with an open script. The script can be edited in the same way as any normal text file in a standard text editor. Standard windows keyboard short-cuts for copy, cut and paste are available. In addition the user can double click anywhere on a statement to open the Insert Statement dialog.

Figure 3   MCL Manager – Script window

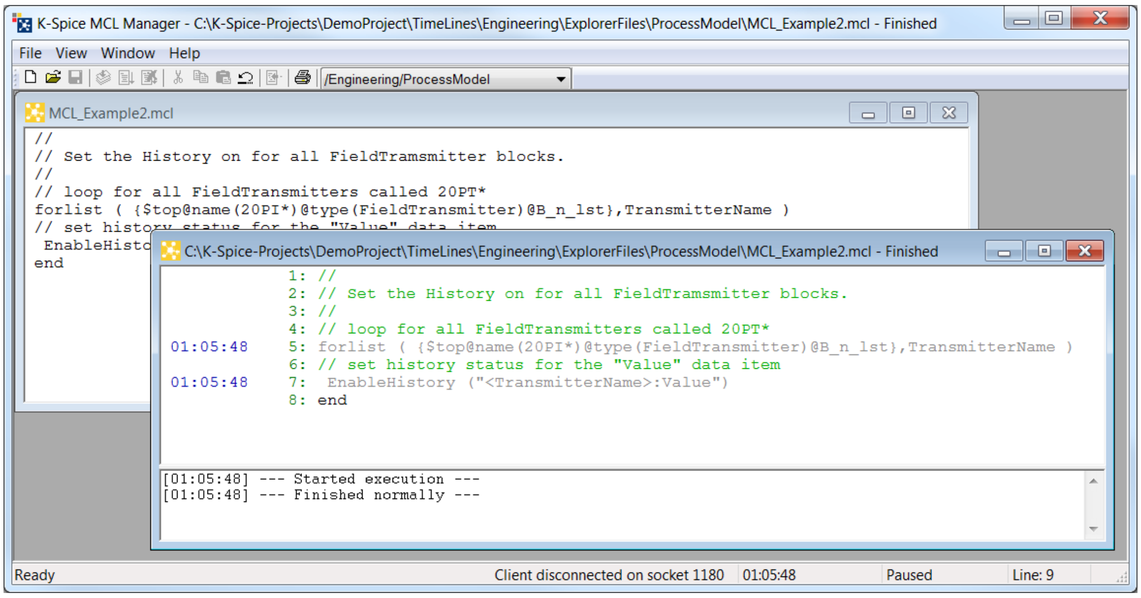

K-Spice MCL Manager - MCL_Example2.mcl
File Edit View Window Help
/Engineering/ProcessModel
MCL_Example2.mcl
Ready
Client disconnected on socket 1664
01:05:48
Paused
Line: 9
NUM

```
//
// Set the History on for all FieldTramsmitter blocks.
//
// loop for all FieldTransmitters called 20PT*
forlist ( {$top@name(20PI*)@type(FieldTransmitter)@B_n_lst},TransmitterName )
// set history status for the "Value" data item
 EnableHistory ("<TransmitterName>:Value")
end
```

Execution window

The image below shows MCL Manager with an executed script. The execution window is opened automatically when a script is run and allows the user to follow the execution of a script in real time. Lines are high lighted as they are executed and the model time is noted at the start of the line once the line has been executed. Lines are greyed out once they have been executed. Any messages from the script are written to the bottom of the window along with the **Started execution** and **Finished normally** messages.

Figure 4   MCL Manager – Execution window


K-Spice MCL Manager - C:\K-Spice-Projects\DemoProject\TimeLines\Engineering\ExplorerFiles\ProcessModel\MCL_Example2.mcl - Finished
File View Window Help
/Engineering/ProcessModel
MCL_Example2.mcl
C:\K-Spice-Projects\DemoProject\TimeLines\Engineering\ExplorerFiles\ProcessModel\MCL_Example2.mcl - Finished
Ready
Client disconnected on socket 1180
01:05:48
Paused
Line: 9

```
//
// Set the History on for all FieldTramsmitter blocks.
//
// loop for all FieldTransmitters called 20PT*
forlist ( {$top@name(20PI*)@type(FieldTransmitter)@B_n_lst},TransmitterName )
// set history status for the "Value" data item
 EnableHisto
end
```

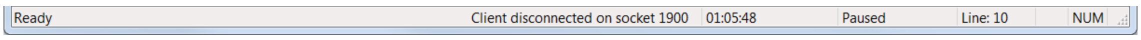
```
            1: //
            2: // Set the History on for all FieldTramsmitter blocks.
            3: //
            4: // loop for all FieldTransmitters called 20PT*
01:05:48    5: forlist ( {$top@name(20PI*)@type(FieldTransmitter)@B_n_lst},TransmitterName )
            6: // set history status for the "Value" data item
01:05:48    7:  EnableHistory ("<TransmitterName>:Value")
            8: end
```

```
[01:05:48] --- Started execution ---
[01:05:48] --- Finished normally ---
```

3.3 Status bar

The **status bar** can be found at the bottom of the MCL Manager window.

Figure 5 MCL Manager – Status bar


Ready | Client disconnected on socket 1900 | 01:05:48 | Paused | Line: 10 | NUM

The status bar has the following sections:

- User feedback message

  Normally the user feedback message is **Ready**. The status message changes depending upon the active item. For example, moving the mouse over the **undo** button on the tool bar will result in the user feedback message changing to **undo the last action**.

- Connection status

  Once a script has been activated or connected to, the connection status shows the most recent connection to an active MCL script. If there are no active connections the message will show the last active connection as 'disconnected'. The socket number refers to the TCP/IP socket used for communication.

- Model time

  This is the model time for the application displayed in the tool bar application drop-down.

- Model status

  The model status is the status for the current application in the tool bar application drop-down. The status messages are **Inactive Timeline**, **Paused** and **Running**.

- Line number

  This is the line number in the selected script at the current cursor location.

- CAPS

  'CAPS' is displayed when the **Caps Lock** is active, otherwise there is nothing displayed.

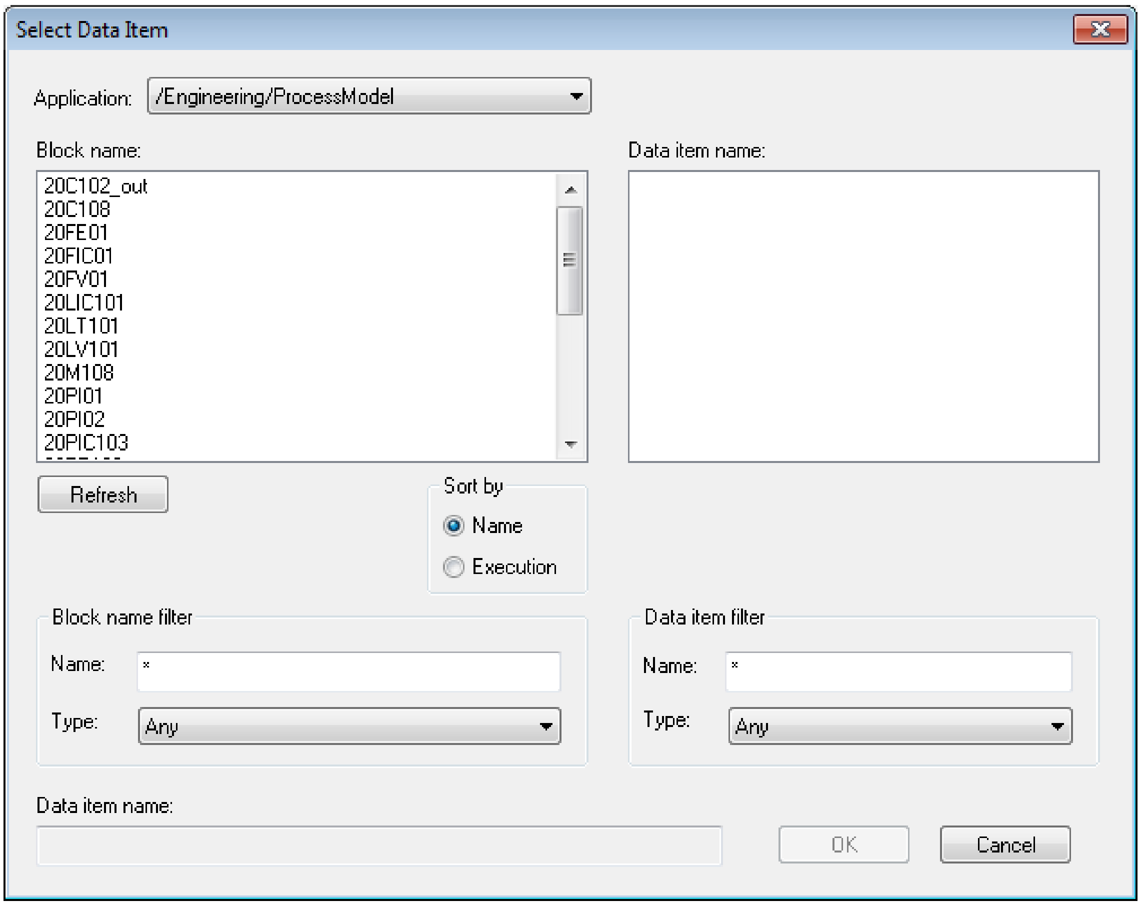
- NUM

  'NUM' is displayed when the **Num Lock** is active, otherwise there is nothing displayed.

4 Inserting data items

The **Edit→Insert Data Item...** menu option opens the **Select Data Item** dialog which allows the user to choose a model data item to be inserted into the script at the current cursor location. The required curly braces are also added to tell MCL the data item value should be requested from, or sent to, K-Spice®.

Figure 6 Select Data Item



Select Data Item

Application: /Engineering/ProcessModel

Block name:

20C102_out
20C108
20FE01
20FIC01
20FV01
20LIC101
20LT101
20LV101
20M108
20PI01
20PI02
20PIC103

Data item name:

Sort by: [✓ Name] Execution

Block name filter — Name: * Type: Any

Data item filter — Name: * Type: Any

Data item name:

The **Select Data Item** dialog has the following sections:

- Application:

  The **Application:** drop-down allows the user to choose the application containing the data item they require.

- Block name:

  The **Block name:** field allows the user to select the model block containing the required data item. This field is filtered to match the **Block name filter** settings. Once a block name has been selected the **Tag name** field will be updated with available data items.

- Refresh

  The **Refresh** button updates the **Block name** and **Tag name** fields using the current **Block name filter** and **Tag name filter** settings.

- Sort by

  The **Sort by** options allow the use to sort the **Block name** field either alphabetically or in the order the blocks are executed in the application.

- Block name filter

  The **Block name filter** — **Name:** field allows the user to filter the **Block name** field. Filtering can be done using "*" and "?". For example "*101*, "*PT??01" etc where "*" represents one or more characters and "?" represents a single character.

  The **Block name filter** — **Type:** drop-down allows the user to filter for specific block types (Valve, Separator etc). The default "Any" shows all block types.

- Tag name:

  The **Tag name:** field allows the user to select a data item from the selected block. This field is filtered to match the **Tag name filter** settings. Once a data item has been selected the user can insert this into the script by pressing **OK** or double-clicking on it.

- Tag name filter

  The **Tag name filter** — **Name:** field allows the user to filter the **Tag name** field. Filtering can be done using "*" and "?". For example "*Time, "Constant?" etc where "*" represents one or more characters and "?" represents a single character.

  The **Tag name filter** — **Type:** drop-down allows the user to filter for specific data item types (Boolean, integer etc). The default "Any" shows all data item types.

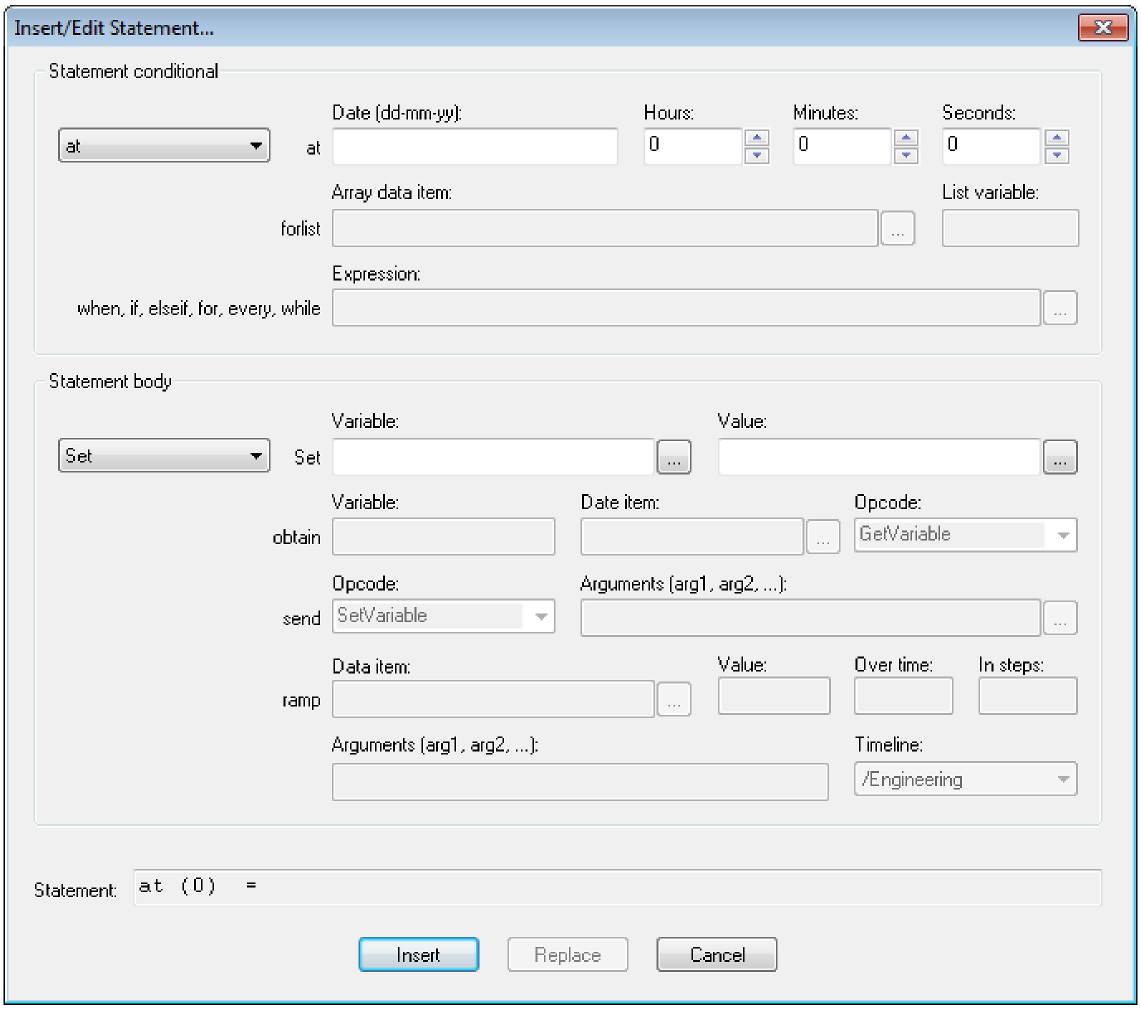
- Model variable name:

  The **Model variable name:** field shows the current data item selected from the **Block name:** and **Tag name:** fields.

5 Inserting statements

The **Edit→Insert Statement...** menu option opens the **Insert/Edit Statement...** dialog which allows the user to create or edit an MCL script statement. This dialog is also accessible from the `Insert Statement` button on the tool bar.

Figure 7 Insert/Edit Statement...



Insert/Edit Statement...

Statement conditional

at ▾ — at — Date (dd-mm-yy): — Hours: 0 — Minutes: 0 — Seconds: 0

forlist — Array data item: ... — List variable:

when, if, elseif, for, every, while — Expression: ...

Statement body

Set ▾ — Set — Variable: ... — Value: ...

obtain — Variable: — Date item: ... — Opcode: GetVariable

send — Opcode: SetVariable — Arguments (arg1, arg2, ...): ...

ramp — Data item: ... — Value: — Over time: — In steps:

Arguments (arg1, arg2, ...): — Timeline: /Engineering

Statement: `at (0) =`

Insert — Replace — Cancel

The **Insert/Edit Statement...** dialog has the following two main sections: **Statement conditional** and **Statement body**, In addition to this there is a **Statement:** field at the bottom of the dialog that shows how the statement will appear when it is inserted using the **Insert** button. If the **Insert/Edit Statement...** is opened by double clicking on an existing statement the **Replace** button can be used to update the original statement. The **Cancel** button can be used at any time to close the **Insert/Edit Statement...** dialog without making any changes to the script.

5.1 Conditional statements

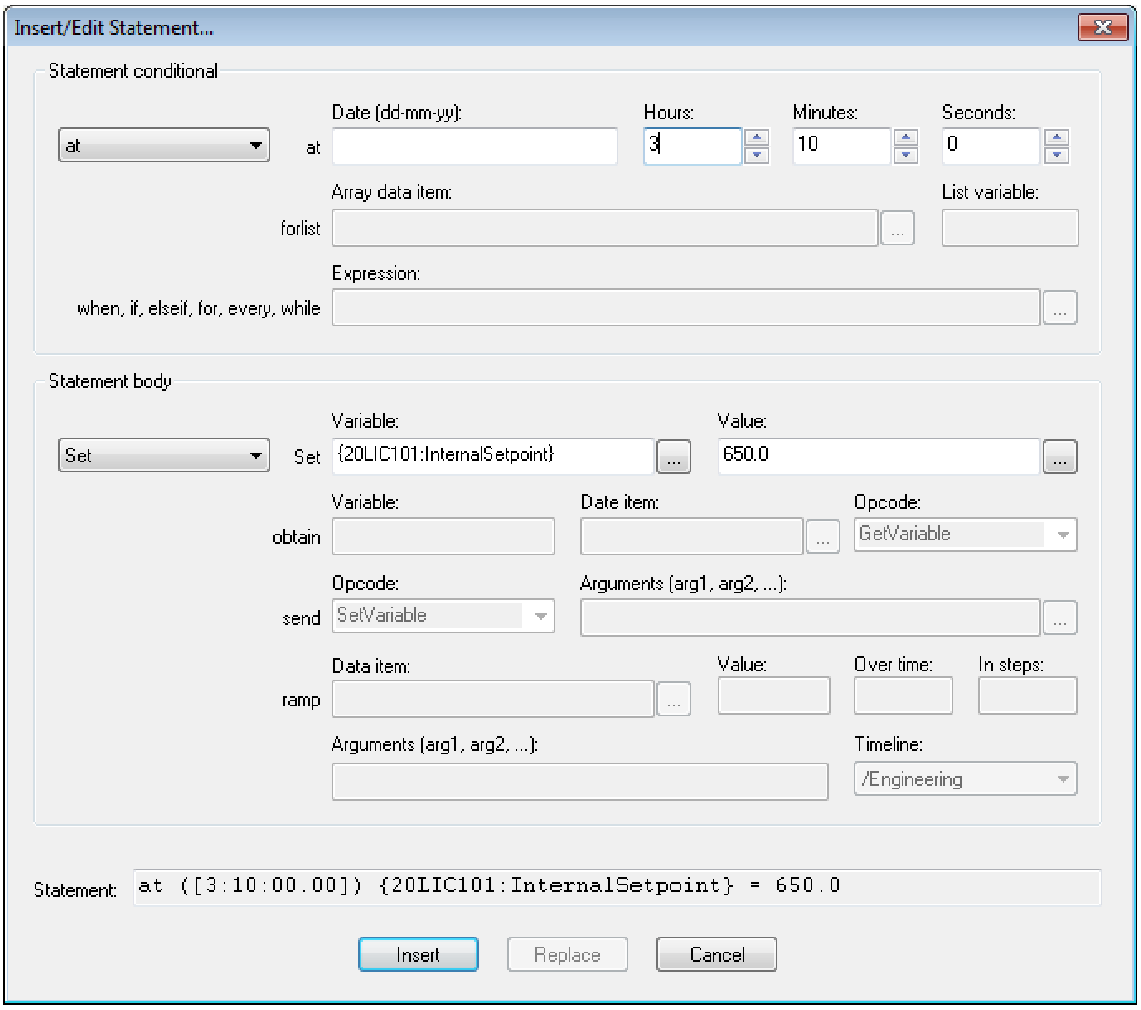
The **Statement conditional** section of the **Insert/Edit Statement...** dialog supports the part of a statement that must be true in order for the Statement body action to be executed.

More information on conditional statements can be found in the **MCL Program Help**.

5.1.1 At

The **At** condition causes the **Statement body** to be executed as soon as MCL detects the specified time is passed. A statement with a time that is before the current **Timeline** time when the script is started will be executed straight away. The **At** condition only requires the trigger time to be set.

Figure 8 Insert/Edit Statement — At



Statement conditional: at — Hours: 3, Minutes: 10, Seconds: 0

Statement body: Set — Variable: {20LIC101:InternalSetpoint} Value: 650.0

Statement: `at ([3:10:00.00]) {20LIC101:InternalSetpoint} = 650.0`

The above **At** statement appears in the script as follows:

```
at ([3:00:00]) {20LIC101:InternalSetpoint} = 650.0
```

In the above example the setpoint for controller *20LIC101* will be changed to 650 when the timeline time reaches 3 hours and 10 minutes. If the timeline time is already beyond 3 hours and 10 minutes the controller setpoint will be changed to 650 as soon as the script is started.

If multiple **Statement body** actions are required from a single trigger the user should choose **None** as the **Statement body** the insert the statement body actions as separate commands with the **Statement conditional** set to **None**. The **Statement body** commands are finished with the **end** command as follows:

```
at ([3:00:00])
 message ("changing 20LIC101 to 650mm")
 OldSetPoint = {20LIC101:InternalSetpoint}
 {20LIC101:InternalSetpoint} = 650.0
end
```

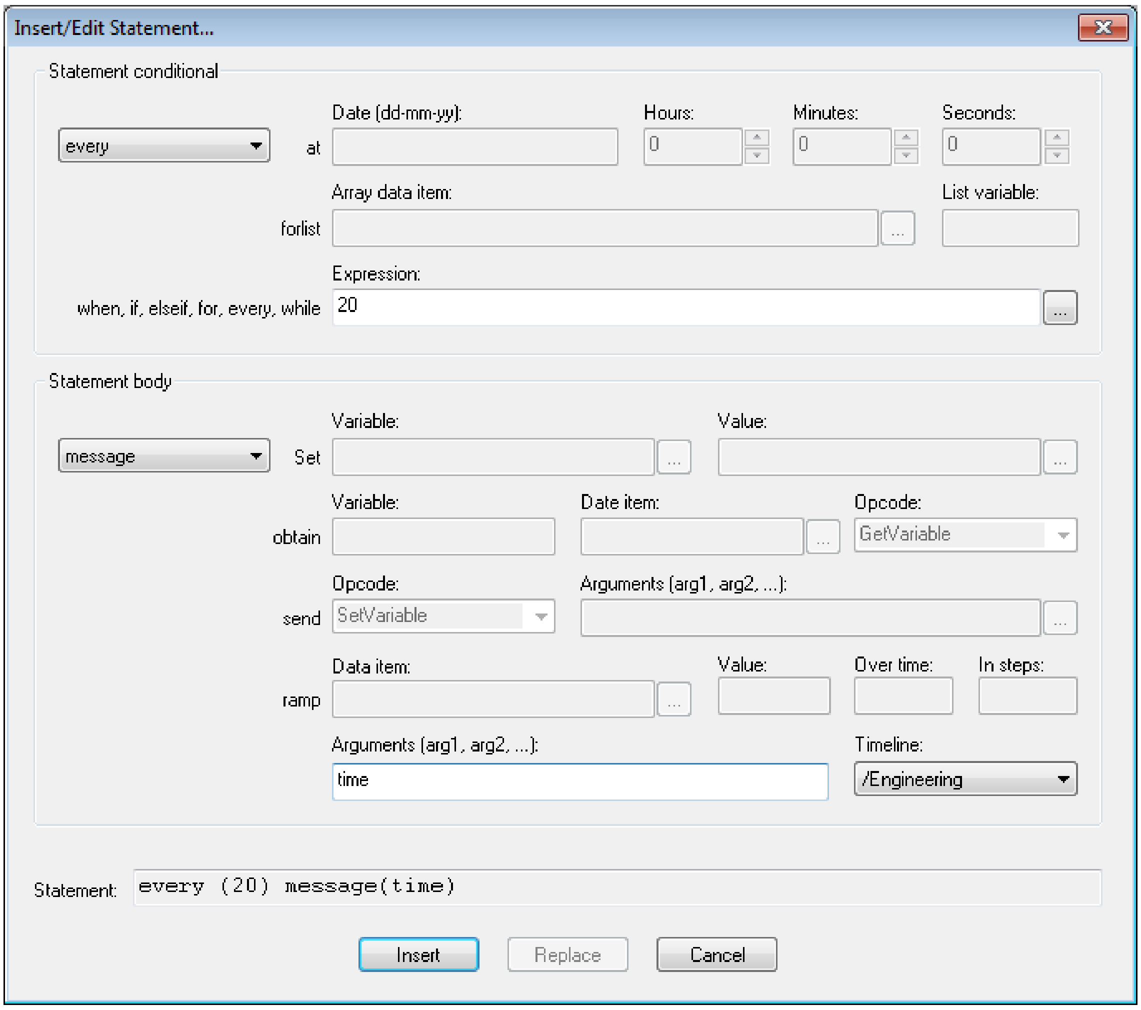
5.1.2 Every

The **Every** condition causes the **Statement body** to be executed as soon as MCL detects the specified time has passed. The **Statement body** will be repeatedly executed at the same interval until the script is terminated. The **Every** condition only requires the repeat time to be set.

Note

*As the **Every** condition repeats it will always be an active condition and the MCL script will not automatically terminate. To have the script terminate automatically the **Stop** or **Abort** command must be used. A running script can also be stopped using the stop button in the tool bar* .

Figure 9 Insert/Edit Statement — Every


Insert/Edit Statement...

Statement conditional

every | at | Date (dd-mm-yy): | Hours: 0 | Minutes: 0 | Seconds: 0

forlist | Array data item: | List variable:

when, if, elseif, for, every, while | Expression: 20

Statement body

message | Set | Variable: | Value:

obtain | Variable: | Date item: | Opcode: GetVariable

send | Opcode: SetVariable | Arguments (arg1, arg2, ...):

ramp | Data item: | Value: | Over time: | In steps:

Arguments (arg1, arg2, ...): time | Timeline: /Engineering

Statement: every (20) message(time)

The above **Every** statement appears in the script as follows:

```
every (20) message(time)
```

In the above example the standard **time** variable is printed every 20 seconds. The following output shows the above script running for 40 seconds before being terminated using the **stop** button:

```
[00:00:00] --- Started execution ---
[00:00:00] 0
[00:00:20] 20
[00:00:40] 40
[??:??:??] --- Program terminated abnormally ---
```

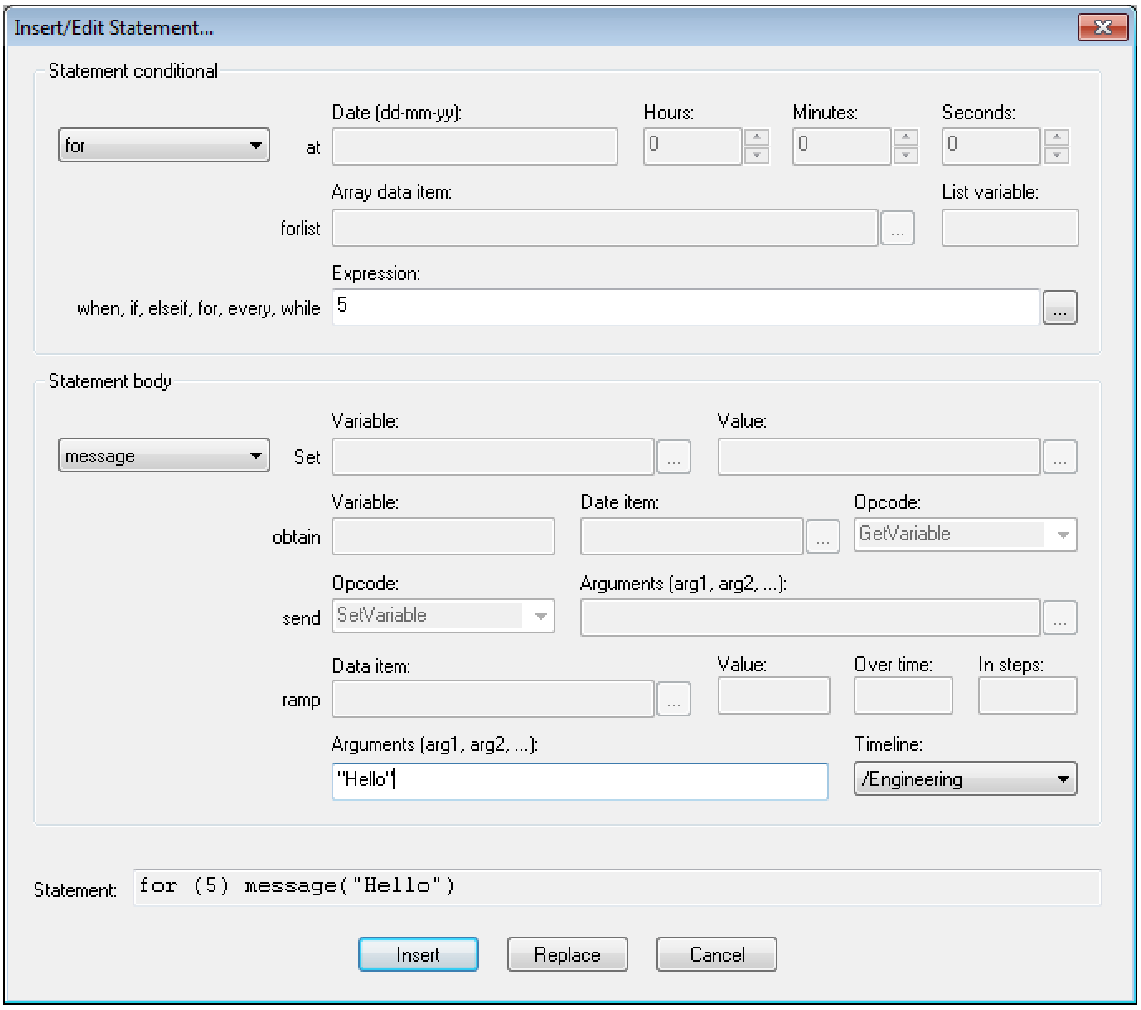
The following example shows the use of the stop command when a certain condition is met:

```
every (20)
 message(time)
 if (time >= 120)
  stop
 end
end
```

5.1.3 For

The **For** conditional causes the **Statement body** to be executed a specified number of times. The **For** conditional requires an integer or integer expression. The **...** button opens the Insert data item dialog to help the user define expressions using model data items. Local script variables and constants can also be used in the expression definition.

Figure 10 Insert/Edit Statement — For



Statement conditional: for
Hours: 0
Minutes: 0
Seconds: 0
Expression: 5
Statement body: message
Opcode (obtain): GetVariable
Opcode (send): SetVariable
Arguments (arg1, arg2, ...): "Hello"
Timeline: /Engineering
Statement: for (5) message("Hello")

The above **For** statement appears in the script as follows:

```
for (5) message("Hello")
```

In the above example the message “Hello” is printed 5 times.

The **For** statement is usually used with multiple statements as follows:

```
Number = 1
Cycles = 5
for (Cycles)
 message("Hello",Number)
 Number = Number + 1
end
```

The above example produces the following output in MCL Manager:

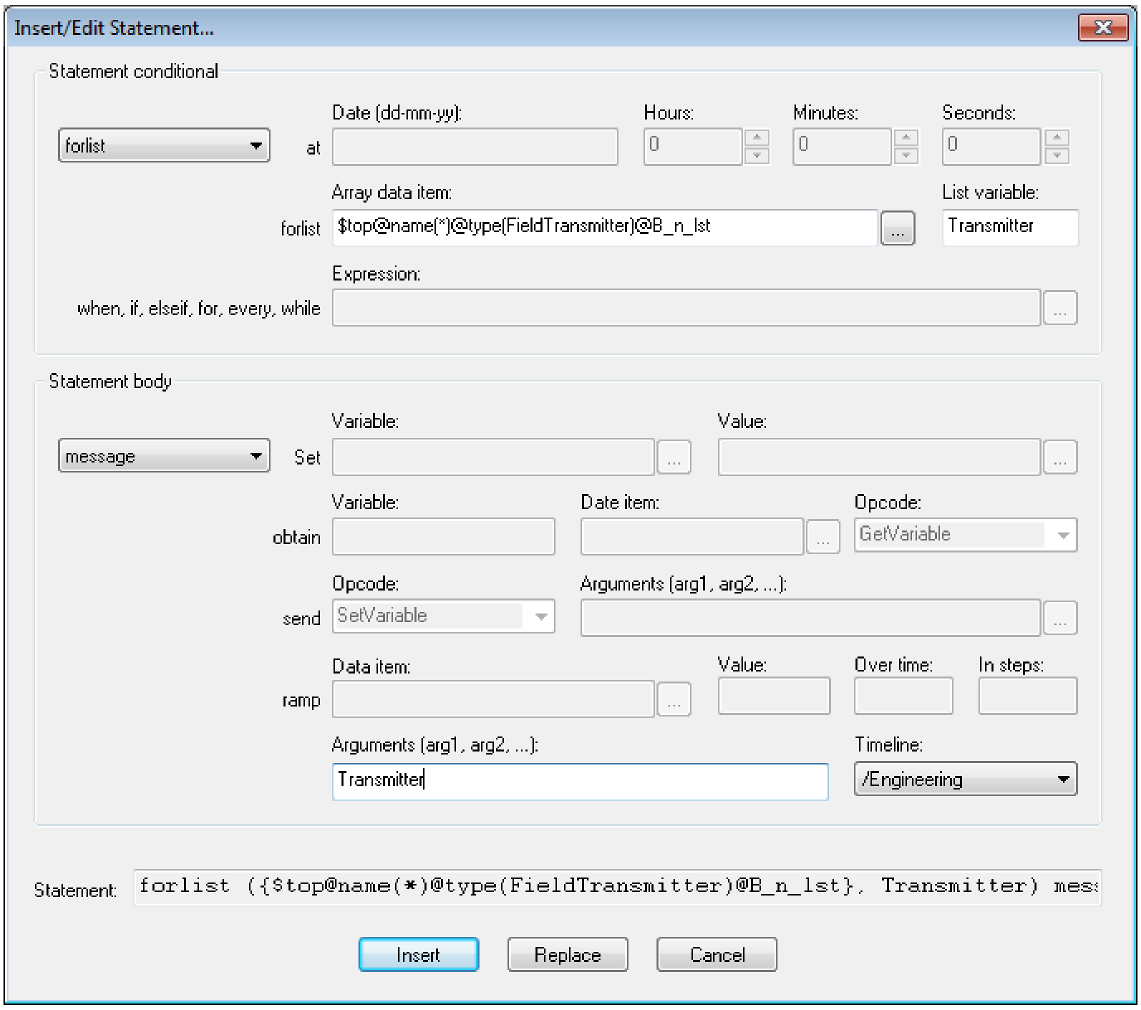
```
[00:00:00] --- Started execution ---
[00:00:00] Hello 1
[00:00:00] Hello 2
[00:00:00] Hello 3
[00:00:00] Hello 4
[00:00:00] Hello 5
```

```
[00:00:00] --- Finished normally ---
```

5.1.4 Forlist

The **Forlist** conditional is similar to **For** conditional in that it causes the **Statement body** to be executed a specified number of times. The difference is that the **Forlist** conditional requires an array input from the model and loops for each item in the array. The **...** button opens the Insert data item dialog to help the user define expressions using model data items. Local script variables and constants can also be used in the as part of the array request but the array must be returned by the model.

Figure 11 Insert/Edit Statement — Forlist


Insert/Edit Statement...

Statement conditional

forlist at Date (dd-mm-yy): Hours: 0 Minutes: 0 Seconds: 0

forlist Array data item: $top@name(*)@type(FieldTransmitter)@B_n_lst List variable: Transmitter

when, if, elseif, for, every, while Expression:

Statement body

message Set Variable: Value:

obtain Variable: Date item: Opcode: GetVariable

send Opcode: SetVariable Arguments (arg1, arg2, ...):

ramp Data item: Value: Over time: In steps:

Arguments (arg1, arg2, ...): Transmitter Timeline: /Engineering

Statement: forlist ({$top@name(*)@type(FieldTransmitter)@B_n_lst}, Transmitter) mes

Insert Replace Cancel

The above **Forlist** statement appears in the script as follows:

```
forlist ({$top@name(*)@type(FieldTransmitter)@B_n_lst}, Transmitter) message(Transmit
```

In the above example the message the name of each FieldTransmitter module in the model is printed. The command is a query to the model and works as follows:

- $top — start the search from the top of the call list (the first block in the model)
- @name(*) — match the block name with * — this gives all blocks
- @type(FieldTransmitter) — match the block type — this filters for FieldTransmitters only
- @B_n_lst — instructs the request to return a block list
- Transmitter — a text string to use as a variable in the loop — this will be set to the array element value for each cycle through the loop
- message (Transmitter) — this is the statement body which would normally be placed on a separate line

The above script results in the following output from MCL Manager when run against the CompressorAndMotorDemo model.

```
[00:00:00] --- Started execution ---
[00:00:00] 20LT101
[00:00:00] 20PT103
[00:00:00] 20PI02
[00:00:00] 20PI01
[00:00:00] --- Finished normally ---
```

Other options that can be included in the request are

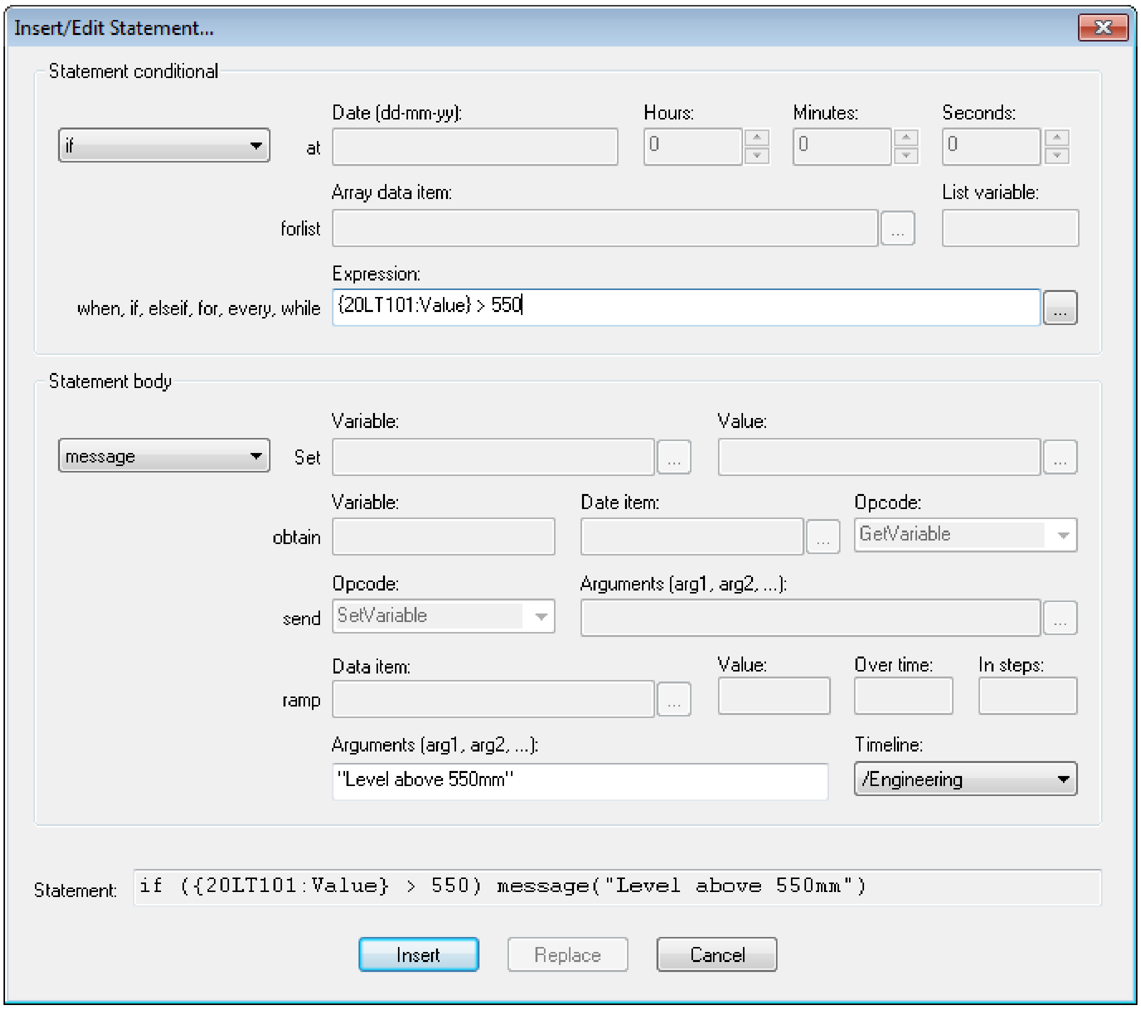
- @from(xxx) — start the search from block “xxx”
- @to(xxx) — end the search from block “xxx”
- @itemname(CvFullyOpen) — match blocks with a data item called “CvFullyOpen”
- @itemvalue(expression) — match blocks which match the expression — for example @itemvalue(CvFullyOpen>1000) for all blocks with a “CvFullyOpen” data item bigger than 1000.

See Appendix A in the MCL Program help for more examples of the **Forlist** command.

5.1.5 If, ElseIf, Else

The **If** trigger causes the **Statement body** to be executed if the conditional statement is true. The optional **ElseIf** trigger causes the **Statement body** to be executed if the conditional statement is true and the preceding **If** and any preceding **ElseIf** statement(s) are false. The optional **Else** trigger does not require an **Expression** to be defined and the **Statement body** will be executed if the preceding **If** and optional **ElseIf** condition(s) are all false. The **...** button opens the Insert data item dialog to help the user define expressions using model data items. Local script variables and constants can also be used in the expression definition.

Figure 12 Insert/Edit Statement — If



Statement conditional: if

Hours: 0, Minutes: 0, Seconds: 0

Expression (when, if, elseif, for, every, while): {20LT101:Value} > 550

Statement body: message

Opcode (obtain): GetVariable

Opcode (send): SetVariable

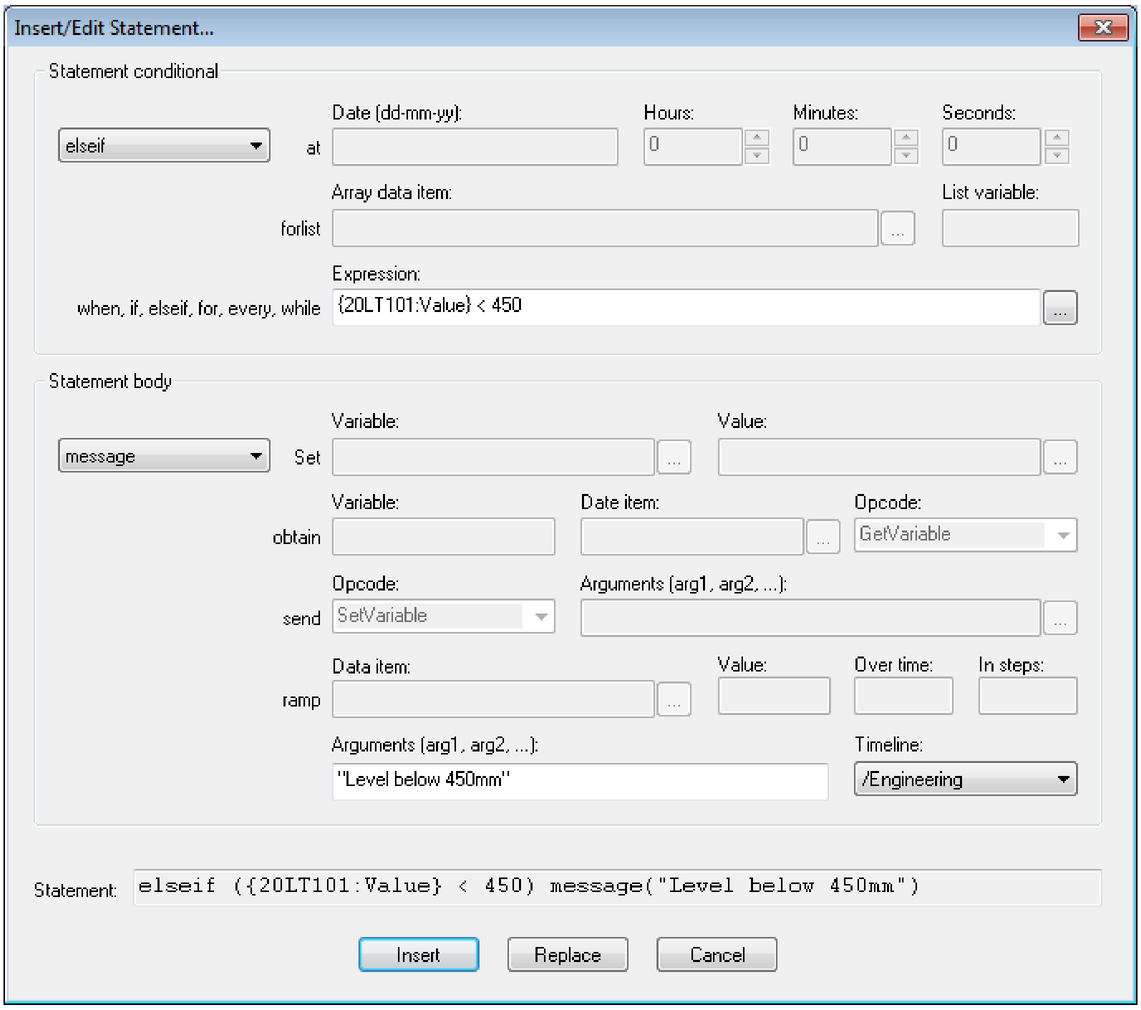
Arguments (arg1, arg2, ...): "Level above 550mm"

Timeline: /Engineering

Statement: `if ({20LT101:Value} > 550) message("Level above 550mm")`

In the above example the message “Level above 550mm” is printed if the transmitter 20LT101 has a process value over 550. The **If** statement will only be tested when it is first encountered. Use the **When** statement to continue testing for a condition until it becomes true.

Figure 13 Insert/Edit Statement — ElseIf



Insert/Edit Statement...

Statement conditional

elseif

at Date (dd-mm-yy): Hours: 0 Minutes: 0 Seconds: 0

forlist Array data item: List variable:

when, if, elseif, for, every, while Expression: {20LT101:Value} < 450

Statement body

message

Set Variable: Value:

obtain Variable: Date item: Opcode: GetVariable

send Opcode: SetVariable Arguments (arg1, arg2, ...):

ramp Data item: Value: Over time: In steps:

Arguments (arg1, arg2, ...): "Level below 450mm" Timeline: /Engineering

Statement: `elseif ({20LT101:Value} < 450) message("Level below 450mm")`

In the above example the message "Level below 450mm" is printed if the transmitter 20LT101 has a process value below 450. The **ElseIf** statement will only be tested when it is first encountered. **ElseIf** must follow an **If** statement or another **ElseIf** statement. If any of the preceding statements are true the statement will not be tested.

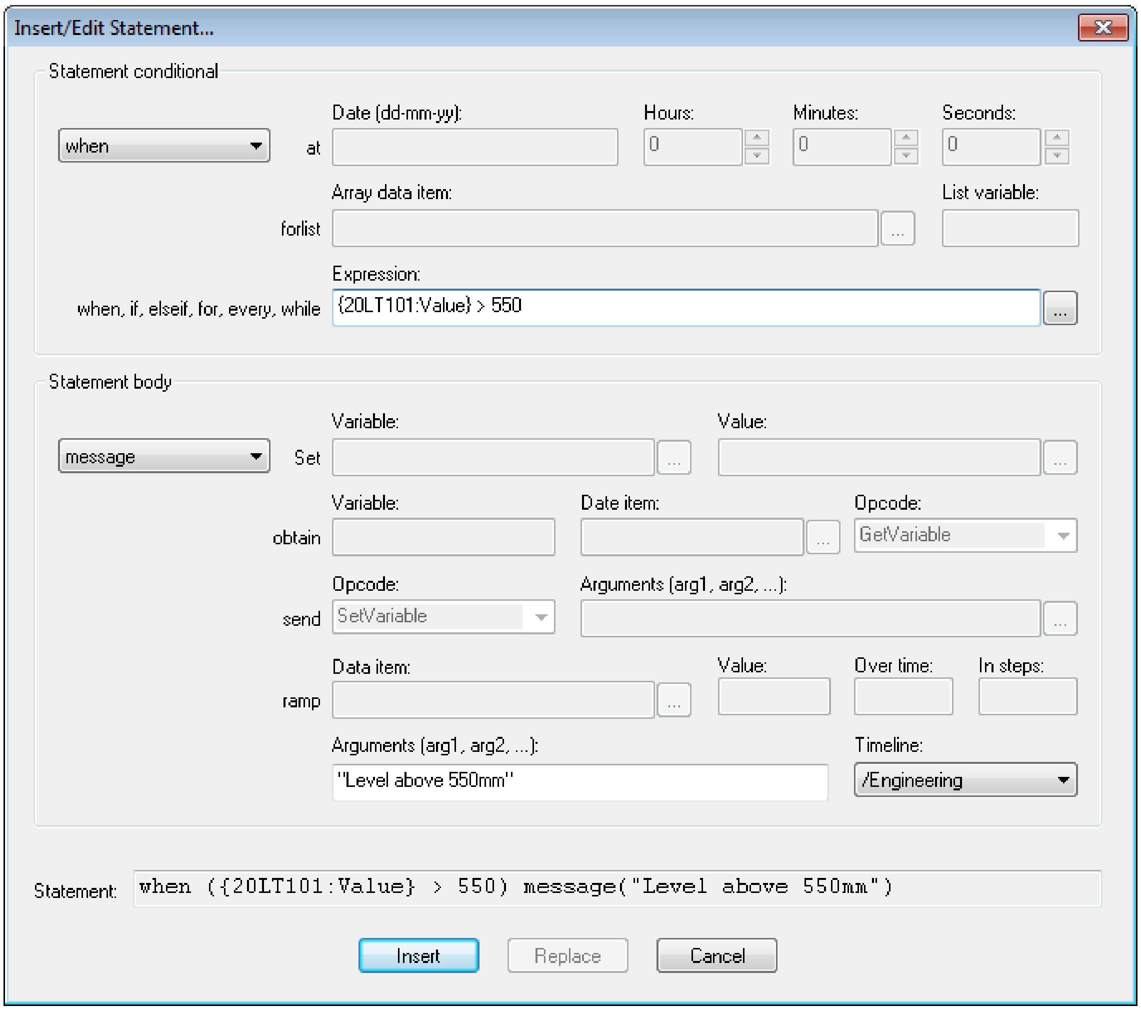
Using **ElseIf** and/or **Else** requires the addition of an **end** statement as follows:

```
if ({20LT101:Value} > 550) message("Level above 550mm")
elseif ({20LT101:Value} < 450) message("Level below 450mm")
elseif ({20LT101:Value} < 500) message("Level below 500mm, but above 450mm")
else message ("Level between 500 and 550mm")
end
```

5.1.6 When

The **When** condition causes the **Statement body** to be executed when the conditional statement becomes true. The **When** condition requires a trigger **Expression** to be defined. The **...** button opens the Insert data item dialog to help the user define expressions using model data items. Local script variables and constants can also be used in the expression definition.

Figure 14 Insert/Edit Statement — When


Insert/Edit Statement...

Statement conditional: when

Expression: {20LT101:Value} > 550

Statement body: message

Arguments (arg1, arg2, ...): "Level above 550mm"

Timeline: /Engineering

Statement: when ({20LT101:Value} > 550) message("Level above 550mm")

In the above example the message “Level above 550mm” is printed when the transmitter 20LT101 process value reaches a value over 550. The **When** statement will continue to be tested until the condition becomes true or the script is terminated.

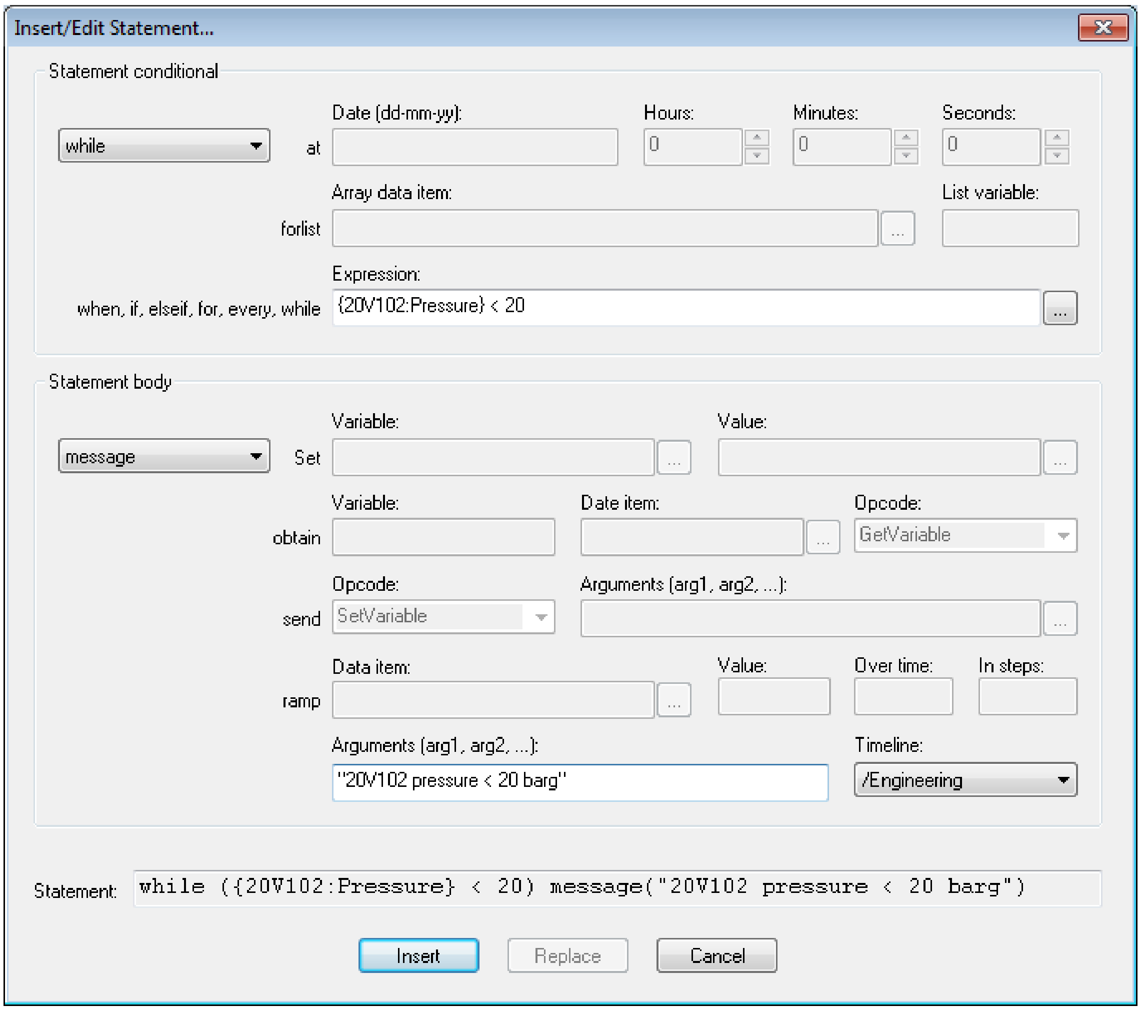
The above **When** condition appears in the script as follows:

```
when ({20LT101:Value} > 550) message ("Level above 550mm")
```

5.1.7 While

The **While** condition causes the **Statement body** to be repeatedly executed while the conditional statement remains true. The **While** condition requires a trigger **Expression** to be defined. The **...** button opens the Insert data item dialog to help the user define expressions using model data items. Local script variables and constants can also be used in the expression definition.

Figure 15 Insert/Edit Statement — While



Insert/Edit Statement...

Statement conditional

while	
at — Date (dd-mm-yy):	
Hours:	0
Minutes:	0
Seconds:	0
forlist — Array data item:	
List variable:	
when, if, elseif, for, every, while — Expression:	{20V102:Pressure} < 20

Statement body

message	
Set — Variable:	
Value:	
obtain — Variable:	
Date item:	
Opcode:	GetVariable
send — Opcode:	SetVariable
Arguments (arg1, arg2, ...):	
ramp — Data item:	
Value:	
Over time:	
In steps:	
Arguments (arg1, arg2, ...):	"20V102 pressure < 20 barg"
Timeline:	/Engineering

Statement: `while ({20V102:Pressure} < 20) message("20V102 pressure < 20 barg")`

In the above example the message "20V102 pressure < 20 barg" is printed repeatedly while the vessel pressure is below 20. The **While** statement will continue to be tested until the condition becomes false or the script is terminated. If the condition is false when the script first encounters the while statement the **Statement body** actions will never be executed.

The above **While** condition appears in the script as follows:

```
while ({20V102:Pressure} < 20) message("20V102 pressure < 20 barg")
```

5.2 Action Statements

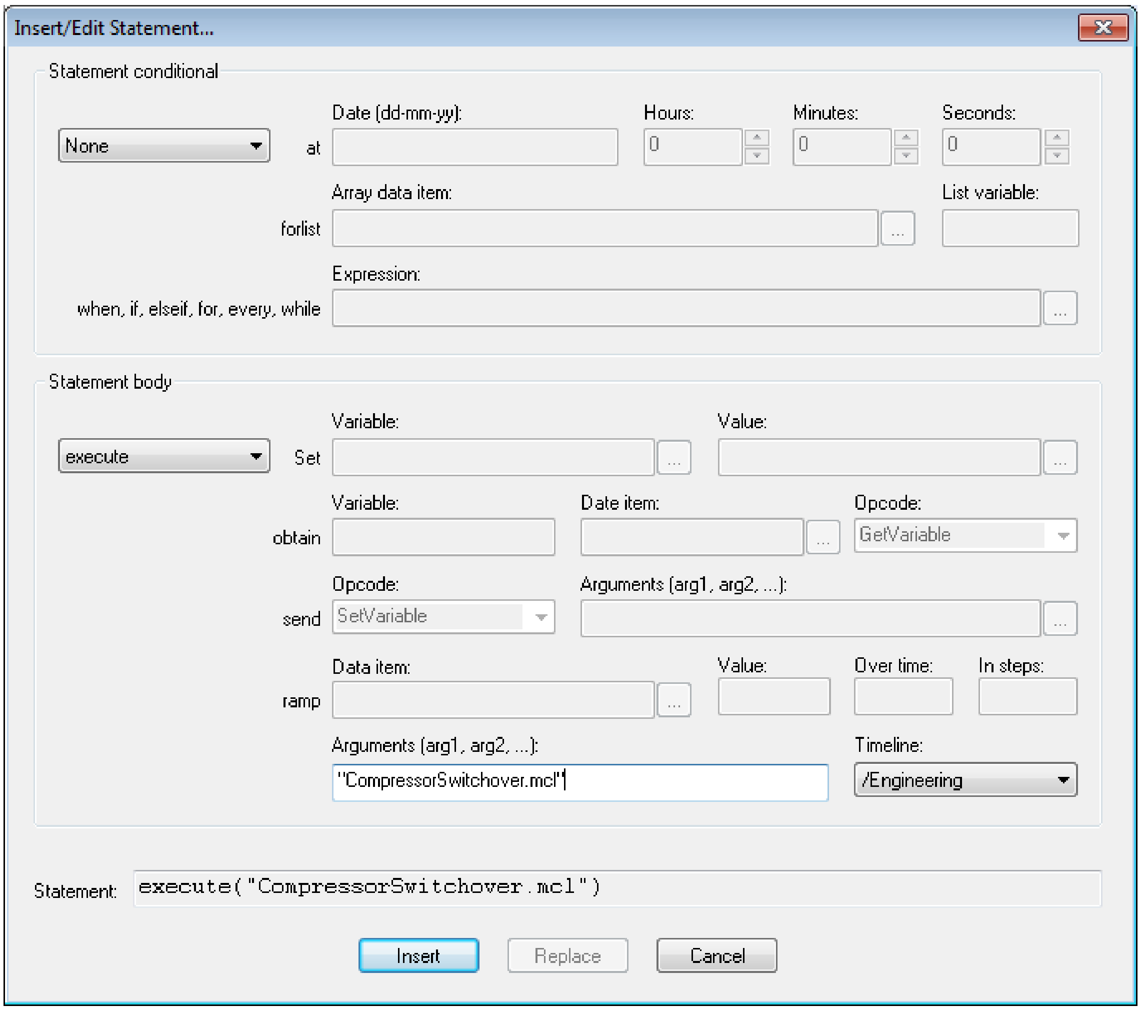
The **Statement body** section of the **Insert/Edit Statement...** dialog supports the part of a statement that is executed when the Statement Conditional test becomes true. If no conditional statement is defined the action will always execute as soon as the script encounters the action.

More information on action statements can be found in the **MCL Program Help**.

5.2.1 Execute

The **Execute** operation is used to run a separate MCL script.

Figure 16 Insert/Edit Statement — Execute



Insert/Edit Statement...

Statement conditional

None at Date (dd-mm-yy): Hours: 0 Minutes: 0 Seconds: 0

forlist Array data item: List variable:

when, if, elseif, for, every, while Expression:

Statement body

execute Set Variable: Value:

obtain Variable: Date item: Opcode: GetVariable

send Opcode: SetVariable Arguments (arg1, arg2, ...):

ramp Data item: Value: Over time: In steps:

Arguments (arg1, arg2, ...): "CompressorSwitchover.mcl" Timeline: /Engineering

Statement: `execute("CompressorSwitchover.mcl")`

The above **Execute** statement appears in the script as follows:

```
execute("CompressorSwitchover.mcl")
```

To run the new MCL script against a specific timeline and application an additional argument can be added as follows:

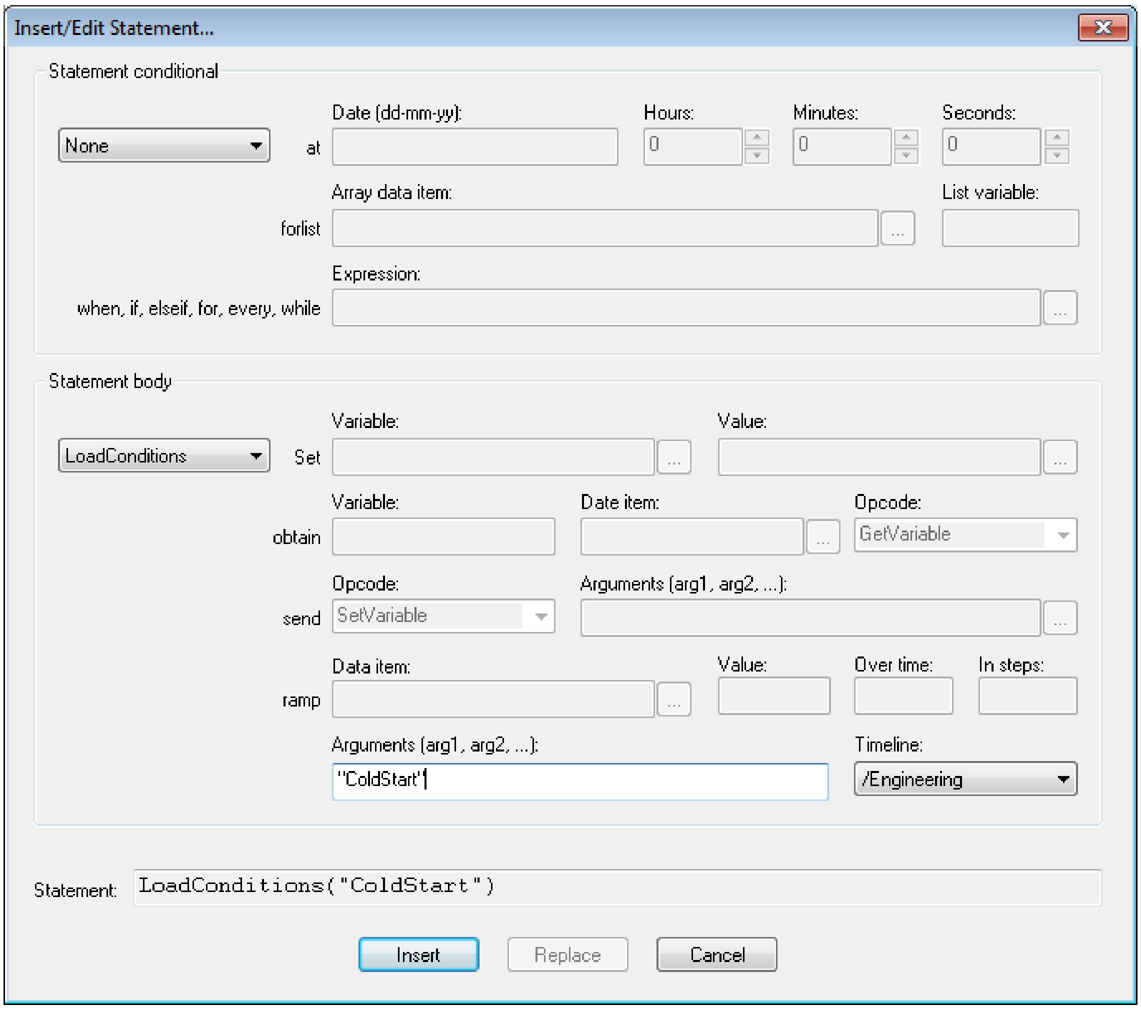
```
execute("CompressorSwitchover.mcl","/Planning/GasSystem")
```

5.2.2 Loading files

There are a number of commands to load K-Spice files — one command for each file type. Each command follows the same structure. The load file commands are as follows:

- LoadModel — load the model topology — blocks, connections etc.
- LoadParameters — load the physical information about the blocks — lengths, performance curves etc.
- LoadTrends — load the history configuration. It is normally loaded automatically with the model
- LoadConditions — load an initial condition.
- LoadHistory — load history data. This would normally be loaded in conjunction with a compatible condition.
- LoadSnapshot — load a previously saved snapshot.

Figure 17 Insert/Edit Statement — LoadConditions



Statement conditional: None
Hours: 0
Minutes: 0
Seconds: 0
Statement body: LoadConditions
Arguments (arg1, arg2, ...): "ColdStart"
Timeline: /Engineering
Statement: `LoadConditions("ColdStart")`

The above **LoadConditions** statement appears in the script as follows:

```
LoadConditions("ColdStart")
```

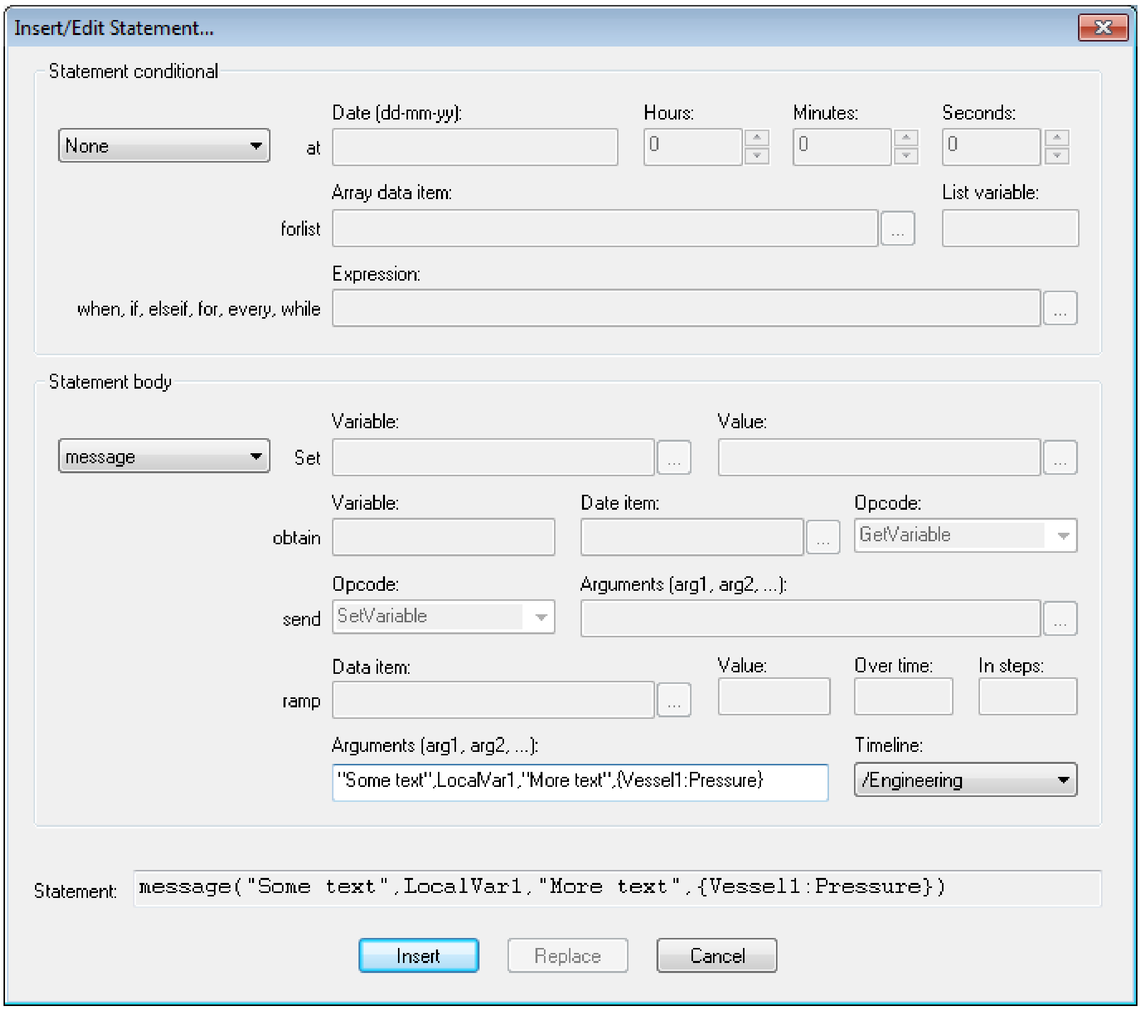
It is possible to load a file into a specific timeline as follows:

```
LoadConditions("ColdStart","/Engineering")
```

5.2.3 Message

The **Message** operation writes information to the Execution Window when a script is run from MCL Manager. The message can be a mixture of plain text, local MCL variables and model data item values, all separated using commas. Plain text should be written in “quoted strings”, MCL variables can be written as they are and model data items should be contained in curly braces {}. The message output is a single string without carriage returns.

Figure 18 Insert/Edit Statement — Message



Statement conditional: None; Hours: 0; Minutes: 0; Seconds: 0

Statement body: message

obtain Opcode: GetVariable

send Opcode: SetVariable

Arguments (arg1, arg2, ...): "Some text",LocalVar1,"More text",{Vessel1:Pressure}

Timeline: /Engineering

Statement: message("Some text",LocalVar1,"More text",{Vessel1:Pressure})

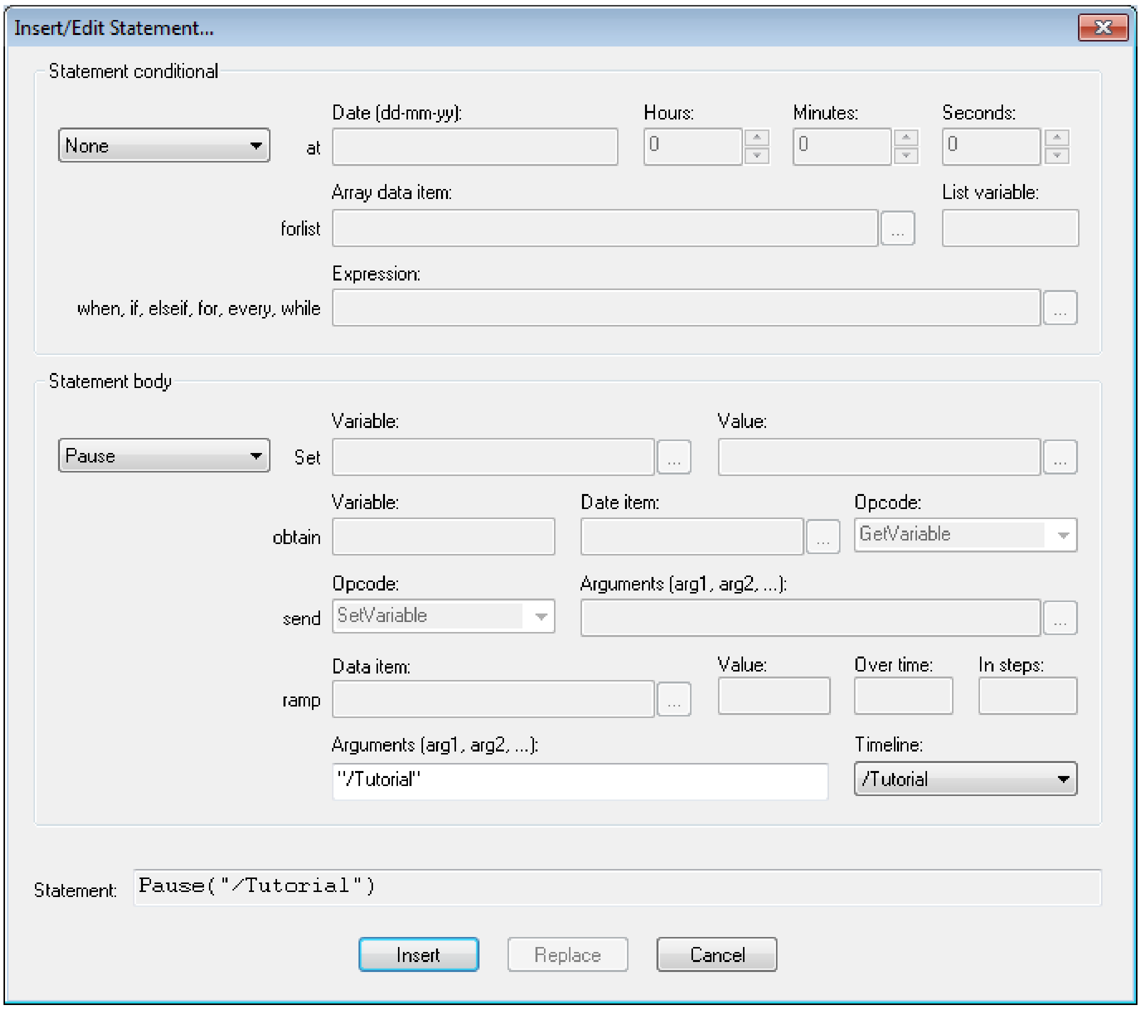
The above **Message** statement appears in the script as follows:

```
message("Some text",LocalVar1,"More text",{Vessel1:Pressure})
```

5.2.4 Pause

The **Pause** operation pauses the current or specified timeline if it is running.

Figure 19 Insert/Edit Statement — Pause



Insert/Edit Statement...

Statement conditional

None at Date (dd-mm-yy): Hours: 0 Minutes: 0 Seconds: 0

forlist Array data item: List variable:

when, if, elseif, for, every, while Expression:

Statement body

Pause

Set Variable: Value:

obtain Variable: Date item: Opcode: GetVariable

send Opcode: SetVariable Arguments (arg1, arg2, ...):

ramp Data item: Value: Over time: In steps:

Arguments (arg1, arg2, ...): "/Tutorial" Timeline: /Tutorial

Statement: Pause("/Tutorial")

The above **Pause** statement appears in the script as follows:

```
Pause("/Tutorial")
```

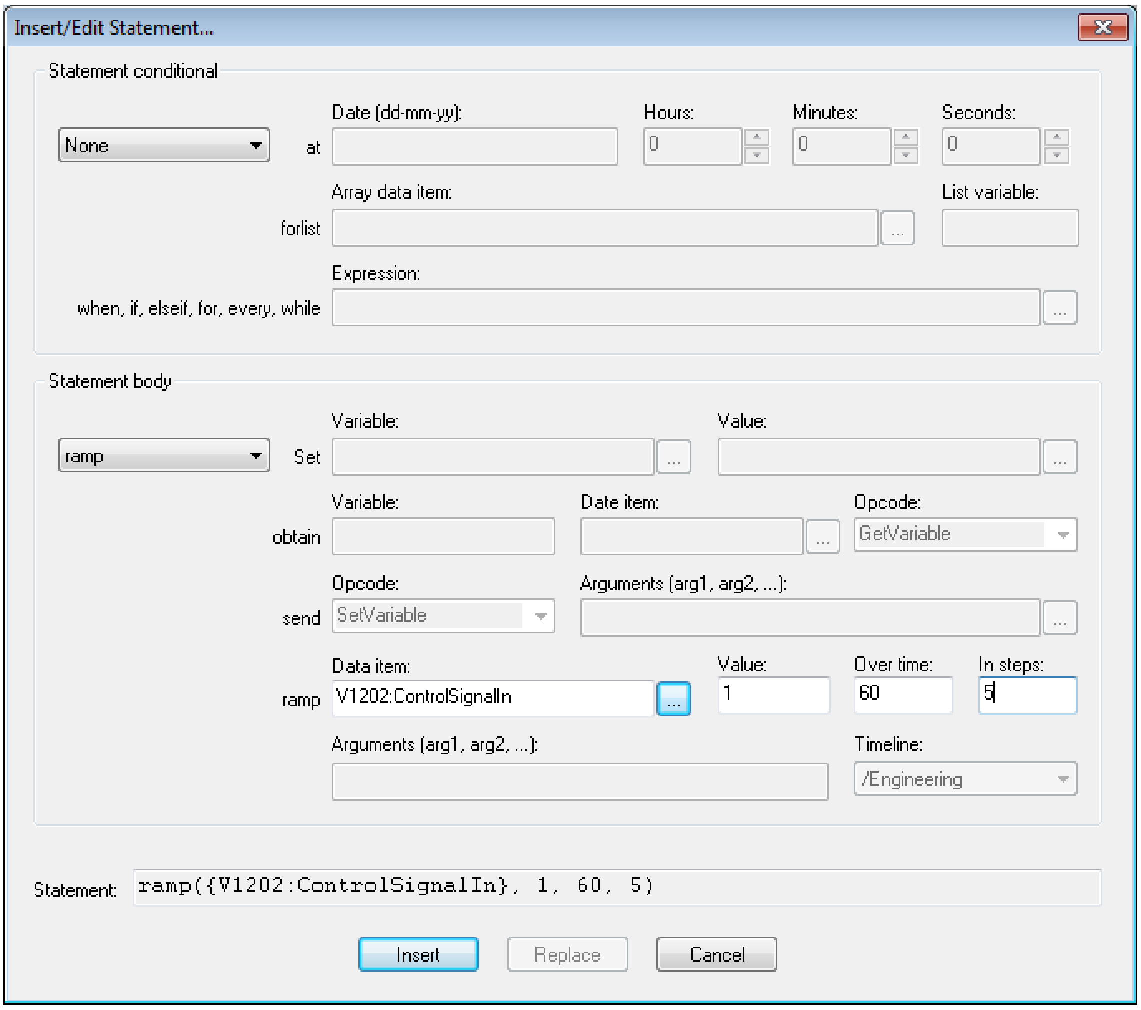
to pause the current timeline use the following:

```
Pause(0)
```

5.2.5 Ramp

The **Ramp** operation allows the user to ramp an application data item over time. The ramp operation is configured with an application data item to be ramped to a target **Value: Over time:** in seconds, **In steps:** of x seconds. The **...** button opens the Insert data item dialog to help the user find the required data item.

Figure 20 Insert/Edit Statement — Ramp



Insert/Edit Statement...

Statement conditional

None at Date (dd-mm-yy): Hours: 0 Minutes: 0 Seconds: 0

forlist Array data item: List variable:

when, if, elseif, for, every, while Expression:

Statement body

ramp

Set Variable: Value:

obtain Variable: Date item: Opcode: GetVariable

send Opcode: SetVariable Arguments (arg1, arg2, ...):

ramp Data item: V1202:ControlSignalIn Value: 1 Over time: 60 In steps: 5

Arguments (arg1, arg2, ...): Timeline: /Engineering

Statement: `ramp({V1202:ControlSignalIn}, 1, 60, 5)`

The above **Ramp** statement appears in the script as follows:

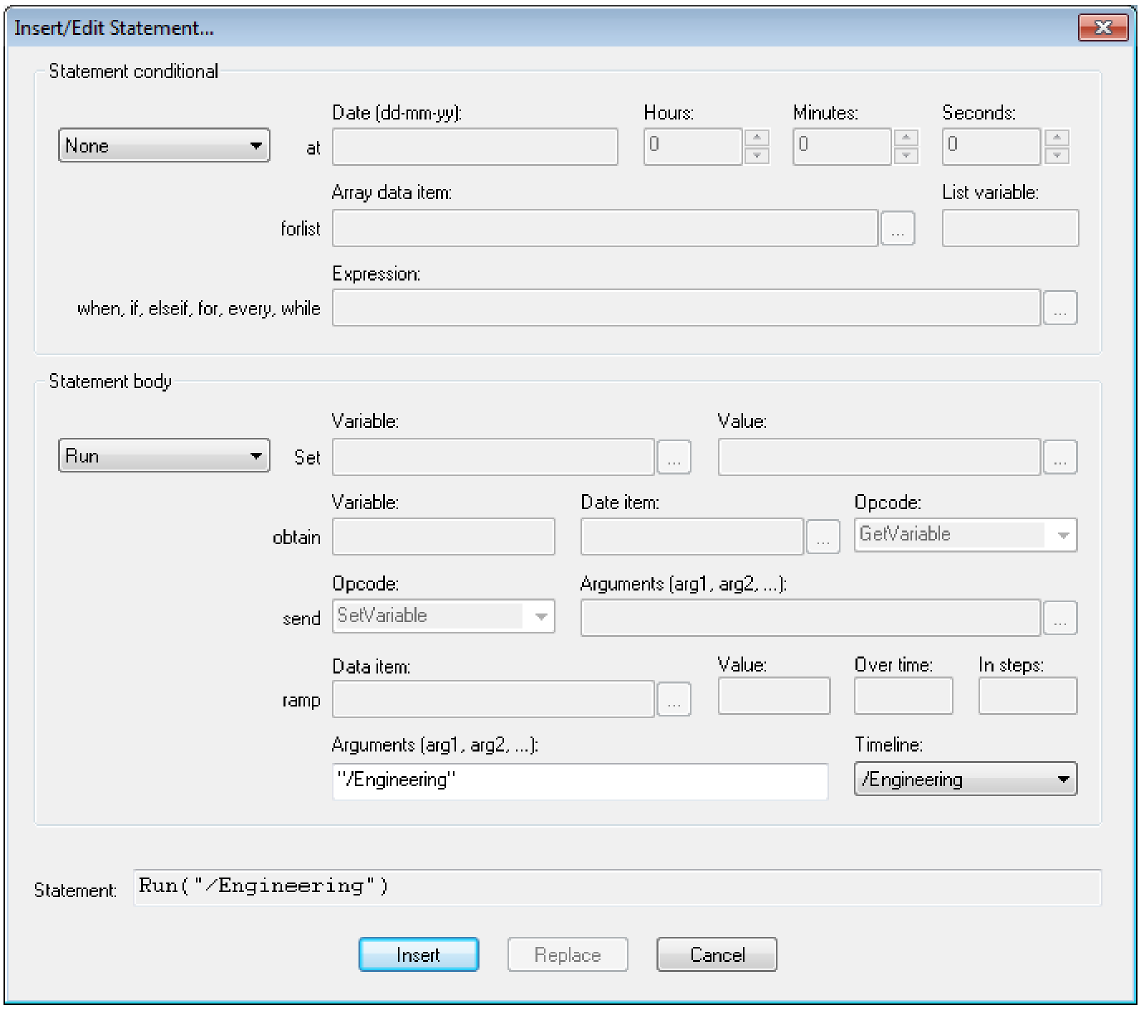
```
ramp({V1202:ControlSignalIn}, 1, 60, 5)
```

This **Ramp** statement will ramp up valve V1202’s control signal to a target value of 1 over 60 seconds with steps every 5 seconds.

5.2.6 Run

The **Run** operation runs the current or specified timeline if it is paused.

Figure 21 Insert/Edit Statement — Run



Hours: 0; Minutes: 0; Seconds: 0

Arguments (arg1, arg2, ...): "/Engineering"

Timeline: /Engineering

Statement: `Run("/Engineering")`

The above **Run** statement appears in the script as follows:

```
Run("/Engineering")
```

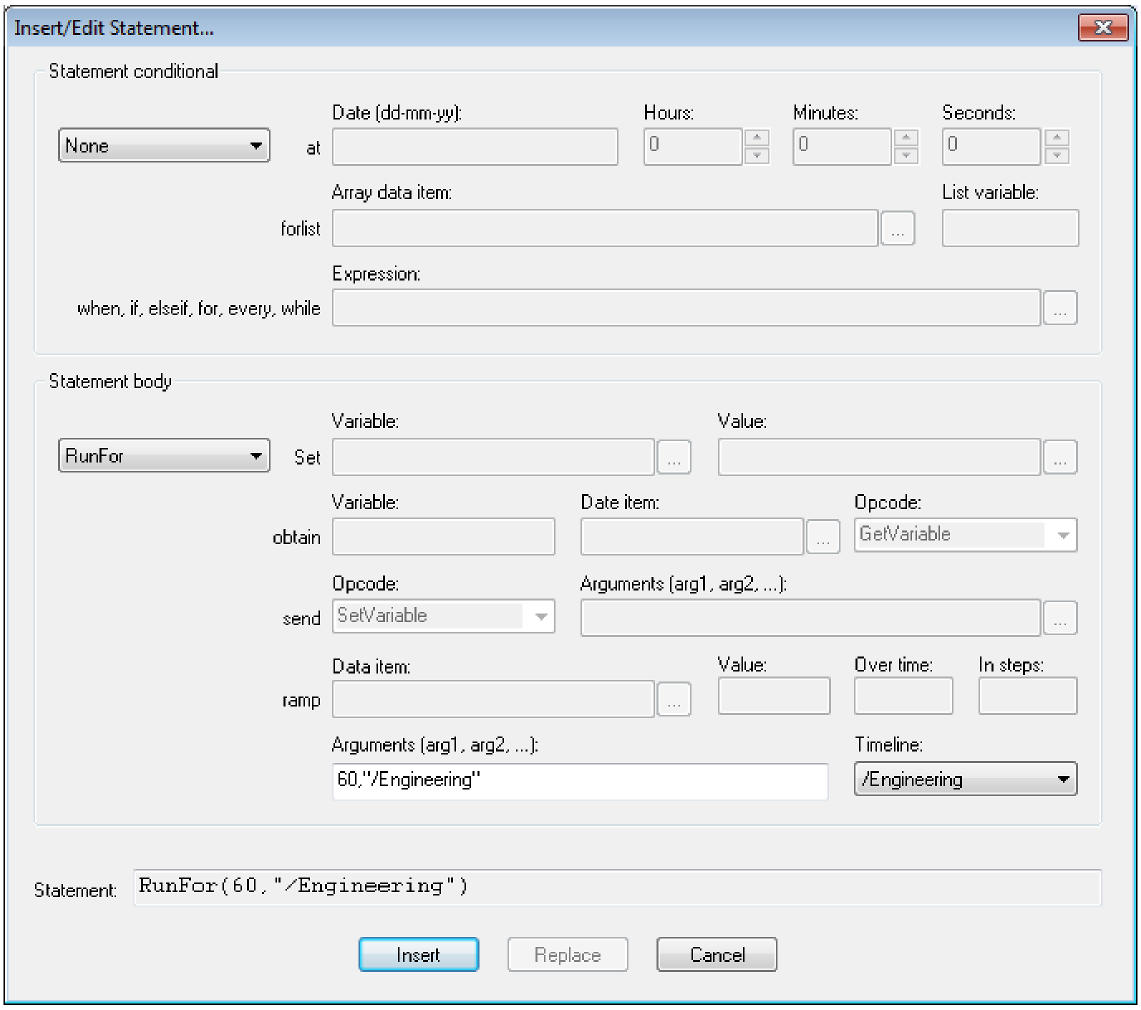
To **Run** the current timeline use the following:

```
Run(0)
```

5.2.7 RunFor

The **RunFor** operation runs the current or specified timeline for the specified time at which point the model will automatically pause. If the time is specified as a single number it is assumed to be in seconds.

Figure 22 Insert/Edit Statement — RunFor



Arguments (arg1, arg2, ...): 60,"/Engineering"

Timeline: /Engineering

Statement: `RunFor(60,"/Engineering")`

The above **RunFor** statement will run the “Engineering” timeline for 60 seconds and appears in the script as follows:

```
RunFor(60,"/Engineering")
```

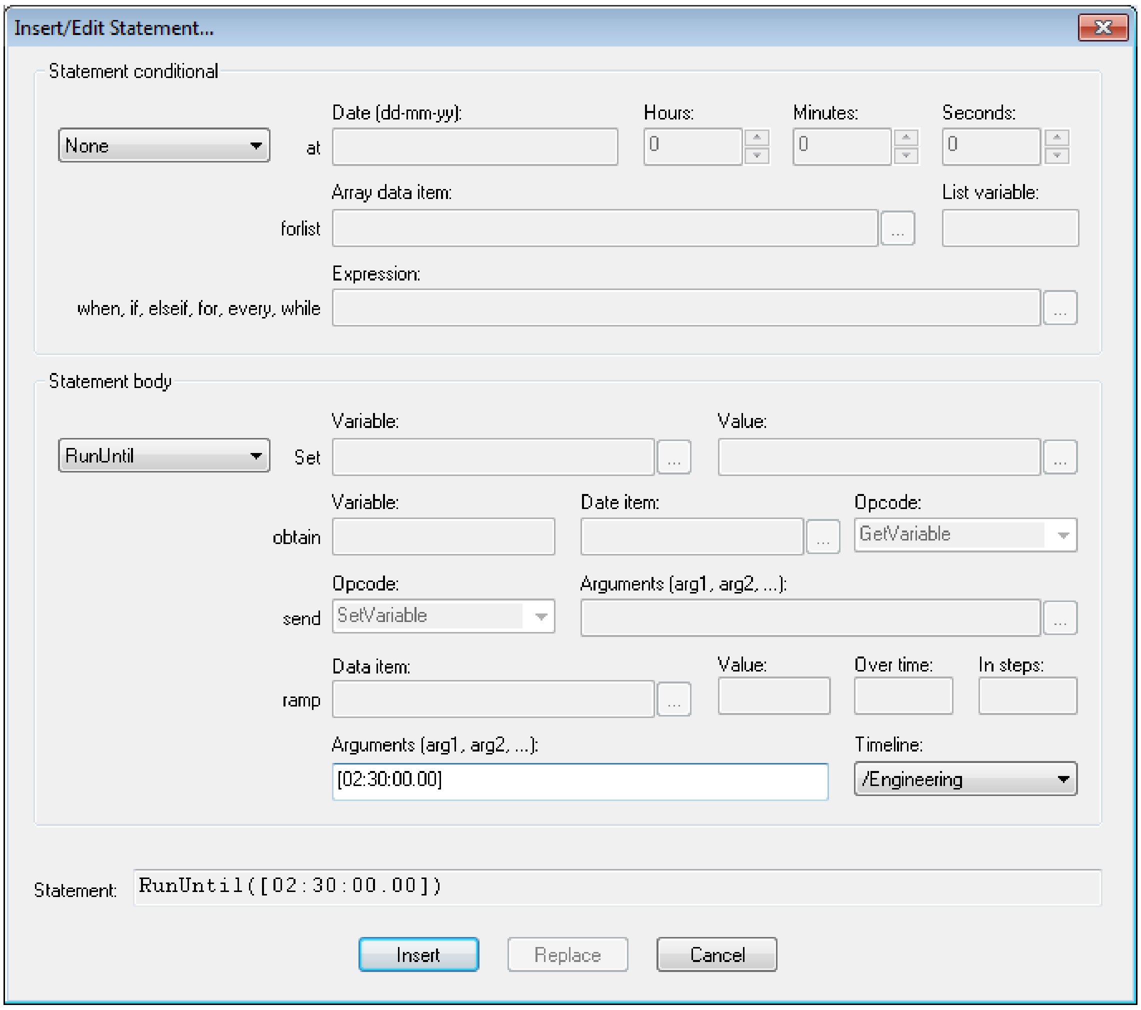
To **RunFor** a specified time on the current timeline use the following:

```
RunFor([02:30:00.00])
```

5.2.8 RunUntil

The **RunUntil** operation runs the current or specified timeline until the specified time and date at which point the model will automatically pause.

Figure 23 Insert/Edit Statement — RunUntil



Arguments (arg1, arg2, ...): [02:30:00.00]

Timeline: /Engineering

Statement: RunUntil([02:30:00.00])

The above **RunUntil** statement appears in the script as follows:

```
RunUntil([02:30:00.00])
```

To **RunUntil** a specific time on a specific timeline use the following:

```
RunUntil([02:30:00.00],"/Tutorial")
```

To **RunUntil** a specific time and date use the following:

```
RunUntil([30-06-12 15:30:00.00])
```

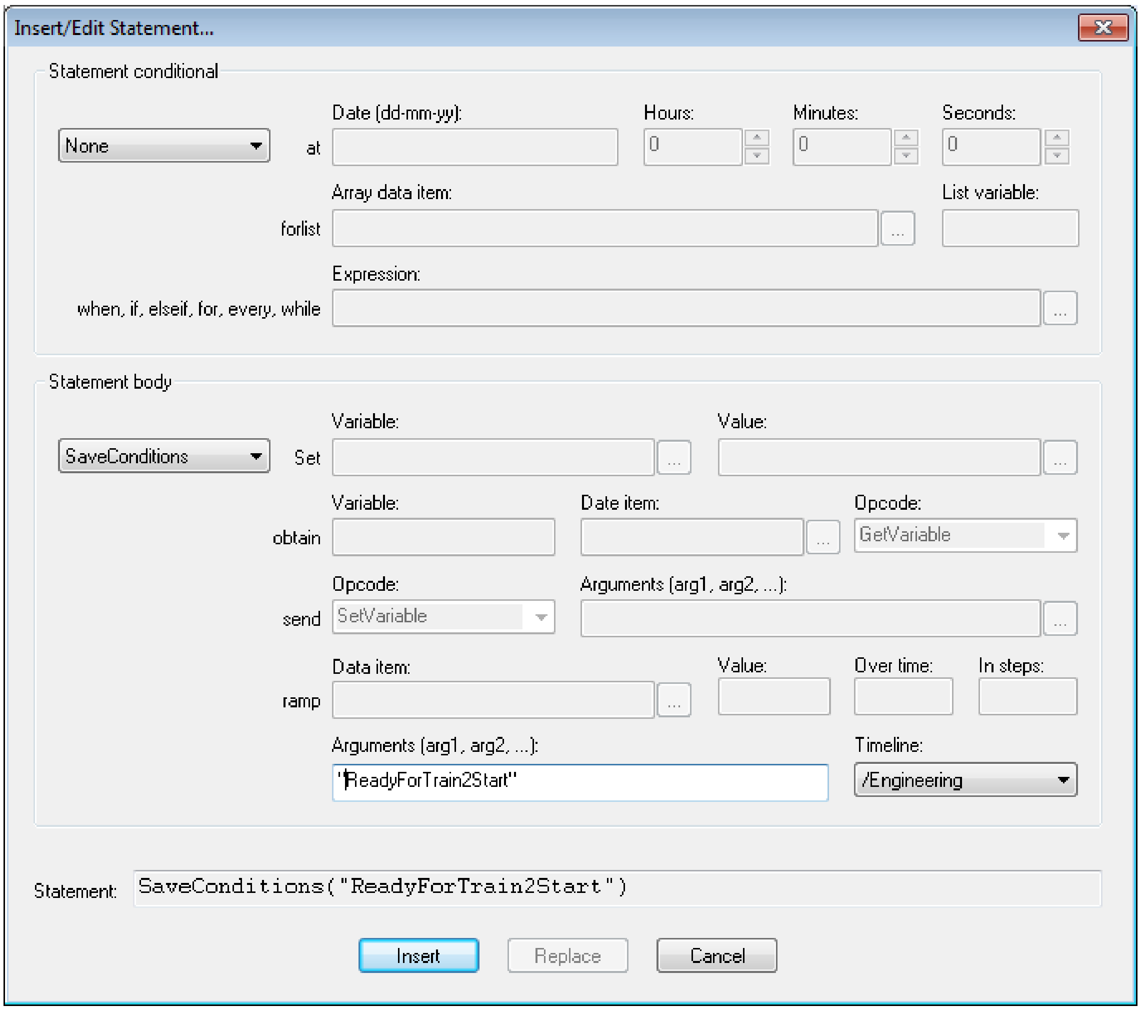
5.2.9 Saving files

There are a number of commands to save K-Spice files — one command for each file type. Each command follows the same structure. The save file commands are as follows:

- SaveModel — save the model topology — blocks, connections etc. This command required an *Engineering* license.
- SaveParameters — save the physical information about the blocks — lengths, performance curves etc.

- SaveTrends — save the history configuration. It is normally loaded automatically with the model
- SaveConditions — save an initial condition.
- SaveHistory — save history data. This would normally be loaded in conjunction with a compatible condition.
- SaveSnapshot — save a previously saved snapshot.

Figure 24 Insert/Edit Statement — SaveConditions



The above **SaveConditions** statement appears in the script as follows:

```
SaveConditions("ReadyForTrain2Start")
```

It is possible to save a file into a specific timeline as follows:

```
SaveConditions("ReadyForTrain2Start","/OTS")
```

5.2.10 Set

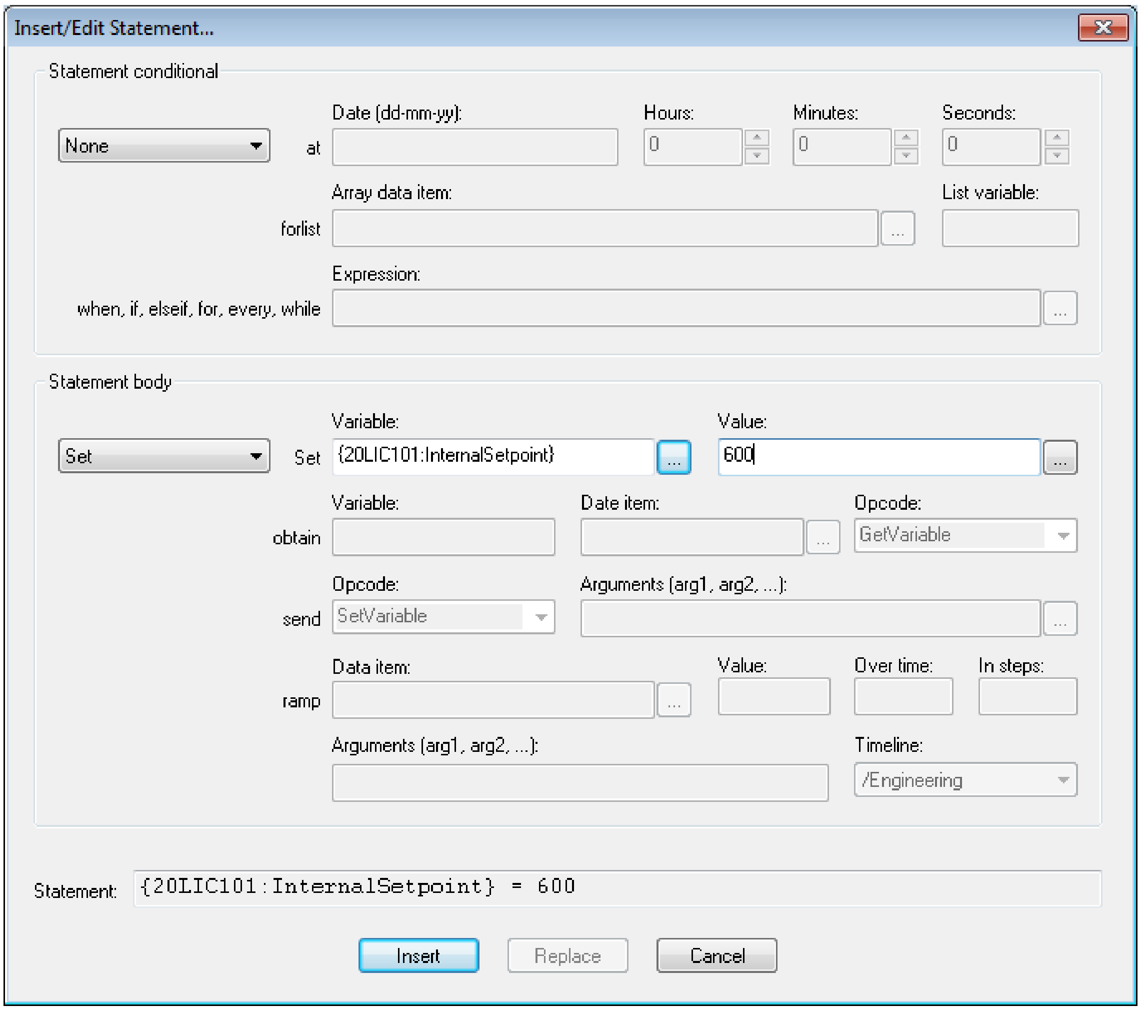
The **Set** operation allows the user to set the value of an application data item or local MCL variable (**Variable:**) to a fixed value, the value of a local MCL variable or a model data item value (**Value:**).

The **Variable:** field can either be a manually entered local MCL variable or set to a model data item name using the first **...** button to open the Insert data item dialog to find the required data item.

The **Value:** field can be a fixed value, a local MCL variable or a model data item set via the second **...** button.

In the example below the data item *{20LIC101:InternalSetpoint}* will be set to 600.

Figure 25 Insert/Edit Statement — Set



Variable: {20LIC101:InternalSetpoint}
Value: 600
Statement: {20LIC101:InternalSetpoint} = 600

The above **Set** statement appears in the script as follows:

```
{20LIC101:InternalSetpoint} = 600
```

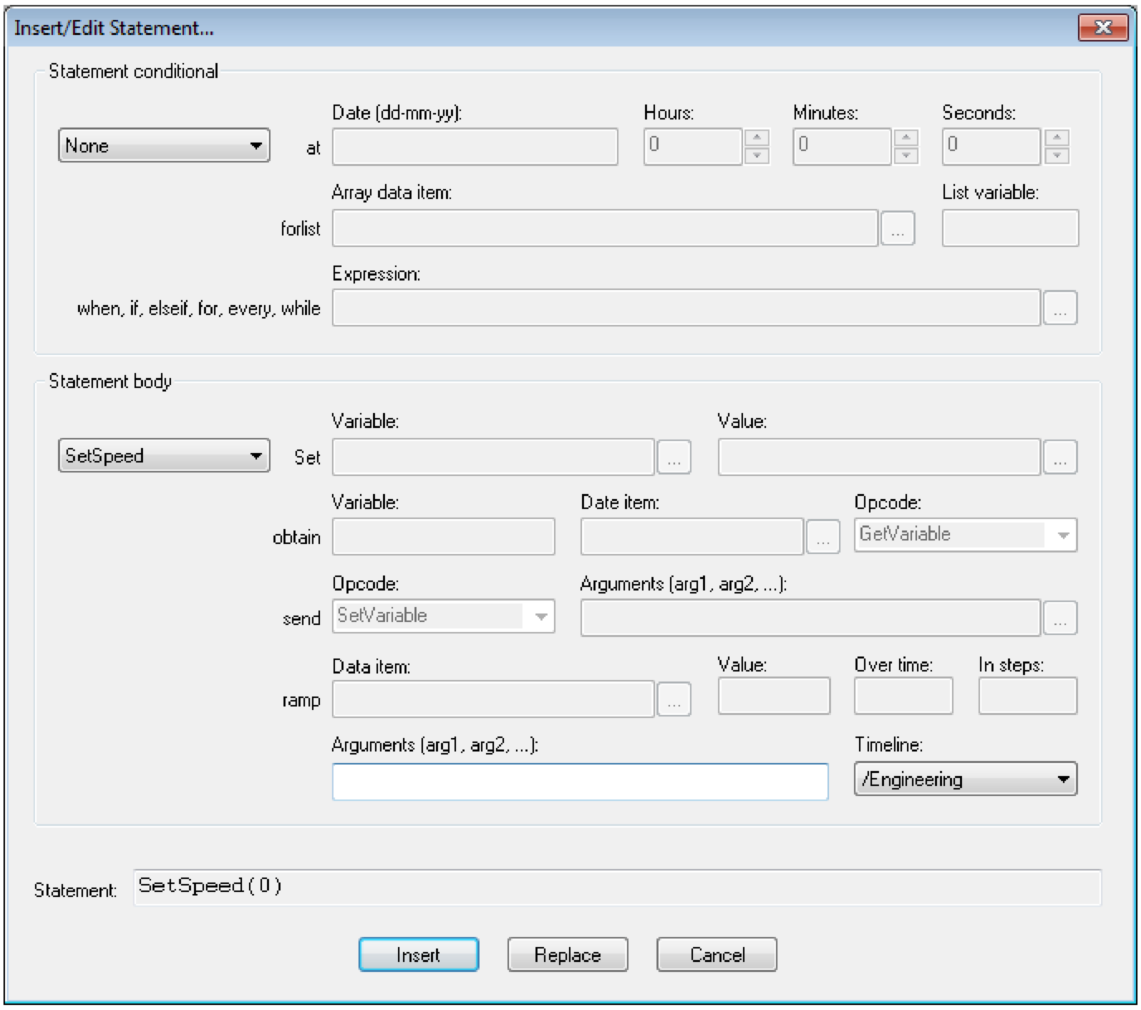
The following **Set** example sets the local MCL variable “Output” to the value of the “ControllerOutput” data item of controller 20LIC101:

```
Output = {20LIC101:ControllerOutput}
```

5.2.11 SetSpeed

The **SetSpeed** operation allows the user to set the speed of the selected timeline. If no timeline is defined the currently selected timeline will be used.

Figure 26 Insert/Edit Statement — SetSpeed



Statement conditional: None, Hours: 0, Minutes: 0, Seconds: 0

Statement body: SetSpeed; obtain Opcode: GetVariable; send Opcode: SetVariable; Timeline: /Engineering

Statement: SetSpeed(0)

The above **SetSpeed** statement sets the requested speed of the current timeline to the maximum set in the timeline properties. The model will be run at this speed or as fast as the hardware permits if this is slower. The script appears as follows:

```
SetSpeed(0)
```

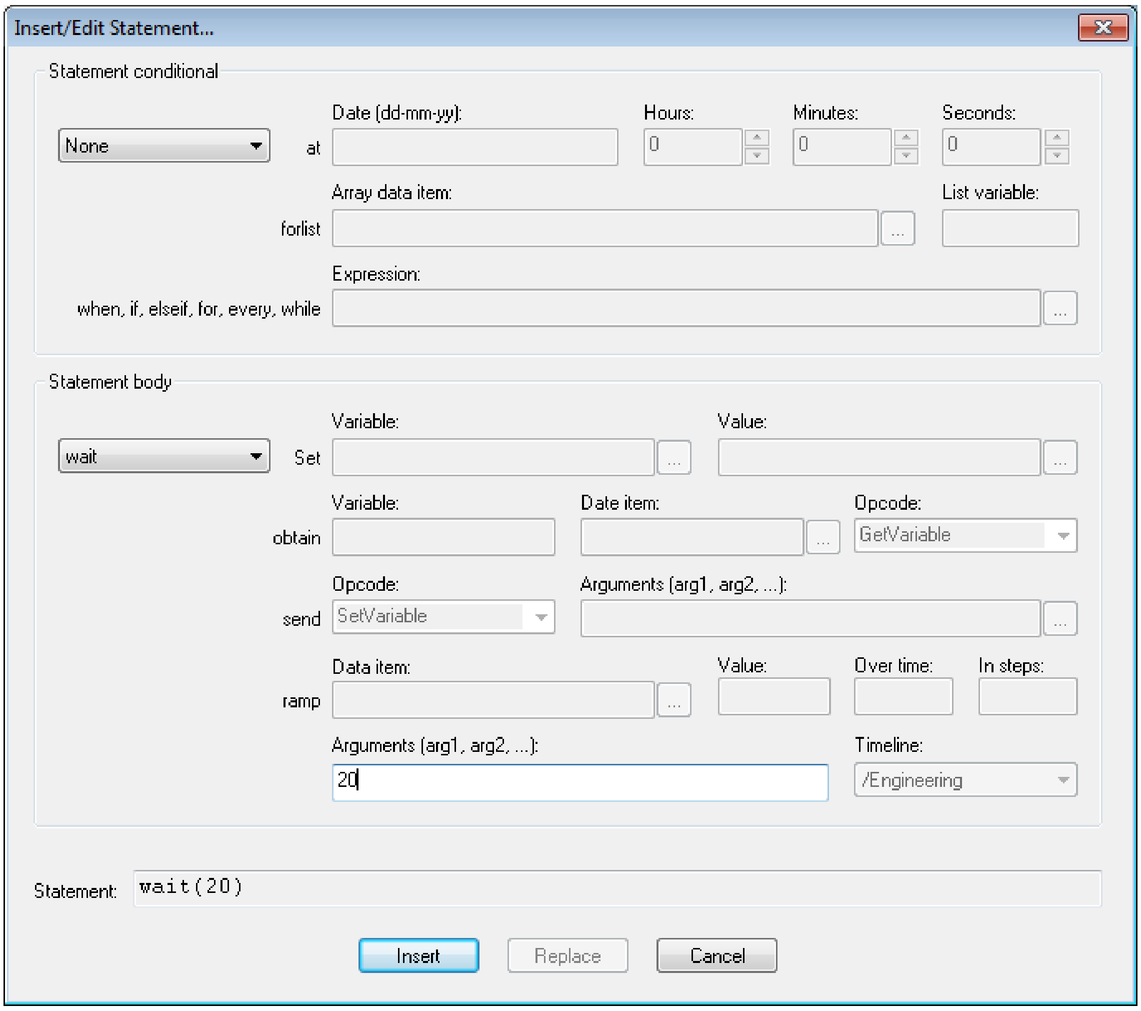
The following **SetSpeed** example sets the speed to real time (x1) on the “Tutorial” timeline:

```
SetSpeed(1,"/Tutorial")
```

5.2.12 Wait

The **Wait** operation causes the execution of the MCL script to pause for a defined time. By default, this pause is based upon execution time (in seconds) of the model.

Figure 27 Insert/Edit Statement — Wait



The above **Wait** statement appears in the script as follows:

```
wait (20)
```

To specify a time in hh:mm:ss format use the standard time formatting for MCL as follows:

```
wait ([00:00:20.00])
```

Sometimes it is useful to have the wait statement wait using real time rather than model time. This could be while waiting for a timeline to load, or some specific operation not tied to the timeline execution, to complete. To do this the additional optional parameter “1” is added to the command:

```
wait (20,1)
```