
MeVisLab Reference Manual



MeVisLab Reference Manual

Copyright © 2003-2024 MeVis Medical Solutions
Published 2024-09-10

Table of Contents

1. Introduction	13
1.1. About the MeVisLab Reference Manual	13
1.2. Associated Documents	13
2. MeVisLab User Interface	14
2.1. Overview	14
2.2. Views	15
3. Modules and Networks	18
3.1. Types of Modules	19
3.2. Module Network Panels	20
3.3. Connector and Connection Types	20
3.4. Connecting, Disconnecting, Moving, and Replacing Connections	24
3.4.1. Connecting Modules	24
3.4.2. Disconnecting Modules	34
3.4.3. Moving Connections	37
3.4.4. Replacing Connections	39
3.5. Mouse Pointers	40
3.6. Mouseover Information	41
3.7. Module Halo	44
3.8. Module Highlighting	45
3.9. Module Handling	47
3.9.1. Module Context Menu	47
3.9.2. Additional Inputs	54
3.9.3. Show Internal Network (Macro Modules)	55
3.10. Network Handling	56
3.10.1. Network Context Menu	56
3.10.2. Connections Context Menus	57
3.11. Using Groups	58
3.11.1. Creating Groups and Adding/Removing Modules	59
3.11.2. Editing, Converting, and Deleting Groups	61
3.11.3. Copying Groups Including Modules	61
3.12. Using Notes	62
3.12.1. Creating Notes	62
3.12.2. Handling Notes	63
3.12.3. Editing and Deleting Notes	63
3.12.4. Copying Notes Including Text	65
3.13. Using the Mini Map	65
3.14. Network Quick Search	66
3.15. Network Selector	69
3.16. Network Preview	69
3.17. Network Mouse Gestures	69
3.17.1. Gesture for Closing the Current Network	70
3.17.2. Gesture for Closing the Current Network Without Requester	70
4. Menu Bar	72
4.1. File Menu	72
4.1.1. New	72
4.1.2. Open	72
4.1.3. Close	72
4.1.4. Close all	73
4.1.5. Save	73
4.1.6. Save As	73
4.1.7. Save Copy As	73
4.1.8. Revert To Saved	73
4.1.9. Recent Files	73
4.1.10. Open Most Recent File	73
4.1.11. Run Project Wizard	73

4.1.12. Create Local Macro	73
4.1.13. Add Local Macro	75
4.1.14. Open File in MATE	75
4.1.15. Show MATE	75
4.1.16. Run ToolRunner	75
4.1.17. Run TestCaseManager	75
4.1.18. Recent Test Cases	75
4.1.19. Run Most Recent Test Case	76
4.1.20. Restart with Current Networks	76
4.1.21. Quit	76
4.2. Edit Menu	76
4.2.1. Undo	76
4.2.2. Redo	76
4.2.3. Clear Undo History	76
4.2.4. Cut	77
4.2.5. Copy	77
4.2.6. Paste	77
4.2.7. Duplicate	77
4.2.8. Delete	77
4.2.9. Select All	77
4.2.10. Deselect All	77
4.2.11. Invert Selection	77
4.2.12. Align / Distribute	77
4.2.13. Auto Arrange Selection	78
4.2.14. Reload Selected Modules	78
4.3. Preferences	78
4.3.1. Preferences — General	79
4.3.2. Preferences — Packages	81
4.3.3. Preferences — Module Groups	82
4.3.4. Preferences — Supportive Programs	83
4.3.5. Preferences — Paths	84
4.3.6. Preferences — Scripting	85
4.3.7. Preferences — Network Appearance	86
4.3.8. Preferences — Network Interaction	91
4.3.9. Preferences — Error / Debug Handling	92
4.3.10. Preferences — Shortcuts	94
4.4. Modules Menu	94
4.5. Applications Menu	95
4.6. Extras Menu	95
4.6.1. Reload Module Database (Keep Cache)	96
4.6.2. Reload Module Database (Clear Cache)	96
4.6.3. Reload Imported Python Modules	96
4.6.4. Show Global MDL Definitions...	96
4.6.5. Run Module Tests...	97
4.6.6. Run Tests On Selection..	98
4.6.7. Generate Module Reference for User Packages (HTML)	98
4.6.8. Show Widget Explorer	98
4.6.9. Debug Widgets	100
4.6.10. Show Connector Details	100
4.6.11. Show Image Connector Preview	100
4.6.12. Clear Image Cache	101
4.7. Scripting Menu	101
4.7.1. Show Scripting Console	101
4.7.2. Scripting Context Menu	103
4.7.3. Edit Network Script	104
4.7.4. Start Network Script	104
4.8. User Scripts	104
4.8.1. Example Scripts	104

4.8.2. Run User Script...	104
4.8.3. Run Last User Script:	104
4.8.4. Run Recent User Script	105
4.8.5. Example Scripts	105
4.9. View Menu	105
4.9.1. View All	105
4.9.2. Zoom To Selection	105
4.9.3. Zoom In	105
4.9.4. Zoom Out	105
4.9.5. Zoom 100%	105
4.9.6. Layout	106
4.9.7. Toolbars	107
4.9.8. Views	108
4.10. Networks Menu	109
4.10.1. Close	109
4.10.2. Close All	109
4.11. Panels Menu	109
4.11.1. Panels Stay In Front Of Main Window	110
4.11.2. Hide Panels Of Invisible Networks	110
4.11.3. Close All Panels	110
4.11.4. Close Panels Of Current Network	110
4.11.5. Minimize All Open Panels	110
4.11.6. Show All Minimized Panels	110
4.11.7. Working with the Panel List	110
4.12. Help Menu	111
4.12.1. (Search in documentation and menu entries)	111
4.12.2. Show Help Overview	111
4.12.3. Browse Help Pages	111
4.12.4. Welcome	111
4.12.5. About	112
4.12.6. Enter License	112
5. Toolbar	113
5.1. File Operations	113
5.2. Edit	113
5.3. Zooming	113
5.4. Script Debugging	113
5.5. Quick Search	113
5.6. Align / Distribute	114
6. Bottom Bar	115
6.1. Loop! indicator	115
6.2. ML Cache	115
6.3. Stop Button	115
6.4. Toggle Layout	115
7. Background Tasks	116
8. Debug Output	119
9. ML Parallel Processing Profiler View	120
10. Module Browser	122
11. Module Inspector	123
11.1. Fields	123
11.1.1. Editing Field Values	123
11.1.2. Module Inspector Fields Context Menu	124
11.2. Files	127
11.2.1. Module Inspector Files Context Menu	127
11.3. Tree	127
11.3.1. Tree Context Menu	128
11.4. About	129
11.5. Related	129
11.5.1. Related Context Menu	131

11.6. Scripting	131
12. Module List	133
13. Module Search	134
13.1. Module Search	134
13.2. Advanced Search	134
13.3. Module Search Result Context Menu	136
13.3.1. General Options	136
13.3.2. Additional Options for Macro Modules	136
13.4. Search in Network	137
14. Network Field WatchList	138
15. Output Inspector	139
16. Parameter Connections Inspector	141
16.1. Parameter Connections Inspector View	141
16.2. Parameter Connections Inspector Context Menu	143
17. Profiling	144
17.1. Introduction to Profiling	144
17.2. Using Profiling	145
17.2.1. Modules	146
17.2.2. Fields	148
17.2.3. Functions	148
18. Recent Outputs	152
19. Screenshot Gallery	153
19.1. Screenshot Gallery	153
19.2. Screenshot Gallery Context Menu	153
19.3. Movies in the Screenshot Gallery	154
20. Scripting Console	155
21. Scripting Assistant	156
22. Search in Network	157
23. Search in Documentation	159
24. Snippets List	161
25. Project Wizard	162
25.1. Project Wizard Introduction	162
25.2. Modules (C++) Wizard	163
25.2.1. First C++ Module Wizard Dialog	163
25.2.2. Inventor Module	165
25.2.3. ML Module	166
25.3. Modules (Scripting) Wizard	171
25.4. Module Field Interface	172
25.5. Packages	174
25.6. Example .Wiz File (Inventor Module), indented for a better readability	175
26. MATE	176
26.1. What is MATE?	176
26.2. Text Editor User Interface	177
26.3. Menu Bar	178
26.4. Module Menu	181
26.5. Outline Area	182
26.6. Edit Area	182
26.7. Preferences	184
26.8. Python Debugger	185
26.9. Module Help Editor	188
26.9.1. Context Menus	190
26.9.2. Formatting	191
26.9.3. How it Works	193
26.9.4. Internal HTML Preview	194
26.10. Session Management	195
26.11. Project Workspaces	196
26.11.1. Project Types	196
26.11.2. Context Menu	197

26.11.3. Views	198
26.11.4. File Locator	199
26.12. GUI Editor	199
26.13. Scripting	200
26.14. Pylint Integration	201
26.14.1. Installation	201
26.14.2. Usage	201
27. Tips and Tricks	203
27.1. Command-Line Options	203
27.2. MeVisLabPackageScanner.exe	205
27.3. EatDicom.exe (deprecated)	205
27.4. Connecting Inventor Engines to ML Modules	206
27.5. Using SyncFloat to Reduce System Load	207
27.5.1. Case 1: Two Inventor and One ML Module Connected in a Circle	207
27.5.2. Case 2: A Macro Module (Including an Inventor Module) and Another Inventor Module Connected in a Circle	208
27.6. Printing MeVisLab Networks	209
27.7. Multi-threading in MeVisLab	209
27.7.1. Multi-threading in the ML	209
27.7.2. Background Tasks	210
27.7.3. Modules for Multi-threading	210
27.8. Set Open Inventor Override Flag (Inventor Modules)	210
28. Settings File and Environment Variables	213
28.1. Possible Locations of mevislab.prefs	213
28.2. Options in mevislab.prefs	213
28.3. Environment variables	217
29. Coding Style Guide	219
29.1. General Rules	219
29.2. Naming Conventions	219
29.2.1. Code	220
29.2.2. Files	220
29.2.3. MeVisLab/ML Identifiers	220
29.2.4. MDL Layout	220
29.3. General Programming	221
29.4. C++	221
29.5. Python	221
29.6. ML Programming	222
29.7. MDL	222

List of Figures

2.1. Typical MeVisLab User Interface	14
2.2. View Docked in the Views Area	15
2.3. Floating View	16
2.4. Moving View to Another Position in Views Area	16
2.5. Stacked Views	17
2.6. Resizing a View in the Views Area	17
3.1. Example Network for SynchroView2D with Viewer (Panel), Automatic Panel, and Settings.....	18
3.2. Modules with Network Panels	20
3.3. View2D with Connected "Invisible" Open Inventor Connector	21
3.4. Compatible Connectors for CSOVisualizationSettings Type	23
3.5. Compatible Connectors for CSOList Type	23
3.6. Compatible Connectors for ML Image Type	23
3.7. Parameter Connection — Panel Mouseover	43
3.8. Connector Image Preview	44
3.9. Connector Detail Info and Image Preview	44
3.10. Module Context Menu	48
3.11. Module Context Menu — Show Window	48
3.12. Automatic Panel	48
3.13. Panel Defined in MDL	49
3.14. Module Context Menu — Edit Instance Name	50
3.15. Modules and Instance Names	50
3.16. Module Context Menu — Show Example Network	50
3.17. Dependency Walker	51
3.18. Module Context Menu — Tests	53
3.19. Module Context Menu — Related Files	54
3.20. View3D With Visible Inventor Inputs (Default)	54
3.21. View3D With Hidden Inventor Inputs	54
3.22. RegionGrowingMacro — Internal Network	55
3.23. Network Context Menu	56
3.24. Parameter Connection Context Menu	57
3.25. Module with Internal/Self-Connected Parameter Connection	57
3.26. Data Connection Context Menu	58
3.27. Network Context Menu — Adding Groups	60
3.28. Network Context Menu — Adding to a Specific Group	60
3.29. Group Context Menu	61
3.30. Note (expanded)	62
3.31. Creating a Note	62
3.32. Dialog for editing of notes, with context menu	63
3.33. Note (collapsed)	63
3.34. Note Context Menu	64
3.35. A note displayed as a network comment	64
3.36. Note in a Group	65
3.37. Mini Map	65
3.38. Navigating in the Mini Map	65
3.39. Parent Navigation Frame for Macro Modules	66
3.40. Parent Navigation Frame Context Menu	66
3.41. Network Quick Search	67
3.42. Network Quick Search — Options	67
3.43. Network Quick Search — Show All Results	68
3.44. Network Quick Search — Highlight Results	68
3.45. Network Selector in Action	69
3.46. Network Selector in Action	69
3.47. Trail of Unrecognized Mouse Gesture	70
3.48. Mouse Gesture for Closing the Current Network	70
3.49. Mouse Gesture for Closing the Current Network Without Requester	71

4.1. File Menu	72
4.2. Local Macro Creation	74
4.3. Modules Connected to Outer Macros	75
4.4. Edit Menu (Windows example)	76
4.5. Align / Distribute	77
4.6. Preferences — General	79
4.7. Preferences — Packages	81
4.8. Preferences — Module Groups	82
4.9. Preferences — Supportive Programs	83
4.10. Preferences — Paths	84
4.11. Preferences — Scripting	85
4.12. Preferences — Network Appearance	86
4.13. Preferences — Network Interaction	91
4.14. Preferences — Error / Debug Handling	92
4.15. Preferences — Shortcuts	94
4.16. Modules Menu	95
4.17. Extras Menu	95
4.18. MeVisLab Global MDL Definitions	97
4.19. Module Selection	98
4.20. MeVisLab Widget Explorer - Attributes Inspector	99
4.21. MeVisLab Widget Explorer - Style Sheet Editor	100
4.22. Scripting Menu	101
4.23. Scripting Editor	102
4.24. Scripting Example	103
4.25. Scripting Context Menu	103
4.26. View Menu	105
4.27. View — Layout Submenu	106
4.28. Store Current Layout	107
4.29. Edit User Layouts	107
4.30. View — Toolbars Submenu	107
4.31. View — Views Submenu	108
4.32. Networks Menu	109
4.33. Panels Menu	109
4.34. Panels Menu — Listing all Open Panels	110
4.35. Help Menu	111
5.1. Toolbar	113
5.2. Quick Search Options	113
5.3. Quick Search — Info Box	114
5.4. Quick Search History	114
6.1. Bottom Bar	115
7.1. ML Background Tasks	116
7.2. ML Background Tasks — Context Menu	116
7.3. ML Background Tasks — Context Menu for Running Processes	117
7.4. Warning for Running Background Tasks	117
7.5. Save in Background for <code>GVRVolumeSave</code>	118
8.1. Debug Output	119
8.2. Context Menu	119
9.1. Parallel Processing View Overview	120
9.2. Parallel Processing View Details	121
10.1. Module Browser	122
11.1. Module Inspector — Fields	123
11.2. Automatic Panel	123
11.3. Module Inspector — Edit Boolean	123
11.4. Module Inspector — Edit Color	124
11.5. Module Inspector — Edit Text	124
11.6. Module Inspector — Edit Values	124
11.7. Module Inspector Fields Context Menu	125
11.8. Module Inspector — Files	127

11.9. Module Inspector Files Context Menu	127
11.10. Module Inspector — Tree	128
11.11. Module Inspector Tree Context Menu	128
11.12. Show Available MDL Tags	129
11.13. Module Inspector — About	129
11.14. Module Inspector — Related	130
11.15. Module Inspector Related Context Menu	131
11.16. Module Inspector — Scripting	132
12.1. Module List	133
13.1. Module Search with Demo Entry	134
13.2. Module Search — Advanced	135
13.3. Module Search — Searching In	135
13.4. Module Search — Operators	135
13.5. Module Search Results — Context Menu	136
13.6. Search in Network	137
14.1. Network Field WatchList	138
15.1. ML Image Inspector	139
15.2. ML Image Inspector: 3D View	140
15.3. ML Image Inspector: Detailed Information	140
16.1. Parameter Connections Inspector View	141
16.2. Parameter Connection Example — View2D and View3D	142
16.3. Parameter Connection Example — View2DExtensions	142
16.4. Parameter Connection Example — Navigating between Fields	143
16.5. Parameter Connections Inspector Context Menu	143
17.1. Functions to be Profiled	144
17.2. Profiling	145
17.3. Profiling Report	146
17.4. Profiling Modules	147
17.5. Profiling — Heading Configuration	147
17.6. Profiling Fields	148
17.7. Profiling Functions as Flat Profile	149
17.8. Profiling Functions as Call Graph	149
17.9. Functions with Filters Visible	150
18.1. Recent Outputs	152
19.1. Screenshot Gallery	153
19.2. Screenshot Gallery Context Menu	154
21.1. Scripting Editor	156
22.1. Scripting Editor	157
22.2. Scripting Editor	157
22.3. Scripting Editor	158
23.1. Search in Documentation	159
23.2. Search in Documentation — ML Example	159
23.3. Search in Documentation — MDL Example	160
23.4. Search in Documentation — Python Example	160
24.1. Snippets List	161
24.2. Snippets List — Context Menu	161
25.1. Project Wizard (no user packages available)	162
25.2. Project Wizard (with user packages available)	162
25.3. First C++ Module Wizard Dialog — ML Module Example	163
25.4. Create an ML Module in a Self-contained Folder	164
25.5. Inventor Type	165
25.6. Imaging Module Properties (New Style)	167
25.7. New Style ML Module	167
25.8. New Style ML Module — Uses Fixed Datatype	168
25.9. New Style ML Module — Uses Datatype Of Input Image	168
25.10. New Style ML Module — Entering The Supported Types	168
25.11. New Style ML Module — Configuring The Input Handling	168
25.12. New Style ML Module — Uses The Same Datatype As	169

25.13. Imaging Module Properties (Classic Style)	169
25.14. Additional ML Module Properties	170
25.15. Project Wizard	172
25.16. Module Field Interfaces	173
25.17. Package Wizard	174
26.1. User Interface	177
26.2. MATE File Menu	178
26.3. MATE Edit Menu	179
26.4. MATE View Menu	180
26.5. MATE Window Menu	180
26.6. MATE Extras Menu	181
26.7. MATE Module Menu — Without Attached Module	181
26.8. MATE Module Menu - With Attached Module	181
26.9. MATE Module Menu — Windows Submenu	181
26.10. MATE Module Menu — Files Submenu	182
26.11. Outline Area	182
26.12. MATE Edit Area	183
26.13. MATE Edit Area — Code Completion for Keywords	183
26.14. MATE Edit Area — Code Completion for Commands Defined in MDL	183
26.15. MATE Edit Area — Context Menu	184
26.16. MATE Preferences	184
26.17. MATE with Python Debugger	186
26.18. MATE Debug Menu	186
26.19. MATE for Module Help	188
26.20. Outline Context Menu	191
26.21. Text Context Menu	191
26.22. Automatically Documented Elements	193
26.23. HTML View	194
26.24. HTML View Decoupling	195
26.25. Decoupled HTML View	195
27.1. MeVisLabPackageScanner Help	205
27.2. Excerpt of the EatDicom help	206
27.3. Dicom Import Log	206
27.4. Field Bridge Example	207
27.5. SyncFloat Example — ML and Inventor Modules	208
27.6. SyncFloat Example — Macro and Inventor Modules	209
27.7. Open Inventor Scene Without Override	210
27.8. Open Inventor Scene With Override	211
27.9. Open Inventor Scene With Ignore Flag (Red)	211
27.10. Open Inventor Scene With Ignore Flag (Blue)	212

List of Tables

1.1. List of MeVisLab Documents	13
3.1. Module Types	19
3.2. Invalid Modules	19
3.3. Connectors	20
3.4. Connecting to an Invisible Connector	22
3.5. Connections	22
3.6. Connecting Modules by Dragging	24
3.7. Dragging a New Connection Generates New Input Connectors to the Sides of Regular Connectors	25
3.8. Even More New Connectors are Available	26
3.9. New Input Connectors are Generated by Positioning the Mouse	27
3.10. Connecting by Moving the Source Module into Proximity	27
3.11. Connecting by Moving the Destination Module into Proximity	28
3.12. Connecting an Inventor Group by Proximity	28
3.13. Connecting to an Inventor Group by Proximity	29
3.14. Appending vs. Prepending to an Inventor Group by Proximity	29
3.15. Connecting a Module by Inserting	30
3.16. Connecting a Module with Two Inputs by Inserting	31
3.17. The Second Input is Connected by Dragging	32
3.18. Variation: First Input is Connected by Proximity	33
3.19. Variation: Second Input is Connected by Inserting into an Existing Connection	34
3.20. Disconnecting by Dragging to Background: Input	35
3.21. Disconnecting by Dragging to Background: Output	36
3.22. Disconnecting by Selection and Pressing DEL	36
3.23. Disconnecting by Context Menu	37
3.24. Move Input Connection	38
3.25. Move Multiple Output Connections	38
3.26. Move Connection Within an Inventor Group	39
3.27. Replace Connection	40
3.28. Mouse Pointers	41
3.29. Mouseover Information	42
3.30. Module Halos — Classic and Alternative	45
3.31. Module Halos Input Output — Classic and Alternative	45
3.32. Highlighting of Selections — Classic Halo	46
3.33. Highlighting of Selections — Alternative Halo	46
3.34. Module Group with Alternative Halo — Selected and Highlighted	46
3.35. Preview of Internal Networks of Macro Modules	47
3.36. Run In Separate Process	52
3.37. Modules in Groups	59
16.1. Connections Symbols	142
17.1. Function Type Icons	150
26.1. Buttons for Debugging	187
26.2. Icons for Debugging	188
26.3. Help Toolbar Buttons	190
26.4. Inline markup	192
26.5. Directives	192
26.6. Roles	193
27.1. Command-Line Options	204

Chapter 1. Introduction

1.1. About the MeVisLab Reference Manual

The MeVisLab (Reference) Manual describes the user interface elements of MeVisLab: the main work area, the menus, the modules and networks, the Views and their options.

1.2. Associated Documents

Besides the document at hand, the following documents are available:

Table 1.1. List of MeVisLab Documents

Title	Contents
Getting Started	Introduction to working with MeVisLab
ML Guide	MeVis Image Processing Library — Programming Guide
ML Reference (HTML only)	MeVis Image Processing Library — API description
MDL Reference	MeVisLab Definition Language (MDL) Panel/GUI Reference
Open Inventor Help	Help for Open Inventor Modules
Open Inventor Reference	Reference for all implemented Open Inventor classes (converted from the original manpages)
Scripting Reference	Scripting Reference for Python in MeVisLab
Toolbox Reference	MeVisLab Toolbox Class Reference for various API libraries
TestCenter Reference	Class Reference for the TestCenter
Package Structure	Information about the package structure in MeVisLab
ToolRunner	Manual for ToolRunner, a stand-alone program for building projects and help files
CMake	Use of CMake in the MeVisLab context

To search in the online documentation, use **Help → Search in Documentation**, see [Chapter 23, Search in Documentation](#).

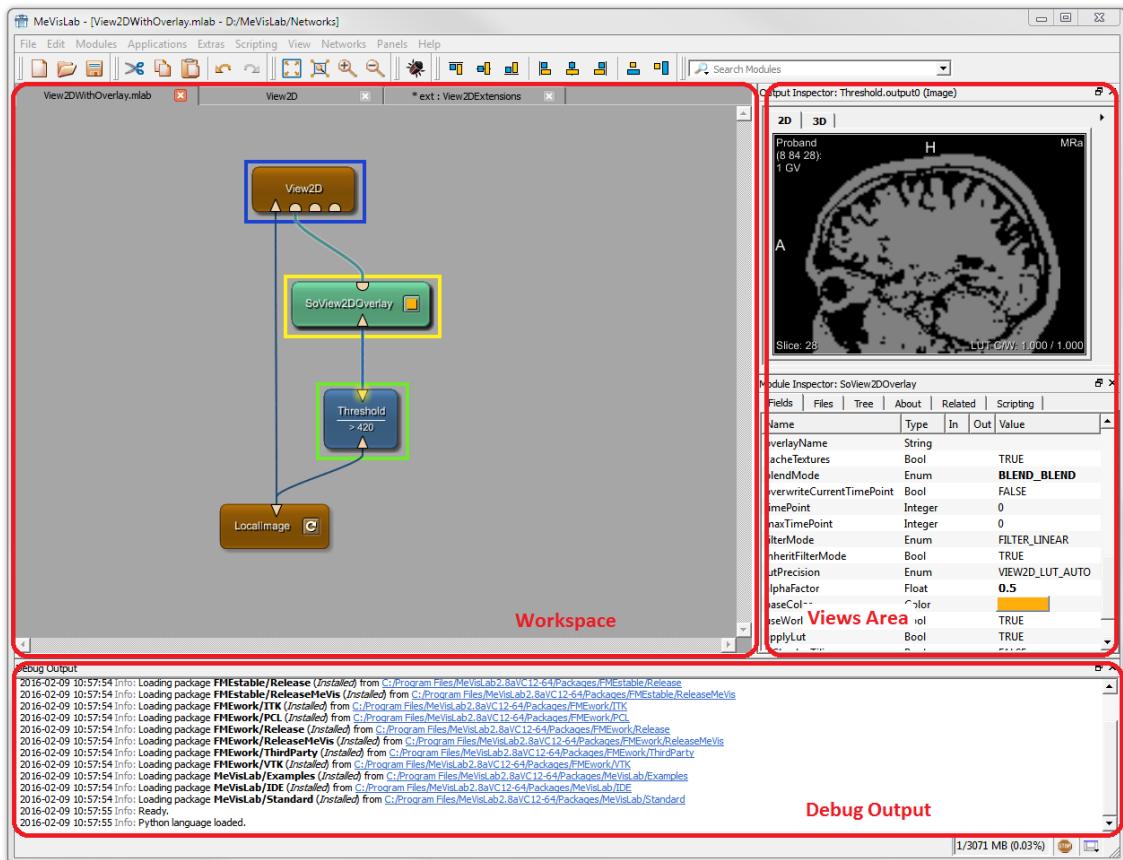
The full list of available documents and resources is available on the Welcome Screen (which can also be opened via **Help → Welcome**).

Chapter 2. MeVisLab User Interface

The layout of the MeVisLab graphical user interface (GUI) heavily depends on the arrangement preferred by the user. Own arrangements can be saved as the “User Default Layout”. In addition, pre-defined GUI layouts can be selected via **View → Layout** or in the bottom bar, see [Chapter 6, Bottom Bar](#).

2.1. Overview

Figure 2.1. Typical MeVisLab User Interface



The user interface offers the following areas:

- the menu bar with typical entries. See [Chapter 4, Menu Bar](#).
- the toolbar with buttons for Edit and Zoom functions. See [Chapter 5, Toolbar](#).
- the workspace with the network display, with tabs for all open networks. See [Chapter 3, Modules and Networks](#).
- the Views area, configurable in the **View → Views** submenu. See [Section 4.9.8, “Views”](#).
- the **Debug Output** (effectively a View), configurable in the **View → Views** submenu. See [Chapter 8, Debug Output](#).
- the bottom bar with information about the used memory space and quick access to layouts. See [Chapter 6, Bottom Bar](#).

2.2. Views

Views can be added and removed via the **View → Views** submenu.

Views (and the toolbar elements) can be moved to another position in the GUI (“docks”) by dragging them around. Either click the “Arrange Windows” icon  or grab the title bar of the View and drag it out or around in the Views area.

Figure 2.2. View Docked in the Views Area

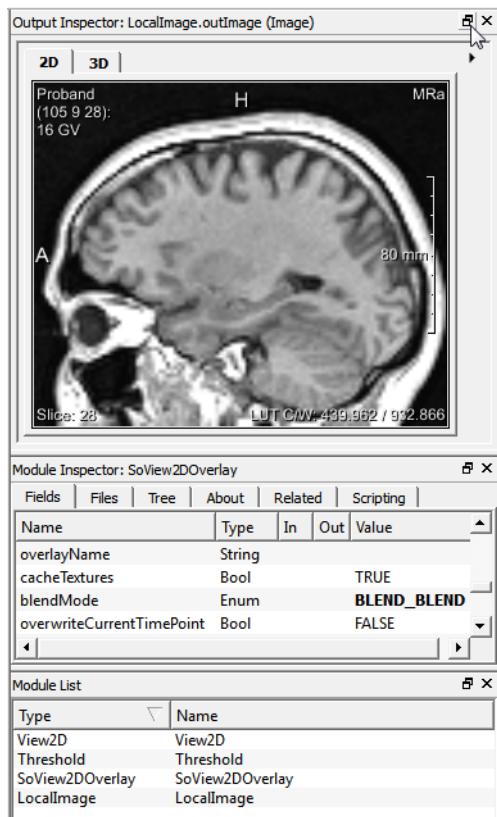
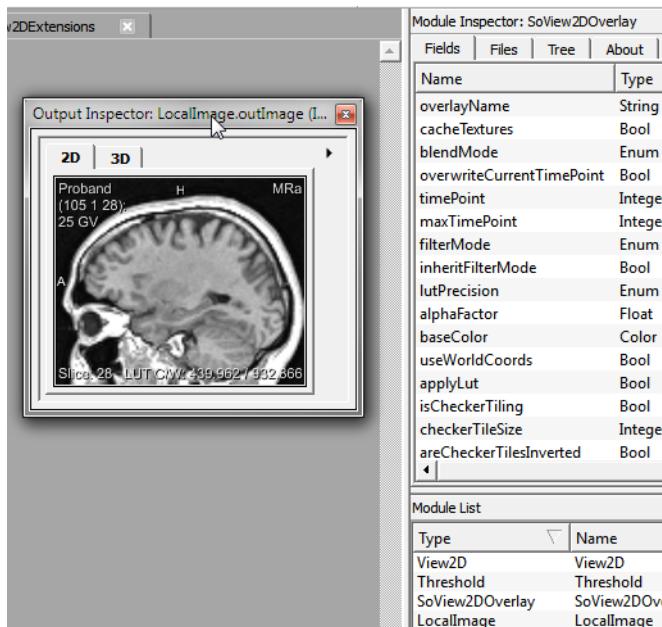
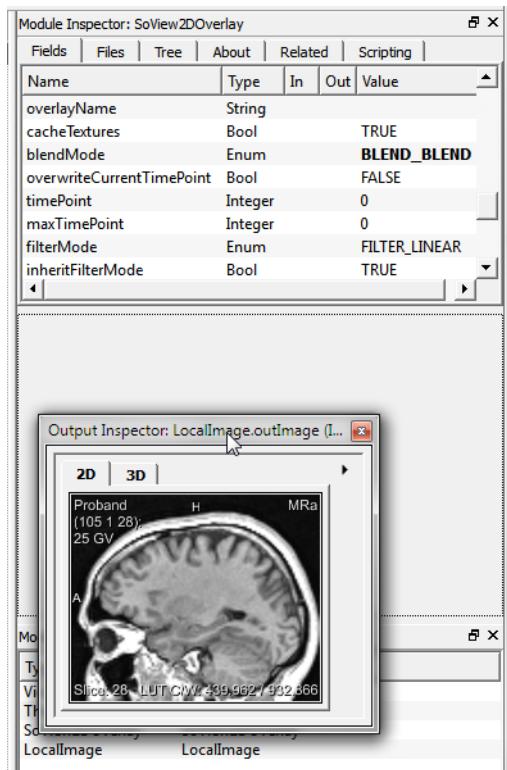


Figure 2.3. Floating View

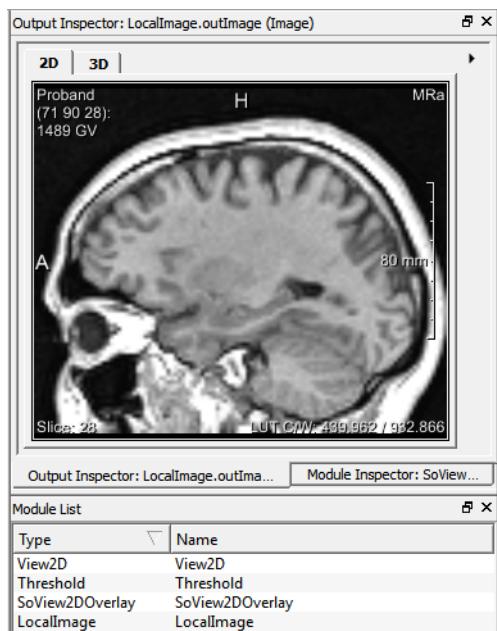
Floating Views can be moved freely on the screen, for example to another monitor. This allows for a larger scaling of Views without affecting the network display.

Before docking, the target area is indicated when hovering the View over areas of the main window.

Figure 2.4. Moving View to Another Position in Views Area

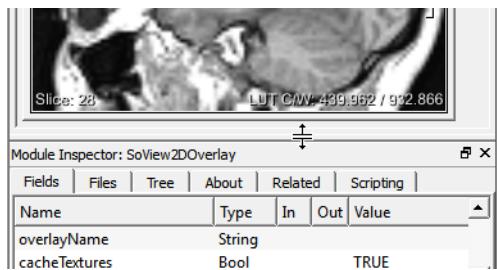
Views can also be stacked above each other by dragging one View onto the other. For each View, a tab will be displayed.

Figure 2.5. Stacked Views



Views can be changed in size by dragging their borders. The resizing is limited by the relative size of the neighboring windows.

Figure 2.6. Resizing a View in the Views Area

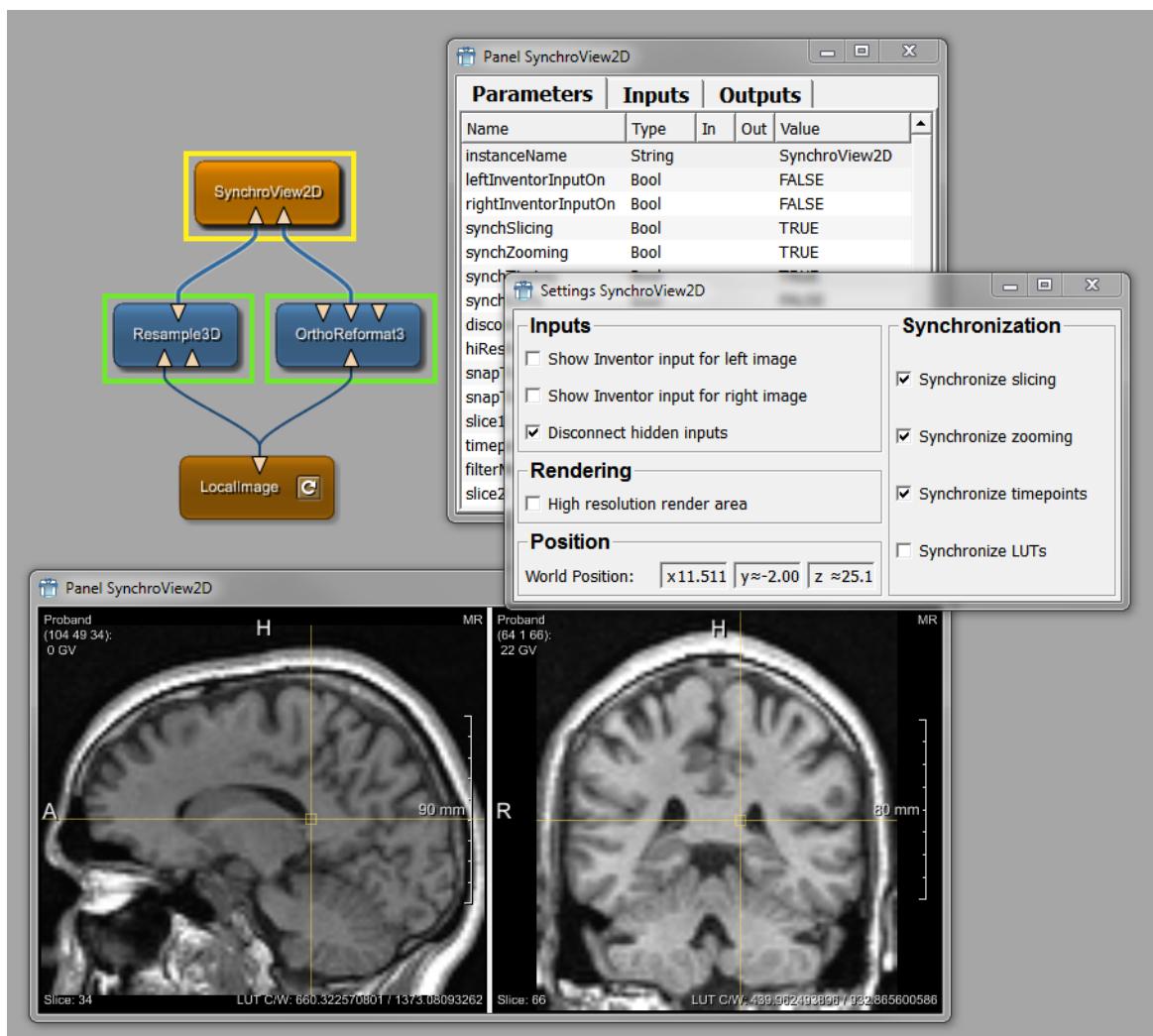


Chapter 3. Modules and Networks

In MeVisLab, the programming of image processing algorithms or interactive image / 3D scene manipulation is basically done by establishing networks that consist of modules and connections between modules. Modules encapsulate certain algorithms written in C++ and provide an interface in MeVisLab in the form of fields. Those fields can represent simple data like numbers or strings, but can also represent more complex data like six dimensional voxel images. Fields of modules of the same type can be connected to form networks that represent algorithms on a higher abstraction layer.

In the following picture, a typical assembly of connected modules in a network, their panels, and viewers can be seen.

Figure 3.1. Example Network for SynchroView2D with Viewer (Panel), Automatic Panel, and Settings



The following information can be found in this chapter:

- [Section 3.1, “Types of Modules”](#)
- [Section 3.2, “Module Network Panels”](#)
- [Section 3.3, “Connector and Connection Types”](#)
- [Section 3.4, “Connecting, Disconnecting, Moving, and Replacing Connections”](#)

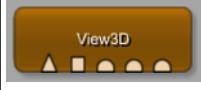
- [Section 3.5, “Mouse Pointers”](#)
- [Section 3.6, “Mouseover Information”](#)
- [Section 3.9, “Module Handling”](#)
- [Section 3.10, “Network Handling”](#)
- [Section 3.11, “Using Groups”](#)
- [Section 3.12, “Using Notes”](#)
- [Section 3.13, “Using the Mini Map”](#)

For module and network shortcuts, see [Section 4.3.10, “Preferences — Shortcuts”](#).

3.1. Types of Modules

There are three types of modules:

Table 3.1. Module Types

Type	Look	Characteristics
ML Module (blue)		page-based, demand-driven processing of voxels
Open Inventor (So) Modules (green)		visual scene graphs (3D)
Macro Module (brown)		combination of other module types, allowing implementing hierarchies and scripted interaction

If a module is invalid, it is displayed in bright red.

The number of warning and error messages that are being printed to the debug console are listed at the upper right corner of the module. Once the debug console is cleared, the warning and error indicators at the module are also cleared. If the module produces information messages, their number is printed in gray at this position. This enables a network or module developer to find the modules in a network that produce messages quickly.

Table 3.2. Invalid Modules

Module Appearance	Explanation
	Invalid module
	Macro containing an invalid module

For information and examples on how to construct networks from modules, please refer to the Getting Started in which image processing pipelines, scene graphs and macro module creation are discussed in detail.

3.2. Module Network Panels

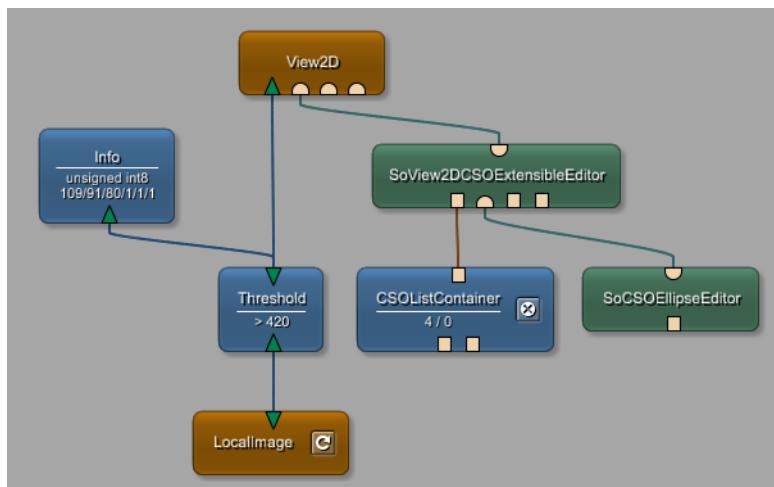
A module can have a simple panel that is rendered in the network and can show a dynamically updated information string and/or a button or checkbox.

This is useful to quickly access the state of a module or to trigger some functionality of the module without having to open the module's panel.

See Section 2.10, “NetworkPanel” for more information.

In the figure below, the `Info` module shows the image's data type and extent, the `Threshold` module shows the comparison operator and the threshold value, the `LocalImage` modules offers a button to reload the image, and the `CSOListContainer` shows the number of CSOs and CSOGroups and offers a button to remove all those objects.

Figure 3.2. Modules with Network Panels



3.3. Connector and Connection Types

In MeVisLab, three types of connectors are defined.



Note

In principle, every module type can have any kind of connector.

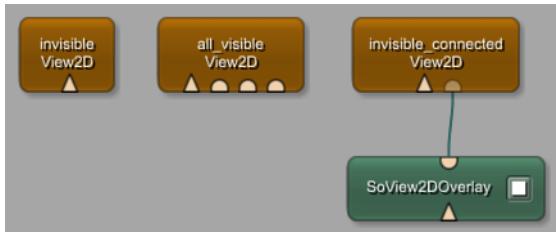
Table 3.3. Connectors

Look		Definition
	triangle	ML images
	half-circle	Inventor scene
	square	Base objects: pointers to data structures

ML image connectors can be set to display their state, see [Section 4.3.7, “Preferences — Network Appearance”](#).

Connectors can exist in a semi-transparent design. This is the case if a connector is hidden but connected. An example for a module with Inventor inputs that can be hidden is the `View2D` modules (see [Section 3.9.2, "Additional Inputs"](#)).

Figure 3.3. View2D with Connected "Invisible" Open Inventor Connector



Note

Modules with hidden input / output fields can be made to show those fields in different ways.

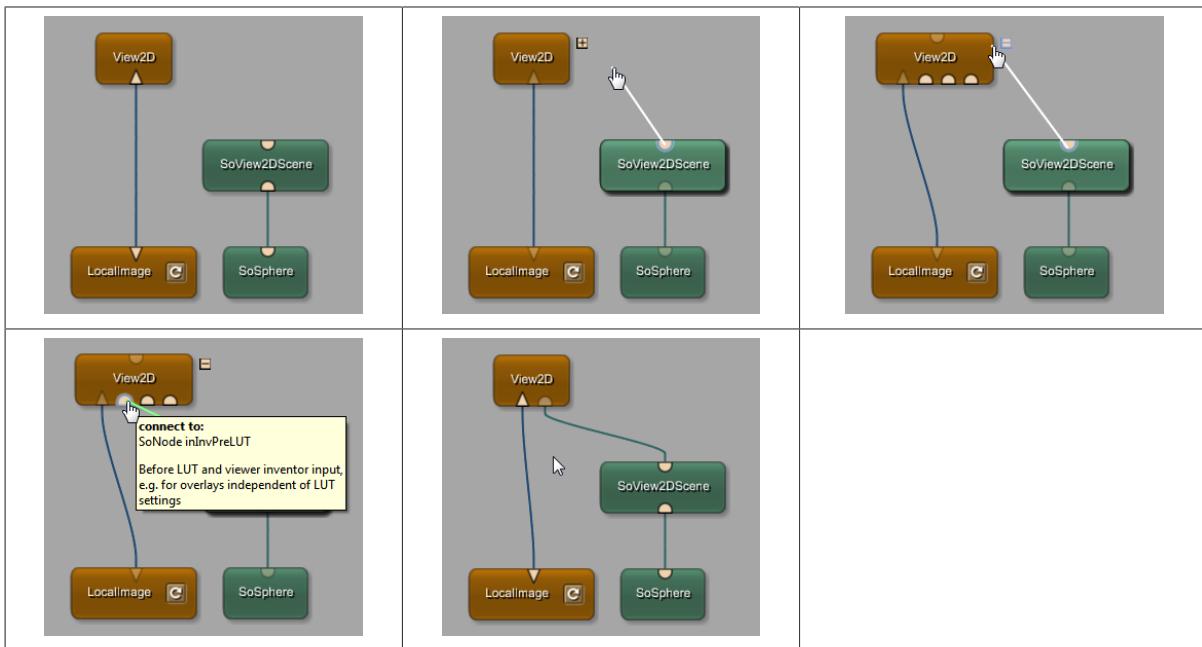
Some modules provide the option to toggle the visibility of hidden fields in the module's context menu (e.g., `View2D`, `View3D`).

Other modules might offer a field on their panel to adjust the number of shown connectors (e.g., `Switch`).

All modules reveal their hidden input and output connectors when starting to draw a connection in the network. If a module has hidden connectors, an icon appears at the top right hand corner of such a module. When hovering with the mouse cursor over that icon while still drawing the connection, the module shows all its connectors that are interactively connectable now if the connection is compatible. On establishing the connection, all other hidden connector disappear again.

On pressing the `SPACE`, the network is rendered in a special information mode where also all invisible connectors are revealed. Pressing `CTRL+SPACE` shows invisible connectors only. To toggle back to normal view press `SPACE` again.

To connect to an otherwise invisible connector, start to drag at a source connector. When the drag has been started, modules with invisible connectors show a plus sign at their upper right corner. Move over this plus sign with the connection still dragged; this reveals the hidden connectors of that module. Now, the connection can be established by dropping the connection on the desired destination connector.

Table 3.4. Connecting to an Invisible Connector

In a MeVisLab network, we distinguish between a data connection and a parameter connection.

A data connection connects modules by their input/output connectors. Those connections carry MLI images, Inventor scene objects, or general Base objects.

A parameter connection connects fields of modules. Such a connection can also connect fields of the same module with each other.

Table 3.5. Connections

Type	Look	Characteristics
Data connections (connector connections)		The direct connection between connectors. Depending on which connectors are involved, the connection is rendered in a different color: blue for ML, green for Open Inventor, brown for Base.
Parameter connections (field connections)		Connections created by connecting parameter fields within or between modules. For more information, see Section 3.10.2, "Connections Context Menus" .

Data connections are established for example by clicking on a connector and drawing the connection to the other connector. Only connectors of the same kind can be connected.

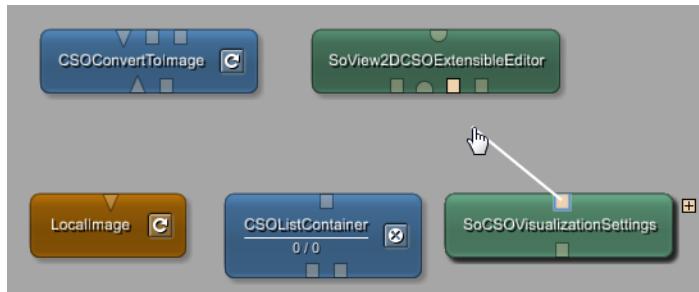
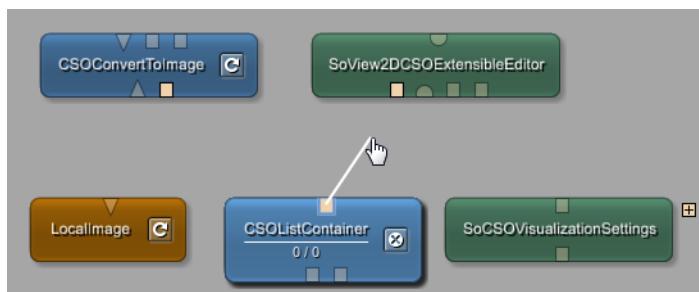
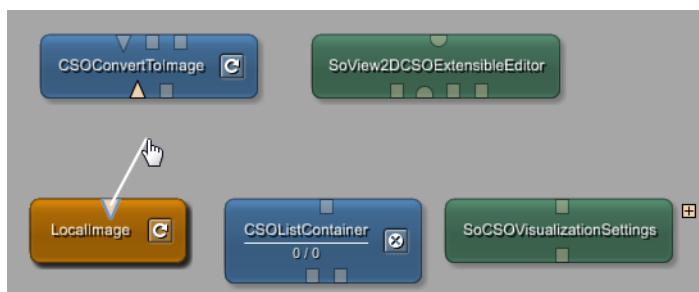


Note

Refer to [Section 3.4, "Connecting, Disconnecting, Moving, and Replacing Connections"](#) for more detailed information on different methods to connect and to disconnect modules.

When interactively connecting Base fields, an internal type system checks whether the particular Base connection is possible.

MeVisLab checks the data types of all available connectors while drawing a connection. Incompatible connectors are rendered in a faded-out style while compatible connectors remain clearly visible.

Figure 3.4. Compatible Connectors for CSOVisualizationSettings Type**Figure 3.5. Compatible Connectors for CSOList Type****Figure 3.6. Compatible Connectors for ML Image Type****Note**

Base connectors can have different data types; connecting these incompatible connectors is possible only via scripting and results in the connection being drawn in red. For more information on Base connectors, see the Getting Started, chapter “A Note on Base Types Checks”.

Parameter connections are established similarly, by clicking on a field on a panel and drawing the connection to another field (on the same panel or another one). For details on parameter connections, see the Getting Started, chapter “Parameter Connection for Synchronization”.

Parameter connections can be moved between fields by clicking on the connected connector/field and pressing **SHIFT** while dragging it to another field. The other connected fields will be updated accordingly.

**Tip**

To abort the interactive establishing and removing of connections between modules (and the horizontal moving of connections), press **ESC**. Alternatively, abort the establishing by drawing the connection a) to a connector of the wrong type (will be displayed in red) or b) back to the output connector and release the mouse. The new connection will not be drawn, and no existing connections will be removed.

3.4. Connecting, Disconnecting, Moving, and Replacing Connections

MeVisLab offers multiple ways of connecting or disconnecting modules in a network. Modules can be connected to each other by connecting their parameter fields or their data fields. Data fields are also called input/output connectors and are located at the bottom/top of the modules in a network. Parameter fields are available on the panel or GUI of a module.

In the following, different methods of handling data connections are shown. Parameter connection handling is not discussed in detail here; for more information about parameter connections, see “Parameter Connection for Synchronization”.

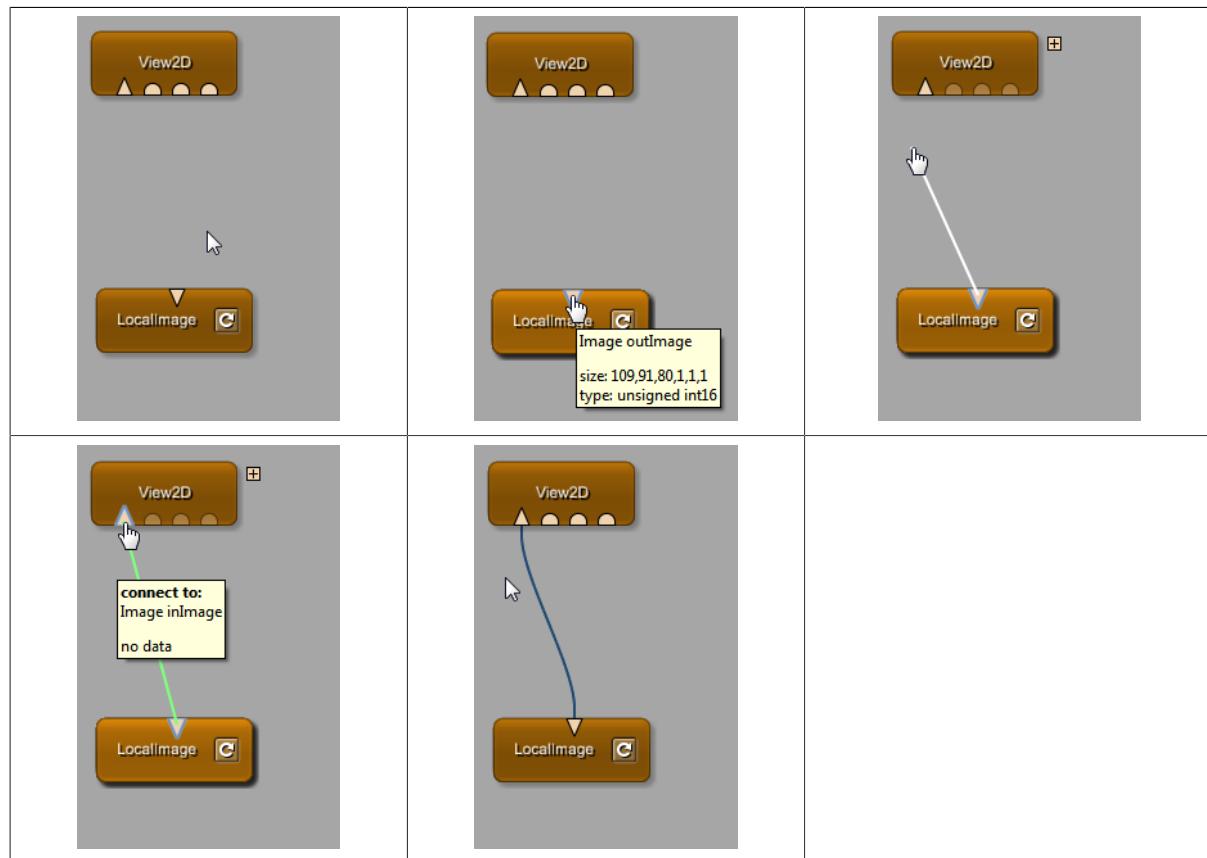
3.4.1. Connecting Modules

Data connections can be established by dragging/drawing, by proximity, i.e., moving modules close to each other, and by inserting a module into an already existing connection.

3.4.1.1. Connecting by Dragging

Move the mouse to one of the connectors, click and hold the left mouse button. Drag the mouse with the left mouse button still pressed and move the mouse to the destination connector. While dragging, an intermediate white connection line is drawn. If the connection is possible, the intermediate connection line turns green. On releasing the mouse button, the connection is established.

Table 3.6. Connecting Modules by Dragging



3.4.1.1.1. Connecting to Inventor Groups by Dragging

Connecting to an Inventor group module (e.g., SoSeparator, SoRenderArea, SoGroup) works the same as described in [Section 3.4.1.1, “Connecting by Dragging”](#).

However, Inventor group modules have an unlimited and dynamic number of input connectors and while not dragging to establish a connection, only the connected connectors are visible. On starting to establish a new connection, all Inventor group modules in a network show the otherwise hidden additional input connectors. Those additional input connectors are smaller than the regular connectors and placed to the left and to the right of the regular connectors. On connecting to an additional connector, the connection is established and the additional connector becomes a regular connector, so that for the next connection, even more additional connectors are available.

Table 3.7. Dragging a New Connection Generates New Input Connectors to the Sides of Regular Connectors

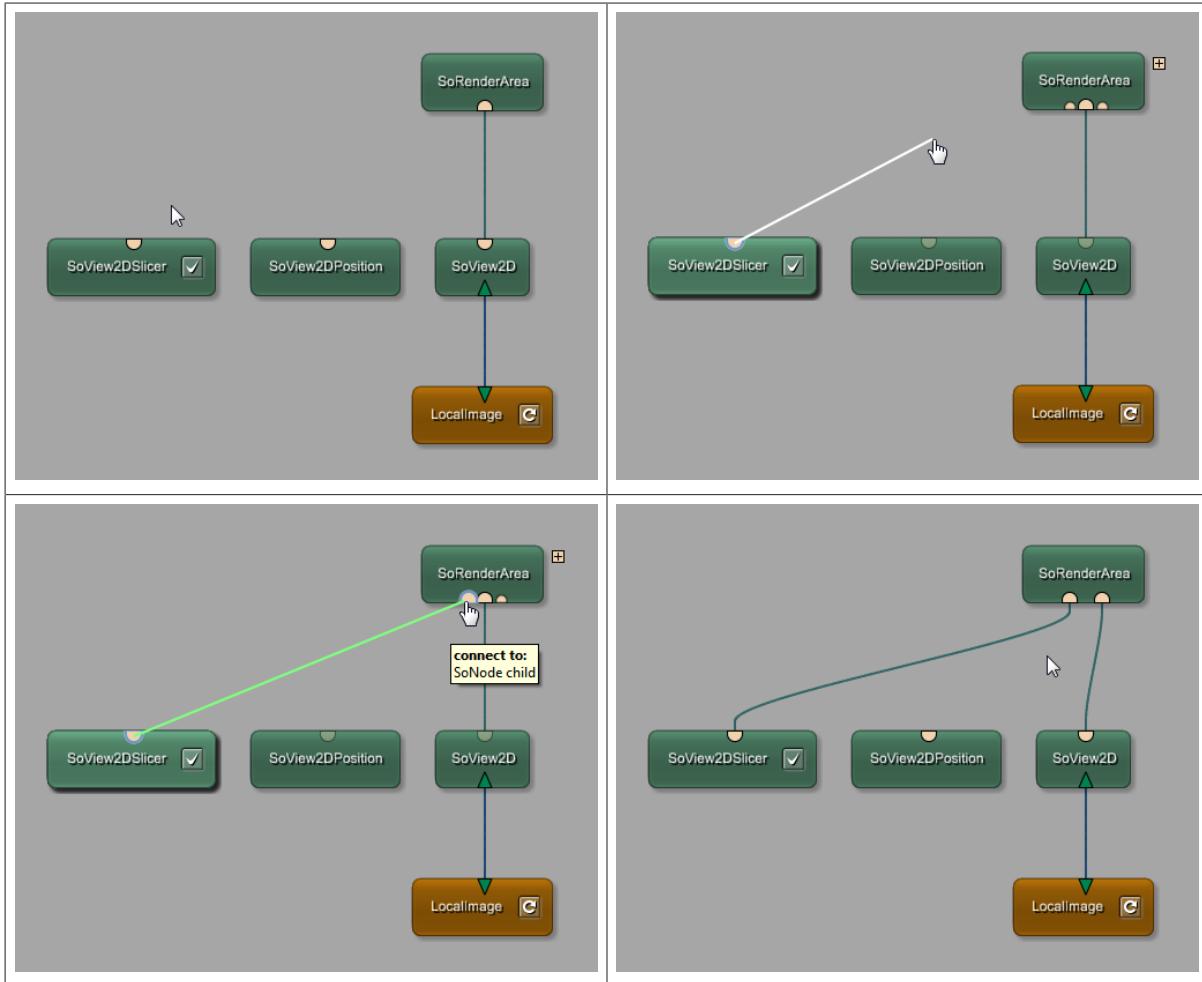
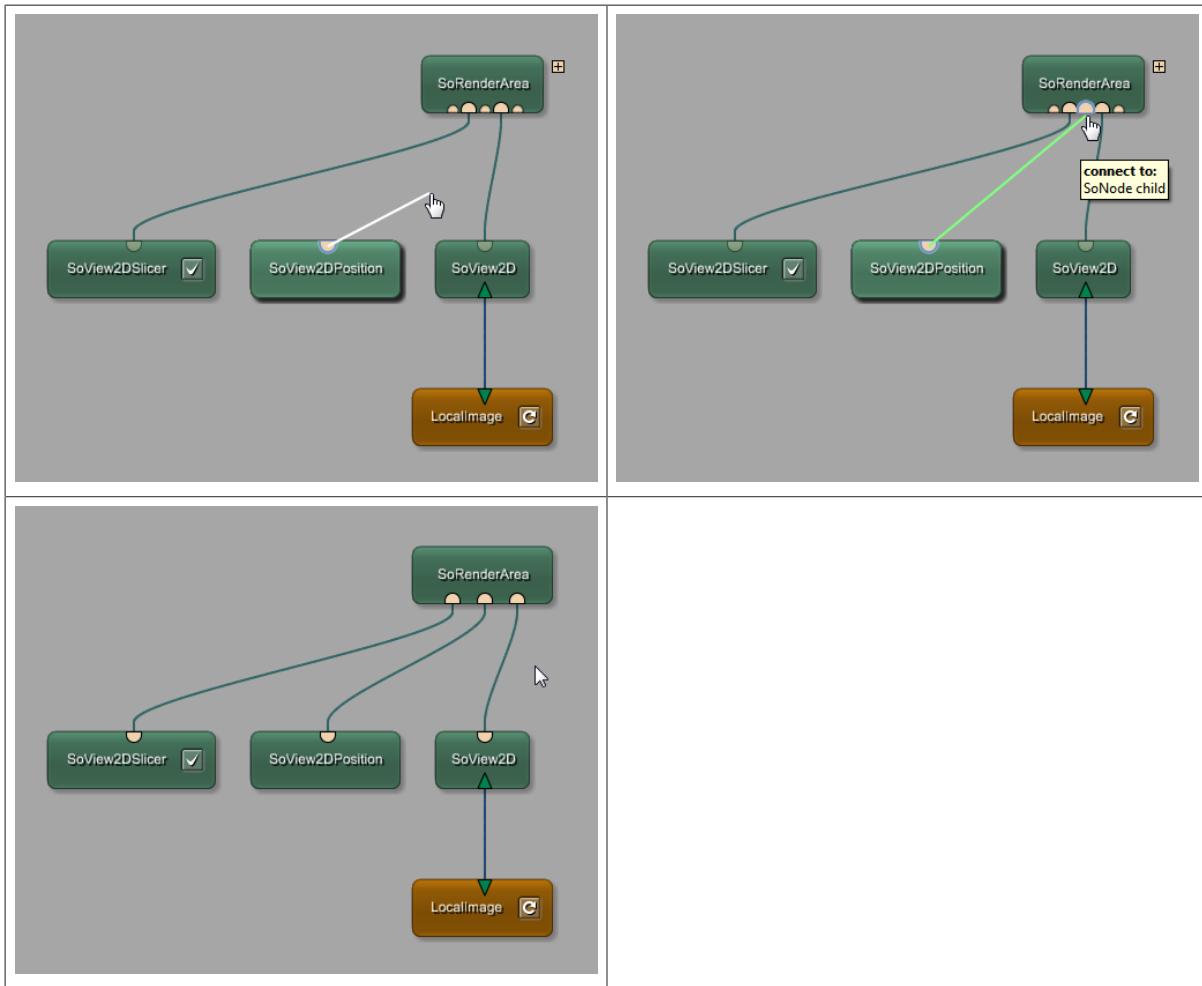
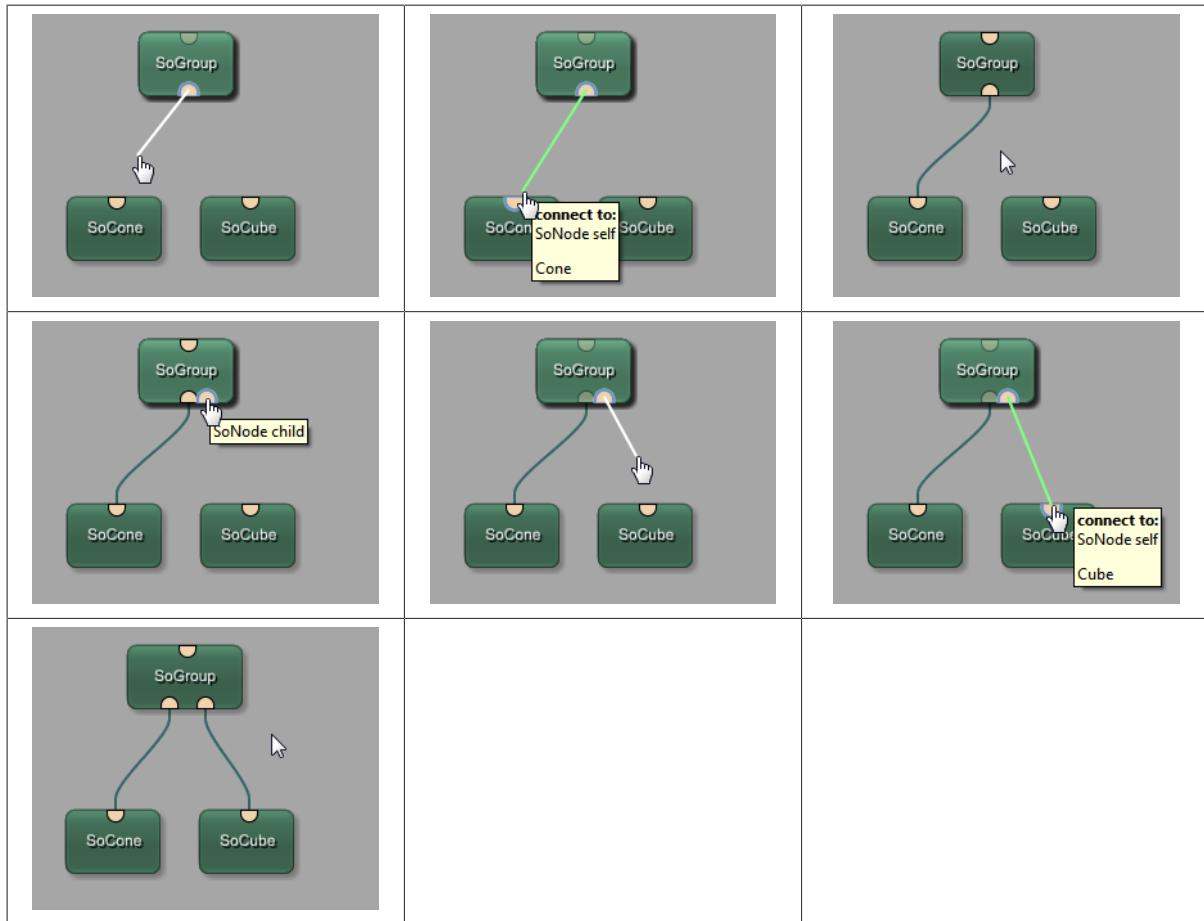


Table 3.8. Even More New Connectors are Available

3.4.1.1.2. Connecting from Inventor Groups by Dragging

Connecting from an Inventor group module is basically the same as connecting to such a module.

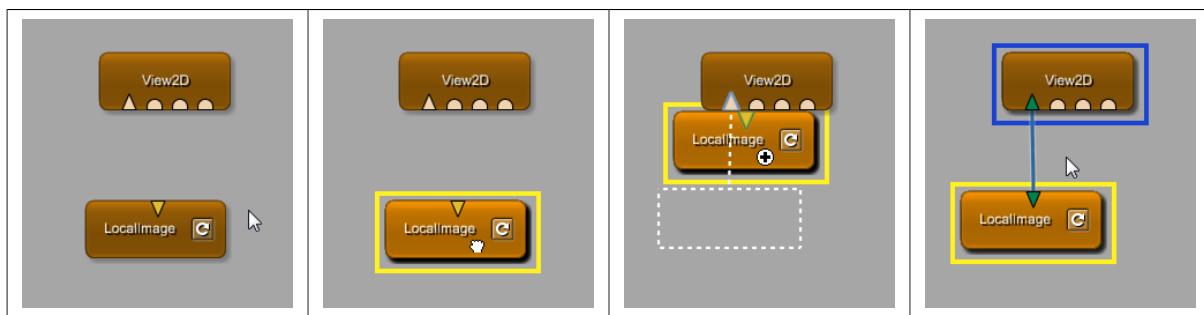
However, new dynamic input connectors appear only if the mouse cursor is placed to the left or to the right of existing regular input connectors. Once an additional connector has appeared under the mouse cursor, it can be used for establishing a connection by clicking on it, holding the left mouse button, and dragging to an output connector.

Table 3.9. New Input Connectors are Generated by Positioning the Mouse

3.4.1.2. Connecting by Proximity

Modules can be connected by moving their input and output connectors close to each other.

Grab a module by clicking and holding the left mouse button. Move the module close to the module with which the connection should be established. If the first free fitting connectors are close enough to each other, a stippled preview of the module to be connected and its connection is rendered. Release the mouse button to establish the connection. The dropped module is automatically positioned at the preview position.

Table 3.10. Connecting by Moving the Source Module into Proximity

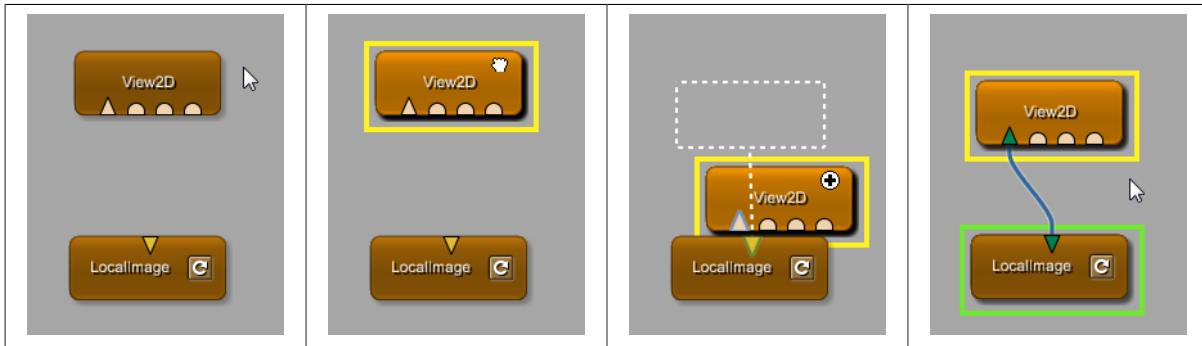
The same procedure works for connecting a destination module.



Note

The stippled preview connection is always rendered in the middle of the preview module's silhouette and not at the actual position of the connector on the module.

Table 3.11. Connecting by Moving the Destination Module into Proximity



3.4.1.2.1. Connecting to Inventor Groups by Proximity

When connecting to an Inventor group by proximity and the group already has at least one input connection, all subsequent connections are generated at the first or the last dynamic connector. A new connection cannot be established by proximity between already existing connections. Use the method described in [Section 3.4.1.1.1, “Connecting to Inventor Groups by Dragging”](#) or [Section 3.4.1.1.2, “Connecting from Inventor Groups by Dragging”](#) for connecting modules in between existing connections.

Table 3.12. Connecting an Inventor Group by Proximity

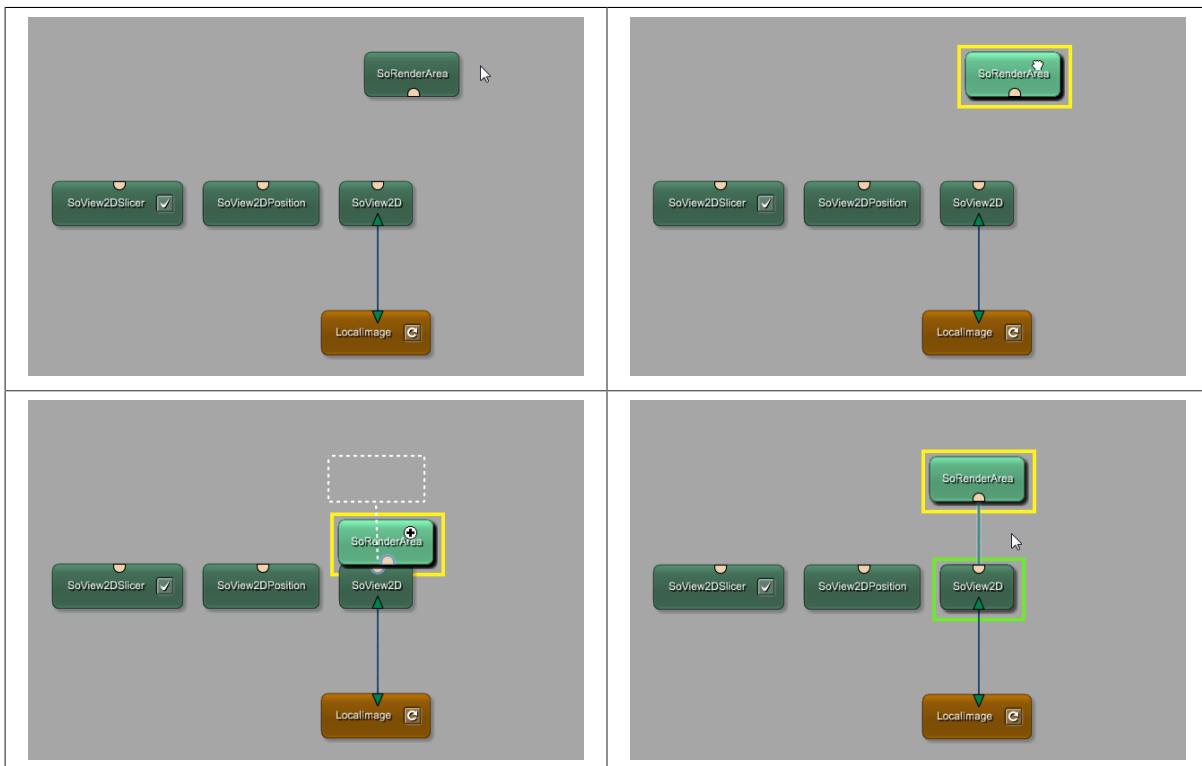
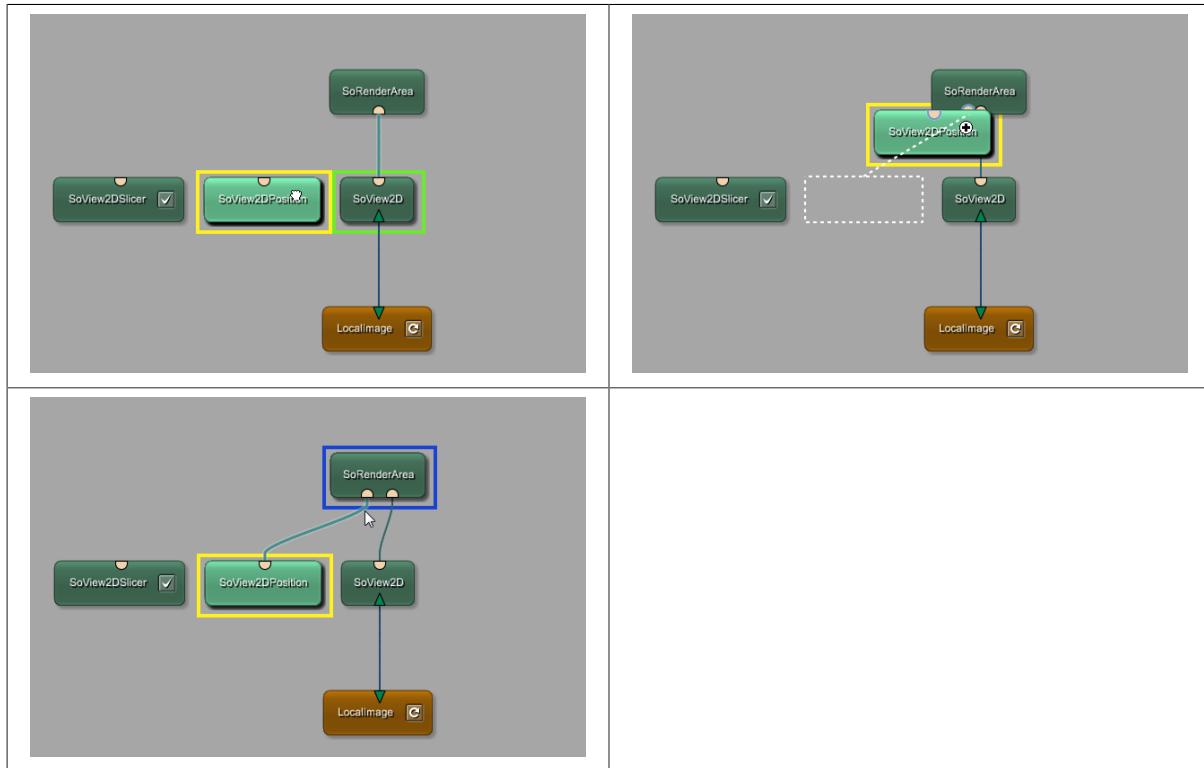
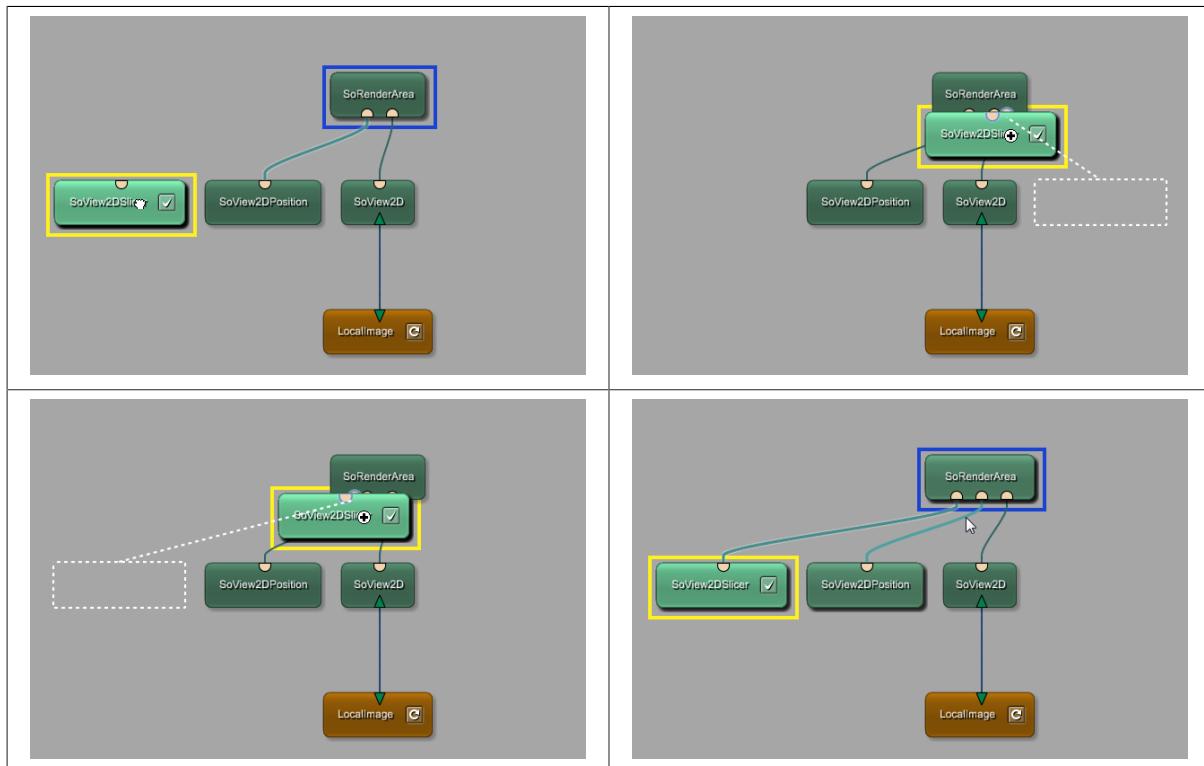


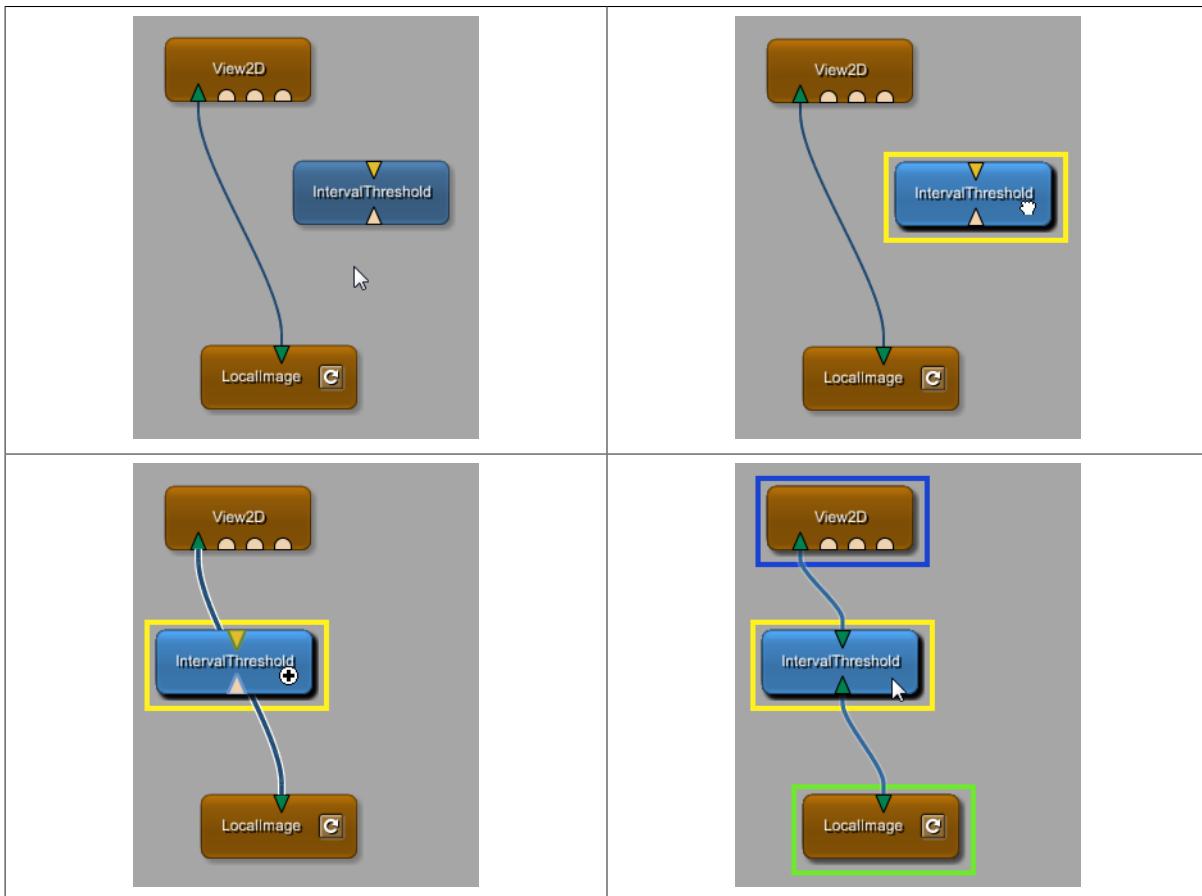
Table 3.13. Connecting to an Inventor Group by Proximity**Table 3.14. Appending vs. Prepending to an Inventor Group by Proximity**

3.4.1.3. Connecting by Inserting into an Existing Connection

To insert a module into an existing connection, start by grabbing the module. This is done by left-clicking it and holding the mouse button while moving the mouse. Move the module over an existing connection.

If the module can be inserted into the connection, the connection is highlighted and the mouse cursor changes to a plus sign. On releasing the mouse button, the module is inserted into the connection.

Table 3.15. Connecting a Module by Inserting



3.4.1.4. Inserting a Module with More than One Input Connector

If a module has more than one input connector and it is dropped onto an existing connection, the left-most free input connector is used for establishing the connection.



Note

In the following two examples, all described methods of establishing connections are mixed.

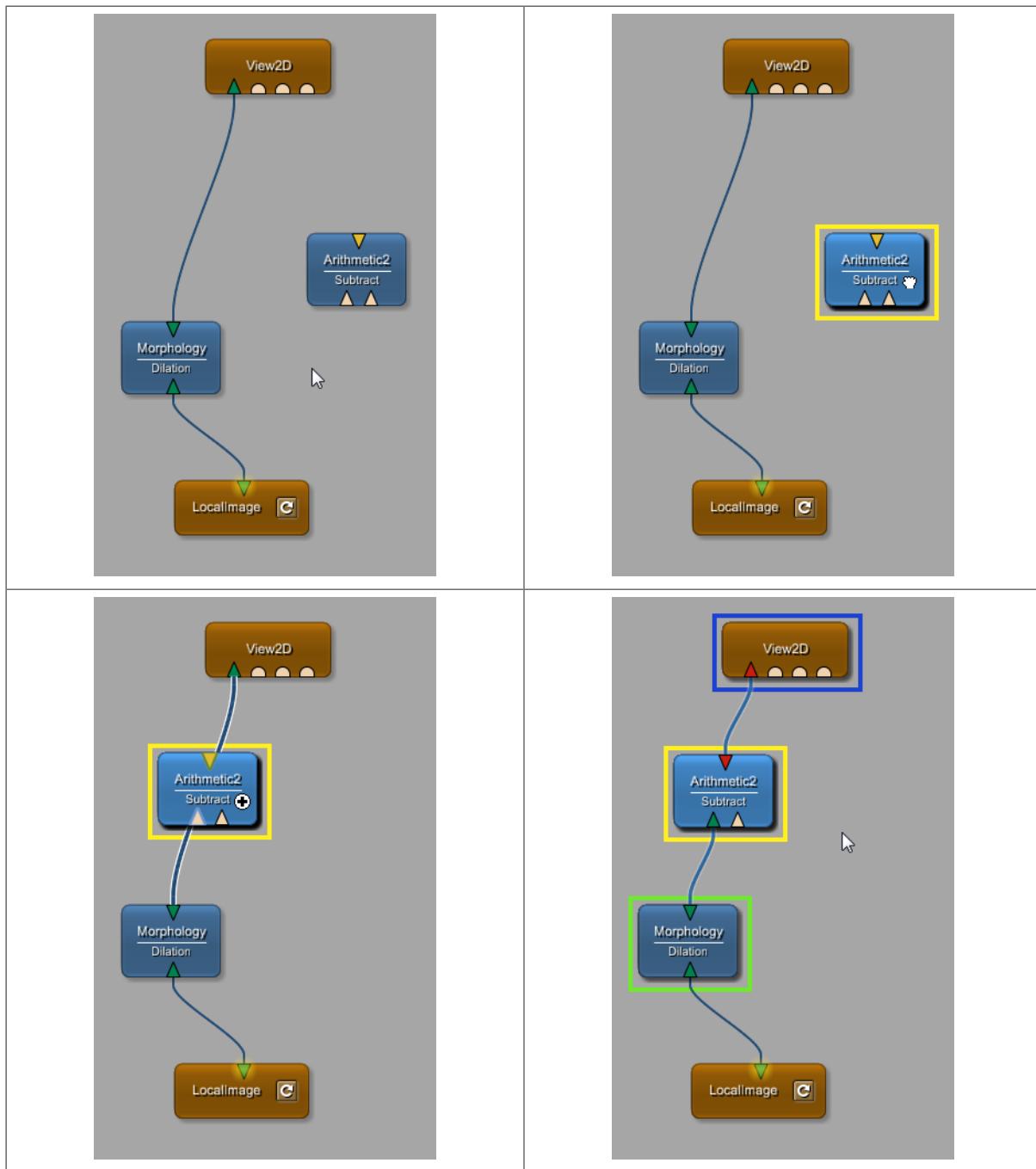
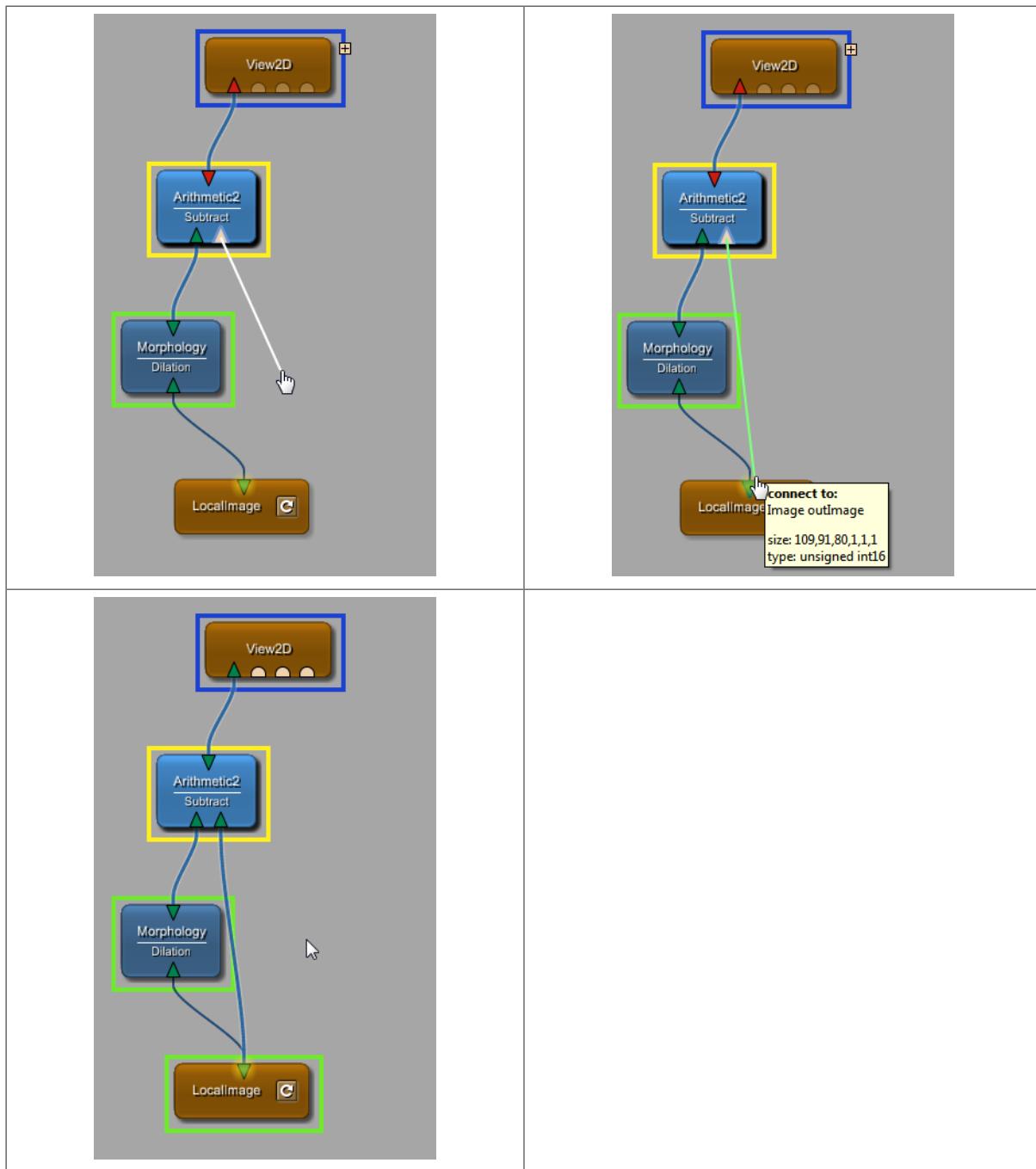
Table 3.16. Connecting a Module with Two Inputs by Inserting

Table 3.17. The Second Input is Connected by Dragging

3.4.1.5. Variation of Inserting a Module with More than One Input Connector

If the left-most input connector is already connected, the first left-most free input connector might just be the second input connector.

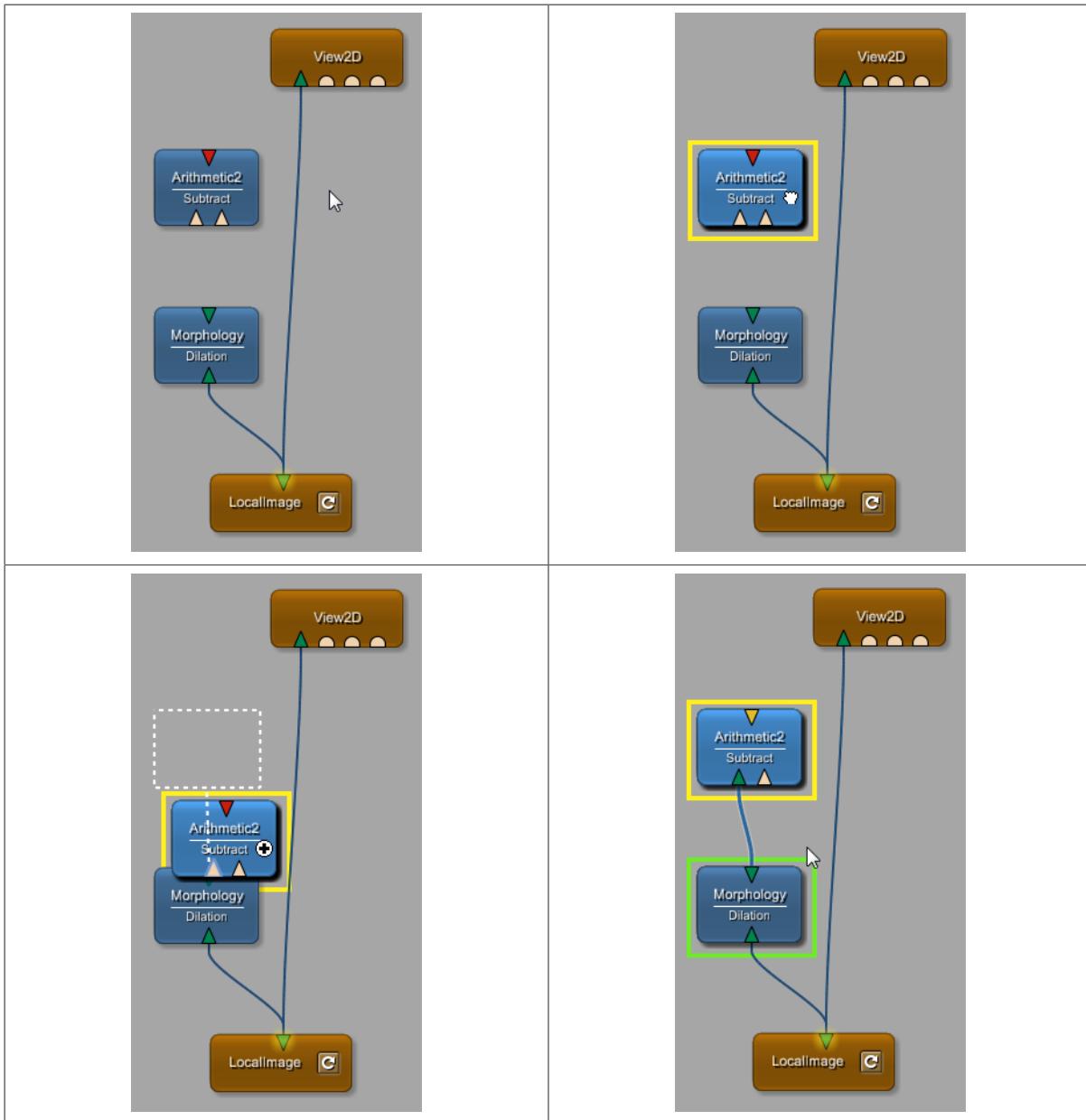
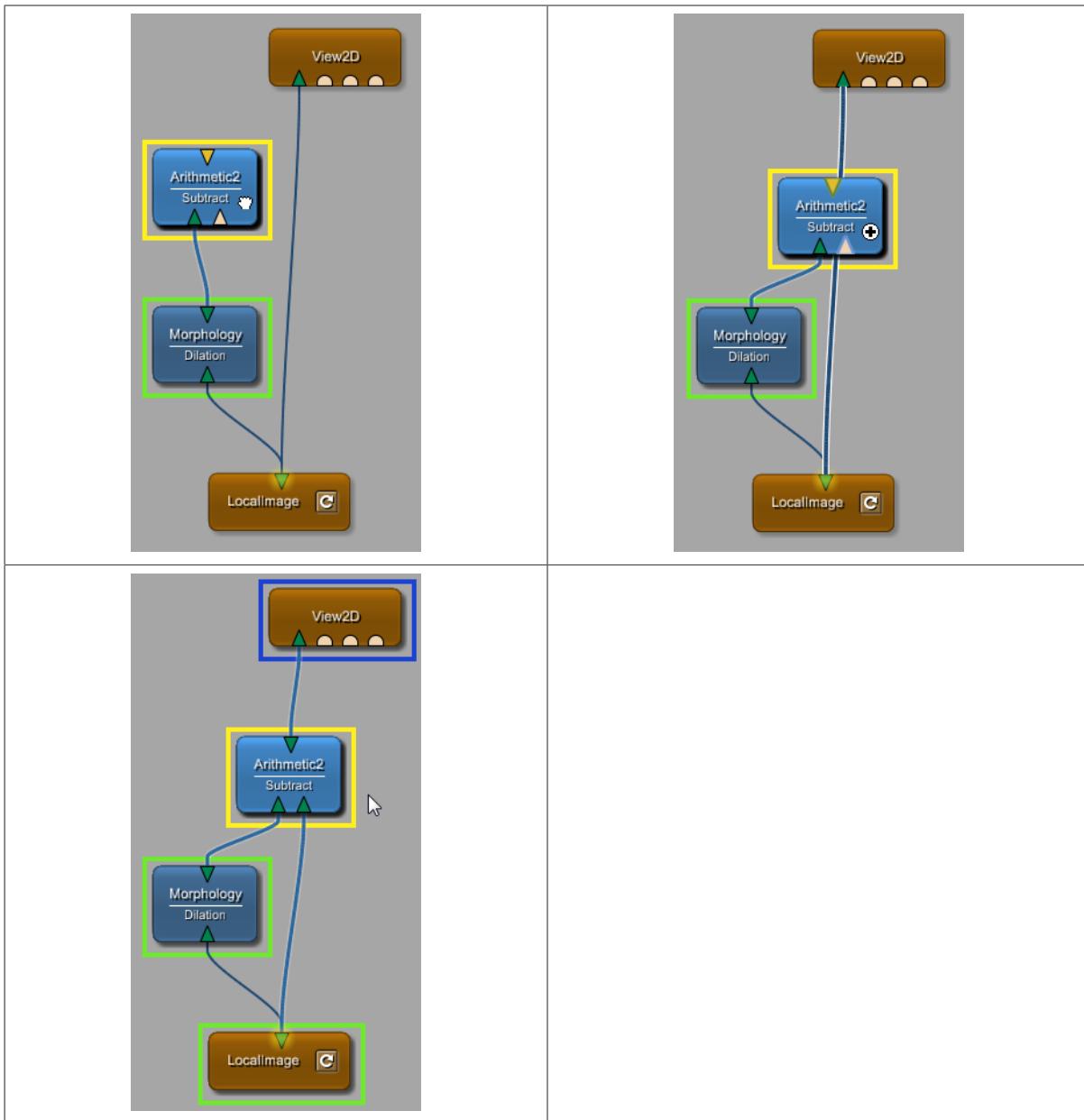
Table 3.18. Variation: First Input is Connected by Proximity

Table 3.19. Variation: Second Input is Connected by Inserting into an Existing Connection



3.4.2. Disconnecting Modules

There are also multiple ways of disconnecting modules.

An input connection or a number of output connections can be removed by dragging a new connection to the network's background. Connections are selectable and any selected network item can be removed by pressing DEL. Connections have a context menu which offers to disconnect that connection, or in case of a bundled module group connection, all of the connections.

3.4.2.1. Disconnecting by Dragging to the Background

A single input connection can be removed by starting to drag a new connection from an already connected input connector and then releasing the drag over an empty region of the network.

If the drag is started on an output connector with multiple connections, and is released over the network's background, all output connections are removed.

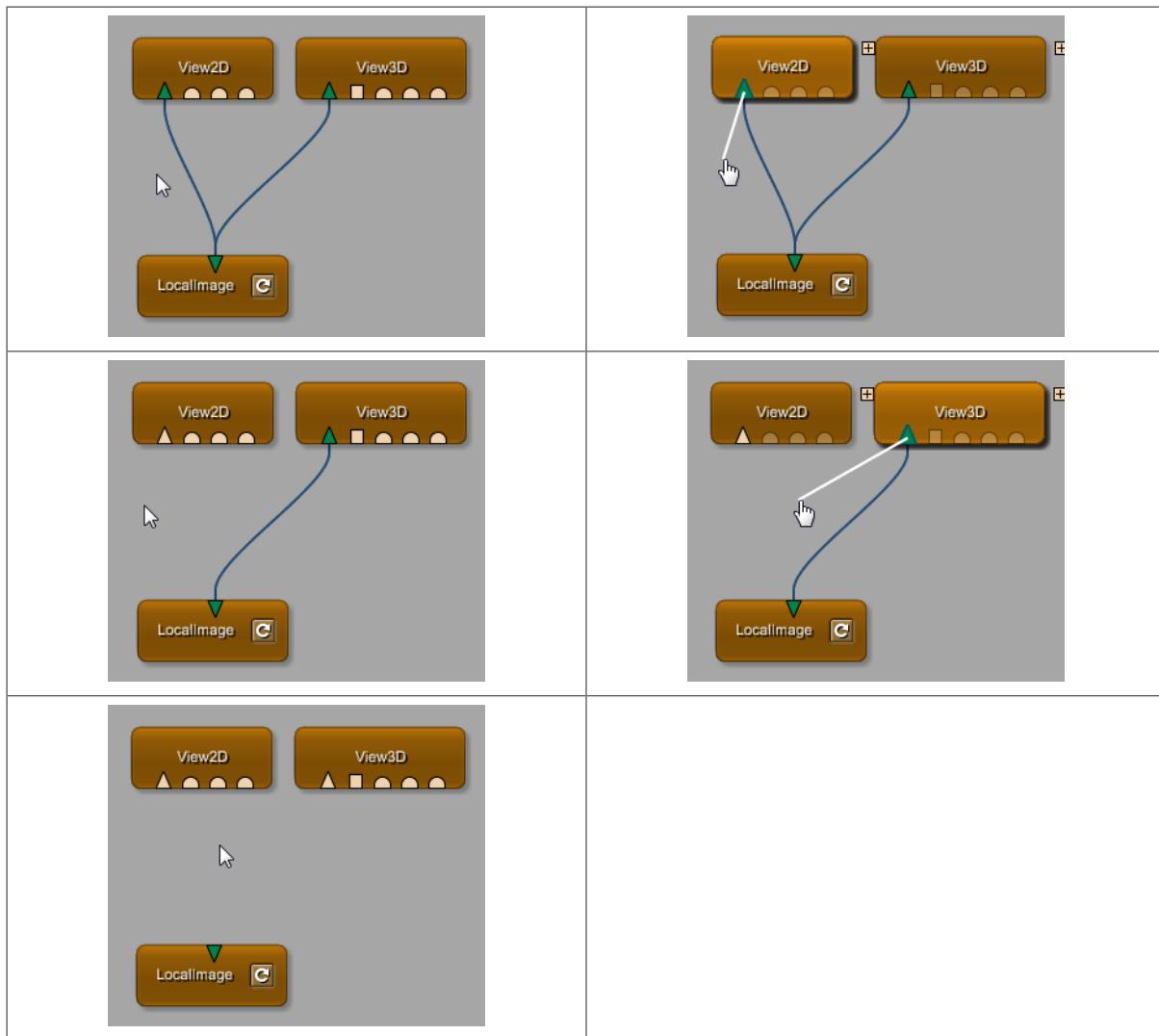
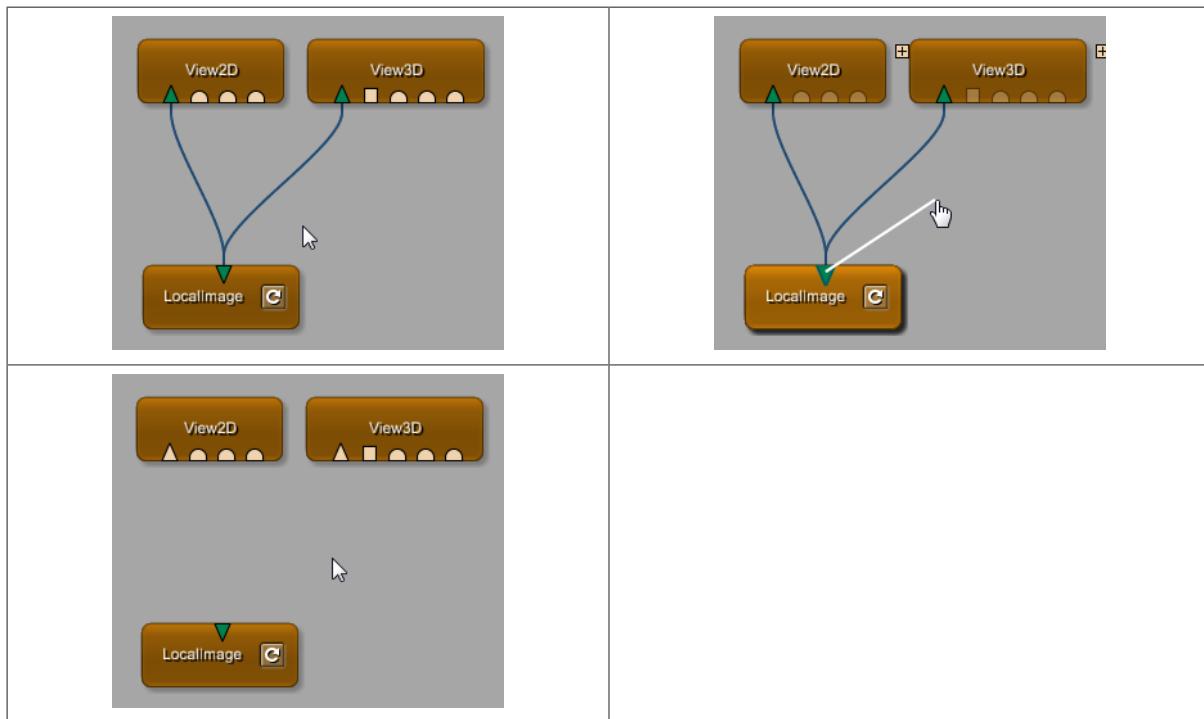
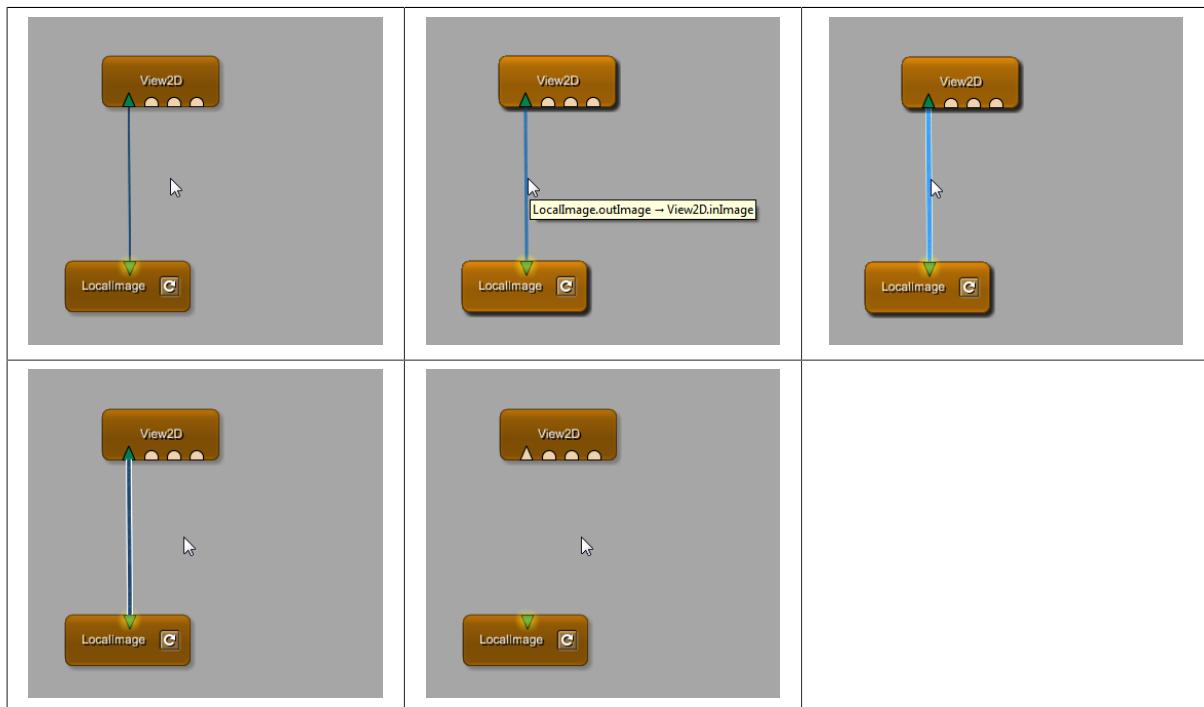
Table 3.20. Disconnecting by Dragging to Background: Input

Table 3.21. Disconnecting by Dragging to Background: Output

3.4.2.2. Disconnecting by Selection

Connections can be selected by clicking them with the mouse.

A selected connection has its own highlighting and is removable by pressing DEL.

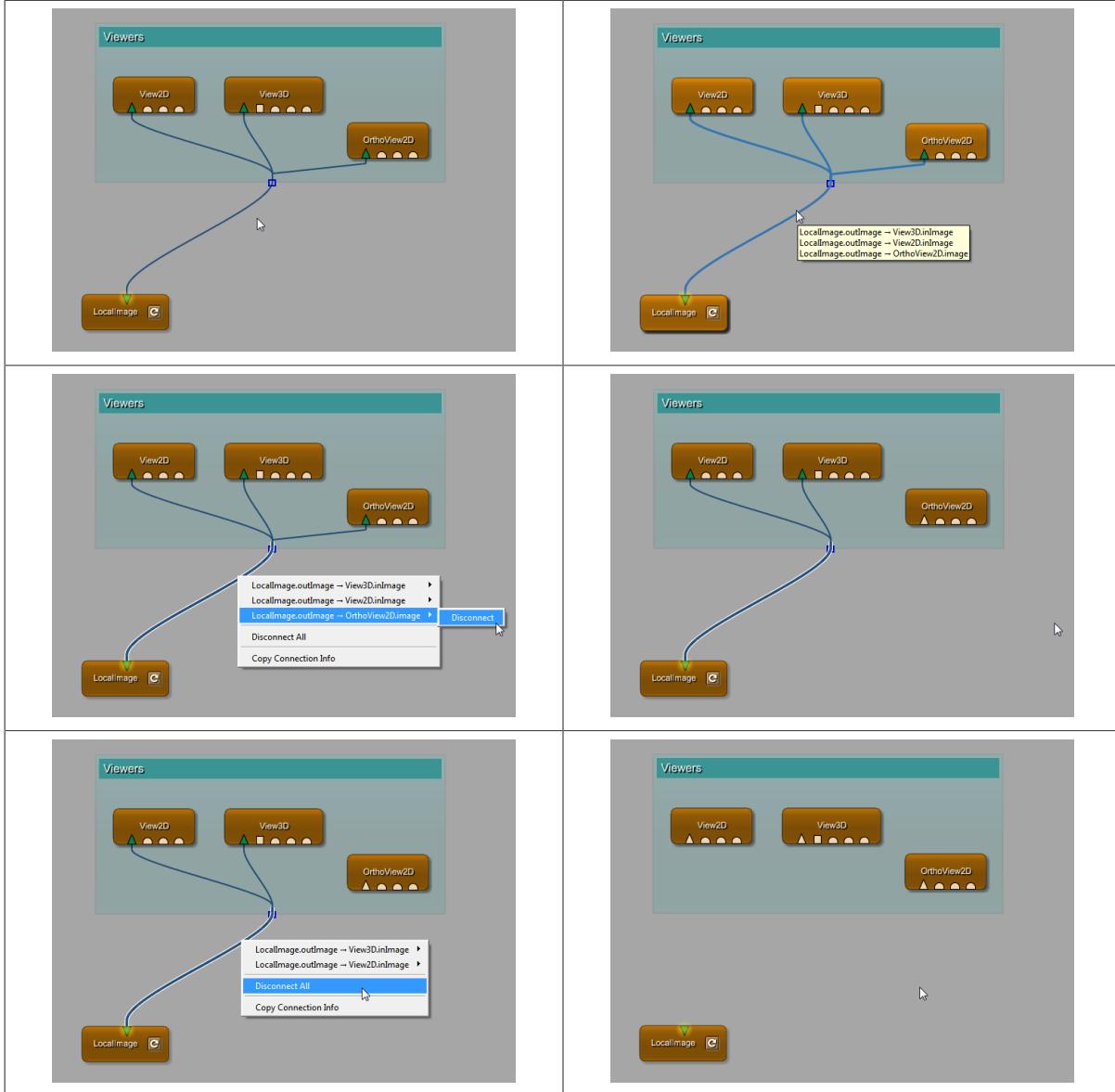
Table 3.22. Disconnecting by Selection and Pressing DEL

3.4.2.3. Disconnecting by Context Menu

Connections not only have a tooltip showing the source and destination connector, but also a context menu where single connections or all connections, if they are bundled, can be disconnected.

The next example features a module group (see [Section 3.11, “Using Groups”](#)) to show that disconnecting by context menu also works for connection bundles.

Table 3.23. Disconnecting by Context Menu

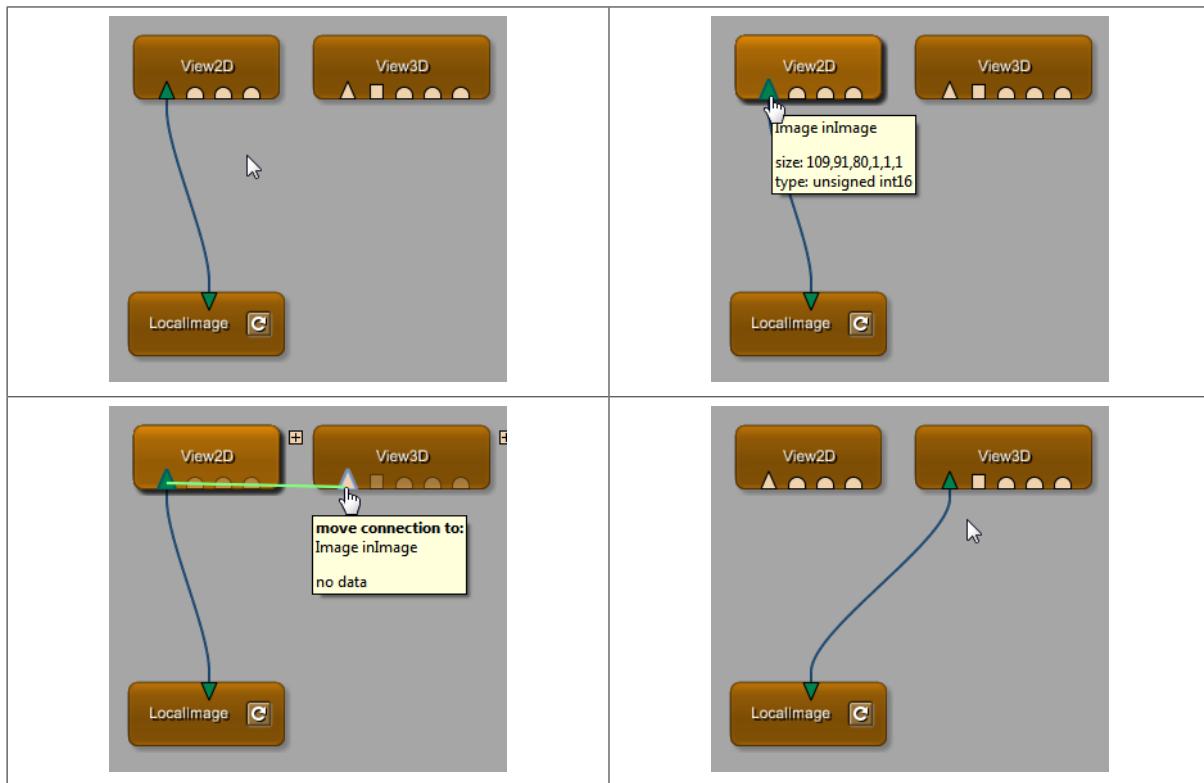


3.4.3. Moving Connections

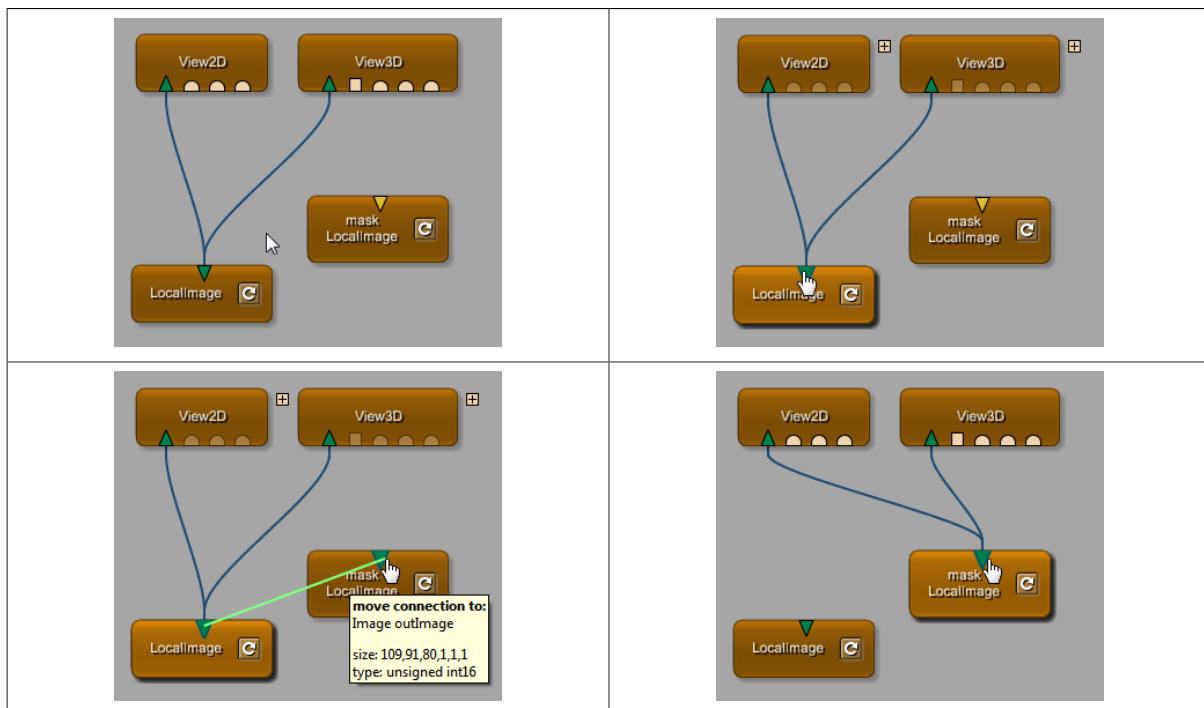
Connections can be moved from input connector to input connector, or from output connector to output connector.

The connector can be on different modules or on the same module.

To move a connection, drag it and move it to the destination connector. If the connection is possible, the intermediate connection is rendered in green. On dropping the connection, the connection to the new connector is established.

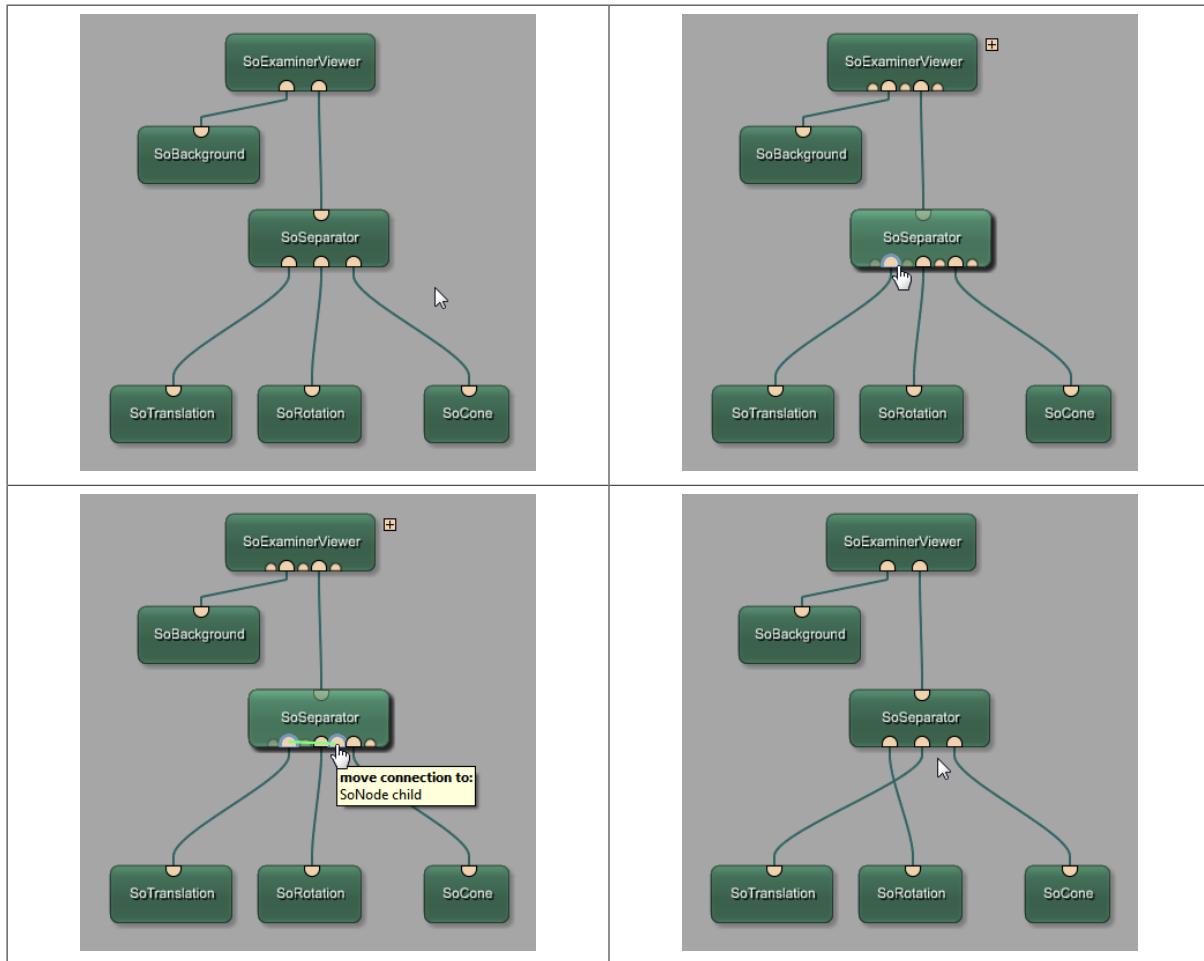
Table 3.24. Move Input Connection

If the moving of connections takes place at an output connector, multiple connections can be moved in a single interaction.

Table 3.25. Move Multiple Output Connections

3.4.3.1. Moving Connections Within Inventor Groups

A connection can also be moved within an Inventor group. On starting the drag, additional connectors are shown to which the connection can be moved.

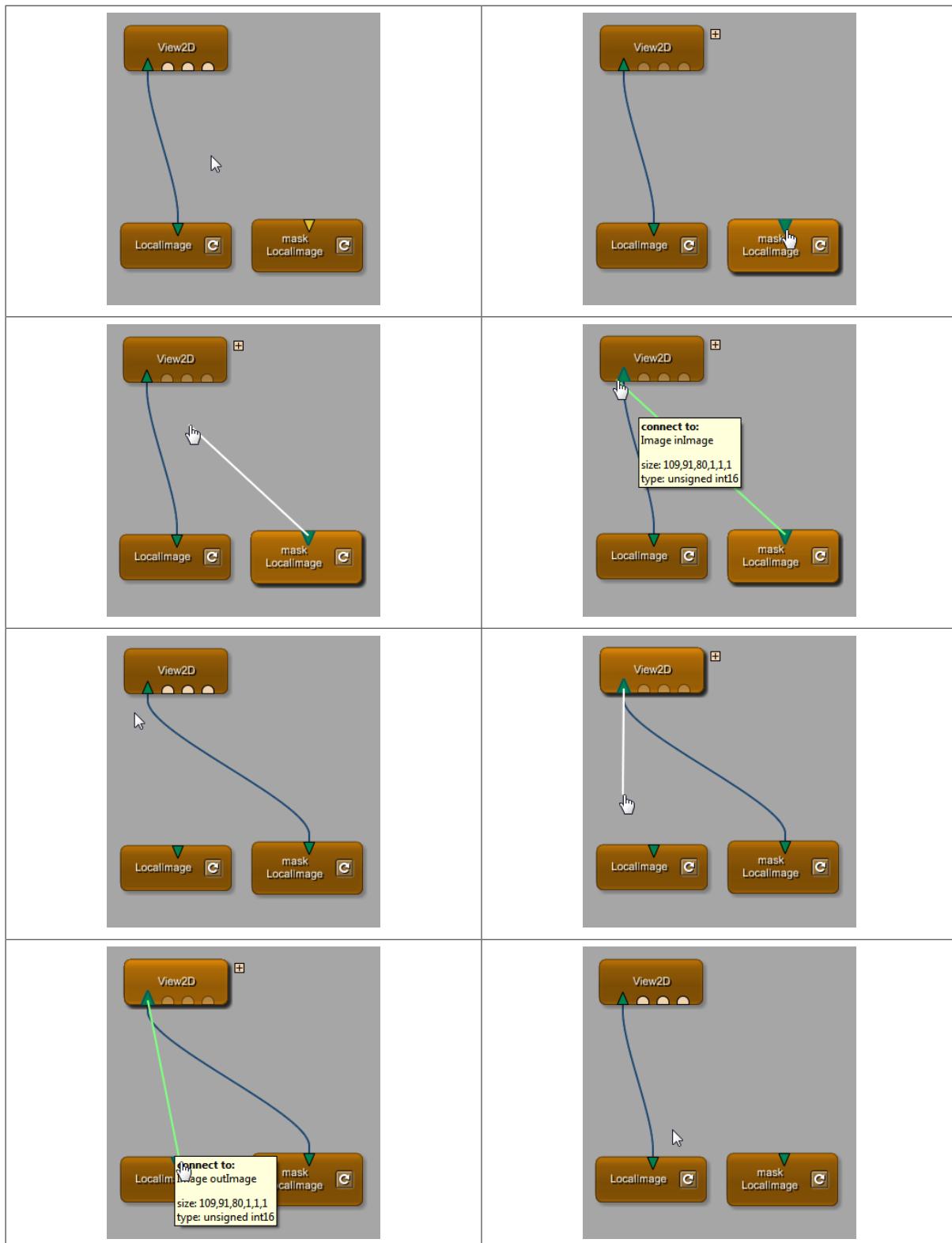
Table 3.26. Move Connection Within an Inventor Group

3.4.4. Replacing Connections

An input connector can only be connected with a single output connector.

When connecting some other output connector to an already connected input connector, the previous connection of that input connector being replaced.

Similarly, if a new connection is dragged from an already connected input connector, the previous connection is replaced by the new connection.

Table 3.27. Replace Connection

3.5. Mouse Pointers

Depending on the action, mouse pointers may look differently.

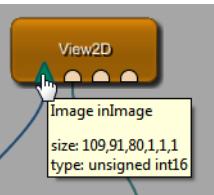
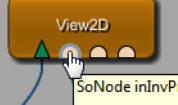
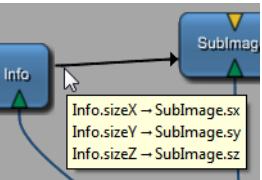
Table 3.28. Mouse Pointers

Action	Pointer
Standard look or when dragging Views or Panels	
When drawing a data connection	
When trying to draw a forbidden connection	
When drawing a selection rectangle	
When dragging a module or network	
When drawing an internal parameter connection	
When drawing a parameter connection between two modules	
When inserting a module into an existing connection or into a module group	

3.6. Mouseover Information

When moving the mouse over modules, connectors and connections, the elements are first highlighted, then context-sensitive information is displayed.

Table 3.29. Mouseover Information

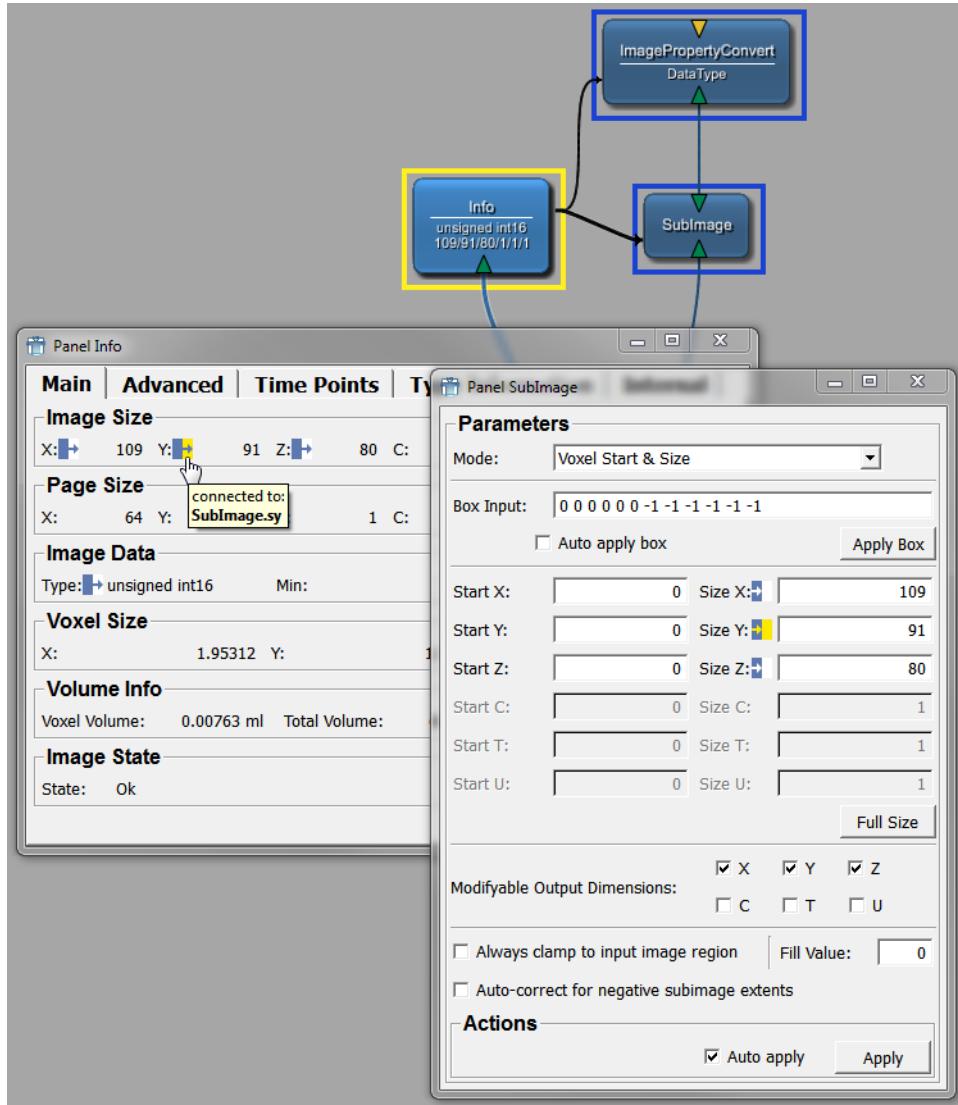
Mouseover	Displayed Information
Module	 <p>LocalImage (mask)</p> <p>Comment: Used to resolve images relative to network or to the local MeVisLab installation</p> <p>Package: MeVisLab/Standard Author(s): Florian Link Genre: File Keywords: load SeeAlso: MakeName LocalSoFile</p>
Image connector	 <p>View2D</p> <p>Image inImage size: 109,91,80,1,1,1 type: unsigned int16</p>
Open Inventor connector	 <p>View2D</p> <p>SoNode inInvPreLUT SoView2DOOverlay</p> <p>Before LUT and viewer inventor input, e.g. for overlays independent of LUT settings</p>
Base connector	 <p>CSOLListContainer 3 / 0</p> <p>MLBase inCSOLList CSOLList Allowed: CSOLList</p>
Data connection	 <p>View2D</p> <p>SoView2DOOverlay.self → View2D.inInvPreLUT</p>
Parameter connection	 <p>Info → SubImage</p> <p>Info.sizeX → SubImage.sx Info.sizeY → SubImage.sy Info.sizeZ → SubImage.sz</p>

Parameter connections also offer the following features when the panels are visible:

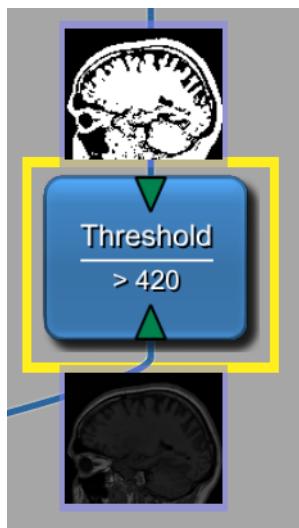
- A mouseover over the parameter connection(s) in the network highlights all connected input/output fields of the parameter connection(s).
- A mouseover over an input/output field of a parameter connection on a panel highlights the other connected field and the parameter connection in the network (thicker line), see [Figure 3.7, “Parameter Connection — Panel Mouseover”](#).

- When moving a parameter connection from one field to another by pressing **SHIFT** while dragging the parameter connection, all connected connections are updated accordingly.

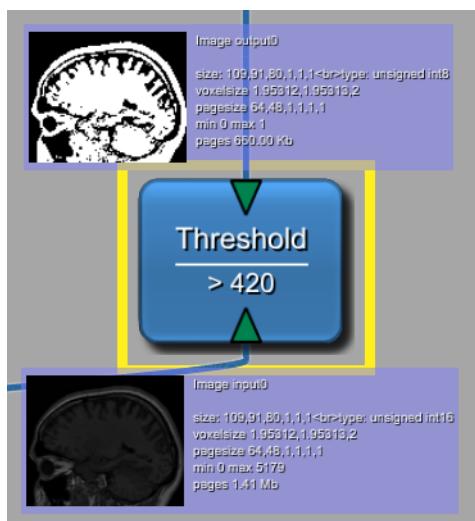
Figure 3.7. Parameter Connection — Panel Mouseover



Additional detailed information on image connectors is available if enabled in **Preferences → Network**, see [Section 4.3.7, “Preferences — Network Appearance”](#).

Figure 3.8. Connector Image Preview**Tip**

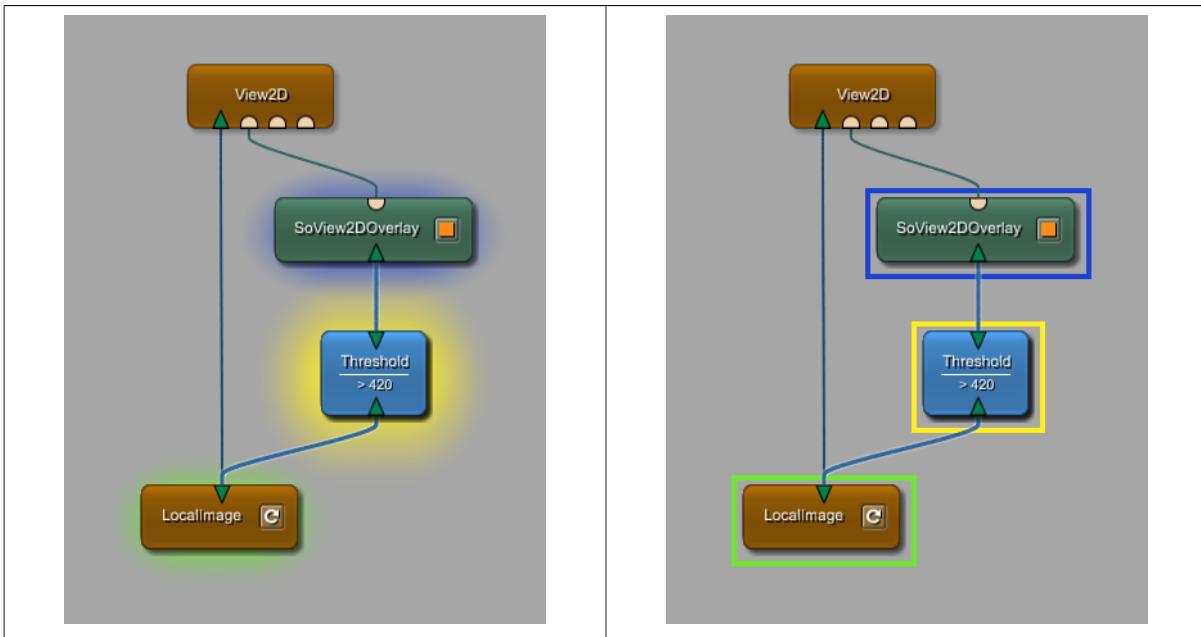
In case of an ML image, the image preview is also slicable. For this, click the image preview and keep the mouse button pressed while moving the mouse up and down.

Figure 3.9. Connector Detail Info and Image Preview**Note**

The amount/depth of visible information depends on the zoom level. An example for this is in the Getting Started, figure “Connector Details Depending on Zoom”.

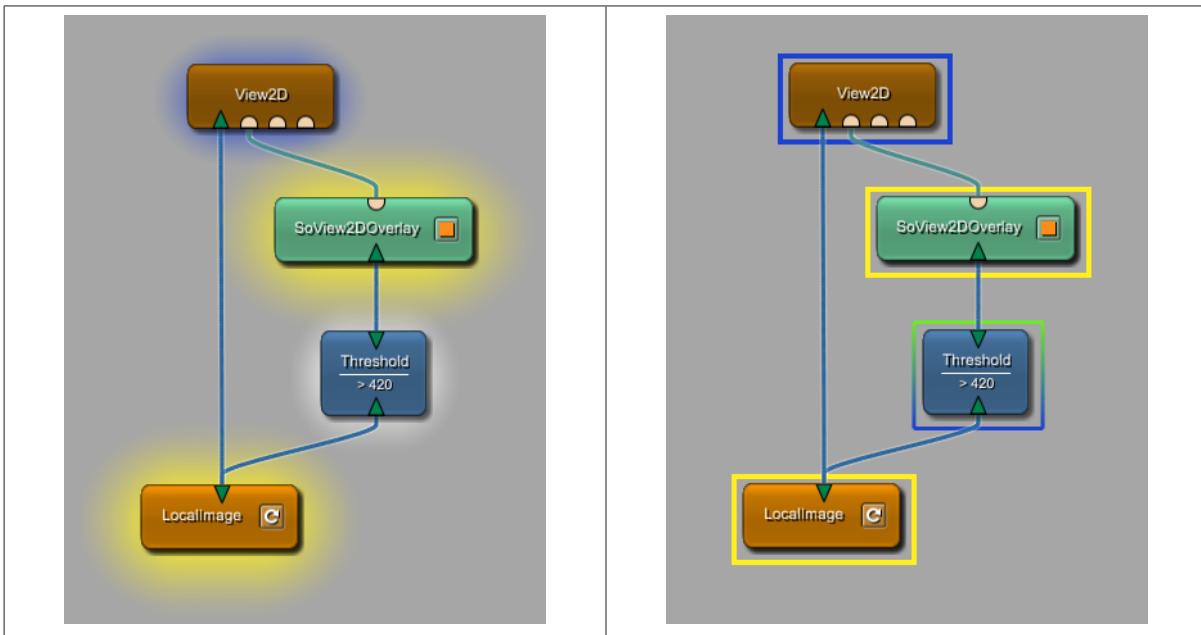
3.7. Module Halo

Selected modules come with what is called a halo effect. Two types are available: classic and alternative. The halo type is set in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).

Table 3.30. Module Halos — Classic and Alternative

The classic halo shows a weaker halo for attached modules; the alternative halo shows the same strength for all modules. For the selected and the attached input and output modules, different colors can be set in the Preferences to make their roles more visible.

If an attached module is input and output module of selected modules at the same time, the halo color will mix. With classic halos, those modules have a white halo. Alternative halos are rendered in both colors as shown in the right figure.

Table 3.31. Module Halos Input Output — Classic and Alternative

3.8. Module Highlighting

To improve the visibility of connections between modules in a more complex network, a special highlighting mechanism is available: When selecting modules in a network and then pressing SPACE,

the workspace is darkened and only the selection and its directly connected modules are highlighted. Pressing SPACE again toggles back to the normal view. In the following example screenshot, the “lut” module is connected via four parameter connections and two data connections, which cannot be easily seen in the non-highlighted display.

Table 3.32. Highlighting of Selections — Classic Halo

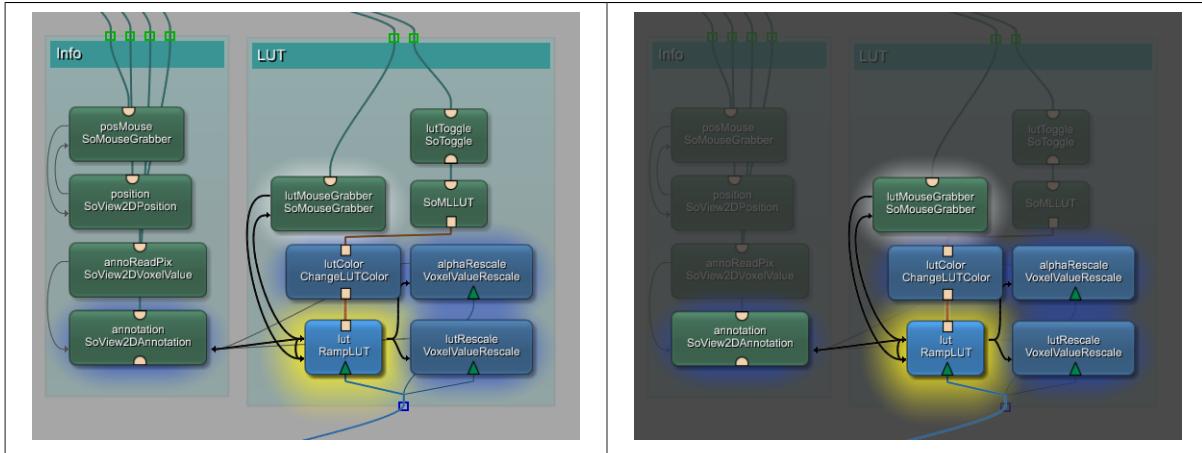
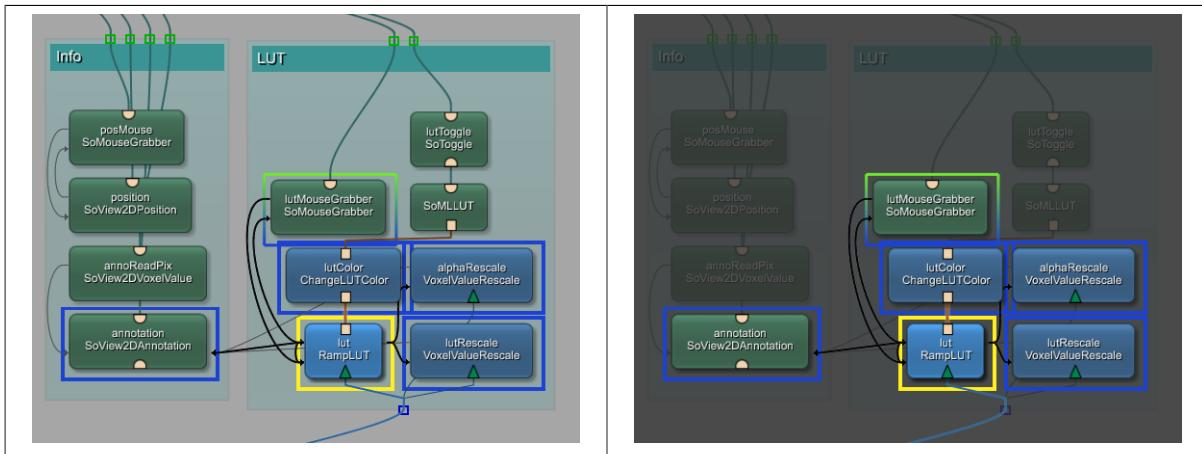
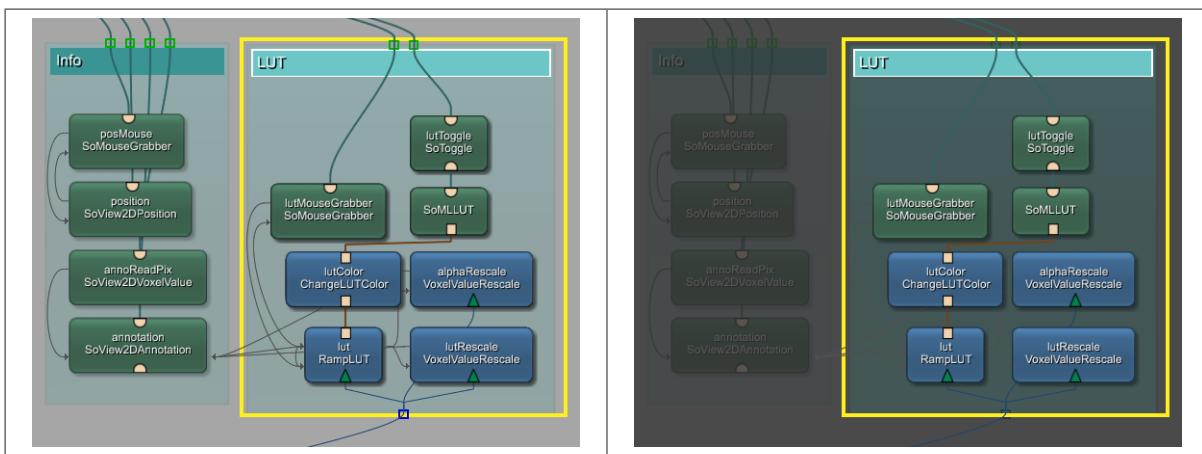


Table 3.33. Highlighting of Selections — Alternative Halo



The alternative halo is also a good way to make notes and groups more visible.

Table 3.34. Module Group with Alternative Halo — Selected and Highlighted

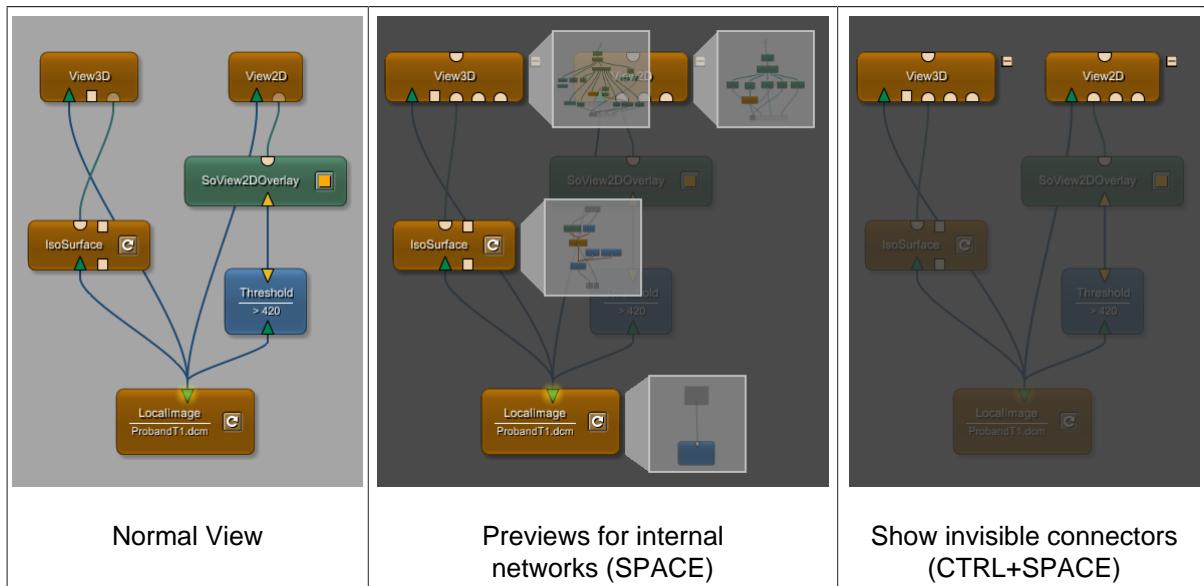


**Tip**

For shortcuts for modules and networks, see [Section 4.3.10, “Preferences — Shortcuts”](#).

When nothing is selected, pressing SPACE will display previews for all internal networks of macros. Clicking on a preview opens the internal network (same functionality as “Show Internal Network” in the context menu or SHIFT + double-clicking the module). Pressing CTRL+SPACE shows invisible connectors only. To toggle back to normal view press SPACE again.

Table 3.35. Preview of Internal Networks of Macro Modules

**Tip**

To remove all selections, press ESC (the network needs to have the focus for this).

No previews are available for script-only macro modules. This might help to find macro modules that have an internal network but should be implemented script-only.

3.9. Module Handling

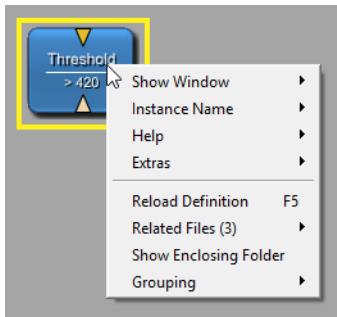
3.9.1. Module Context Menu

Right-click modules to open the module context menu. Its contents slightly depend on the module type. Available groups of entries:

- [Section 3.9.1.1, “Show Window”](#)
- [Section 3.9.1.2, “Instance Name”](#)
- [Section 3.9.1.3, “Help”](#)
- [Section 3.9.1.4, “Extras”](#)
- [Section 3.9.1.5, “Reload Definition”](#)
- [Section 3.9.1.6, “Related Files”](#)
- [Section 3.9.1.7, “Show Enclosing Folder”](#)

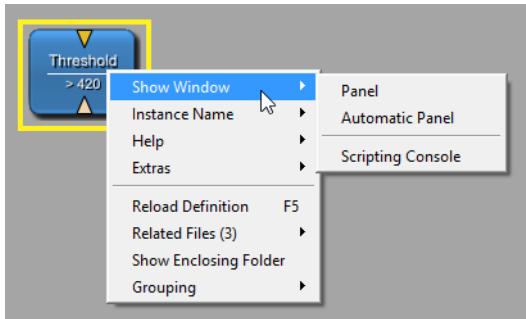
- [Section 3.11, “Using Groups”](#)
- Open Inventor only: [Section 27.8, “Set Open Inventor Override Flag \(Inventor Modules\)”](#)

Figure 3.10. Module Context Menu



3.9.1.1. Show Window

Figure 3.11. Module Context Menu — Show Window



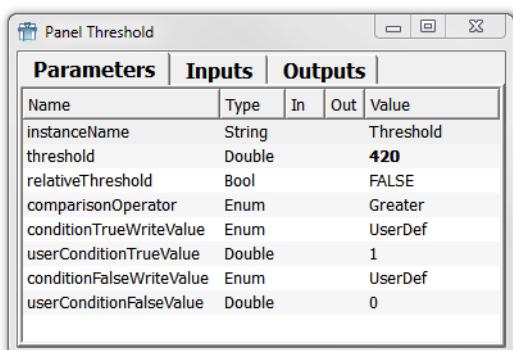
Each module has at least one panel, the automatic panel which is a list of all fields and parameters of the module. Use it for an overview or for editing the parameters (see also [Section 11.1, “Fields”](#)).



Tip

Refer to chapter [Section 4.3.10, “Preferences — Shortcuts”](#) for a shortcut for opening a module's automatic panel.

Figure 3.12. Automatic Panel



The automatic panel lists all fields of the module in order of their initialization in the C++ code or of their definition in the Macro definition. It also shows the data type of the field, whether it is an input or output field, and its current value. The value can be edited directly on the automatic panel.

If a field has a value different from the default value of the field for that module it is displayed in a bold font (but only if the field is editable). This makes it easier to see which parameters have been changed for the module.



Tip

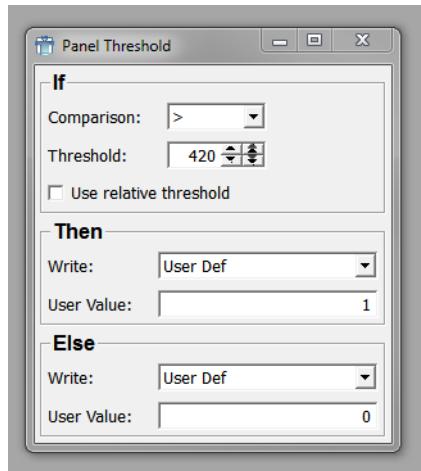
The header of the list of fields of the automatic panel has a context menu where you can toggle how to sort the fields or to turn off the sorting at all. In the later case, the fields are ordered as they are implemented in C++ or in the script.

An option 'Show Attribute Column' turns on an otherwise hidden column where non-persistent and non-editable fields are marked with 'p-' and 'e-', respectively. Values of non-editable fields are also shown with an italic font.

Typically, another type of panel is also available, which shows the parameter fields in a layouted style. It is written in MDL. Important points:

- It is possible to add fields that are not in the C++ code.
- It is possible to add field listeners that can trigger script code.
- It is possible to keep rarely used fields out of the layouted panel. This way, the panel usability might be enhanced. (Fields can always be edited in the automatic panel.)

Figure 3.13. Panel Defined in MDL



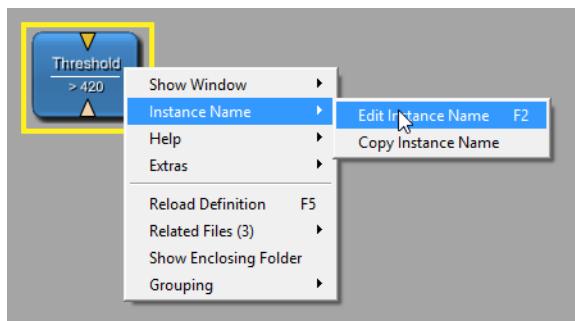
Other windows may be available. For example, for the `View2D` module, a Viewer and a Settings window are available. For information on defining windows, see the MDL Reference, chapter "1.3.2.1. Window". For an example, see the Getting Started, chapter "Adding the Macro Parameters and Panel".

Show Scripting Console

Opens the **Scripting Console** for scripting in the context of the current module, see [Section 4.7.1, "Show Scripting Console"](#).

3.9.1.2. Instance Name

Edit Instance Name

Figure 3.14. Module Context Menu — Edit Instance Name

This option allows distinguishing several instances of the same module. Within a network, each module instance must have a unique name. If no specific instance name is given, the copies of the modules are numbered automatically (1, 2, 3, etc.). Alternatively, the instance can be renamed manually.



Note

Instances of modules have to be unique because modules are addressed by their instance names in scripting.

Select the option or use the respective shortcut (see [Section 4.3.10, “Preferences — Shortcuts”](#)) to open a dialog for entering a new instance name.

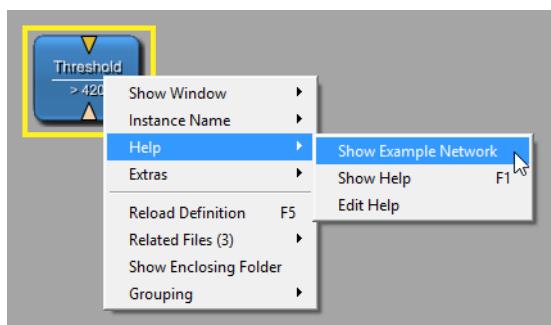
The instance name is displayed above the module name. If the instance name is the module name plus a number, only the instance name is displayed, since it includes the module name.

Figure 3.15. Modules and Instance Names

Copy Instance Name

This option copies the module's instance name to the system's clipboard (paste buffer). This can for example be used to copy the module's name into scripting code.

3.9.1.3. Help

Figure 3.16. Module Context Menu — Show Example Network

Show Example Network

Opens the example network in a new network tab. Only active if an example network exists (otherwise the entry is grayed out).

If a module has multiple example networks, this entry is a menu item, showing the number of example networks. On selecting, a subnetwork offers all available example networks by their names.

Show Help

Displays the HTML help file for the module in the default browser. Only active if a help file exists.

Edit Help

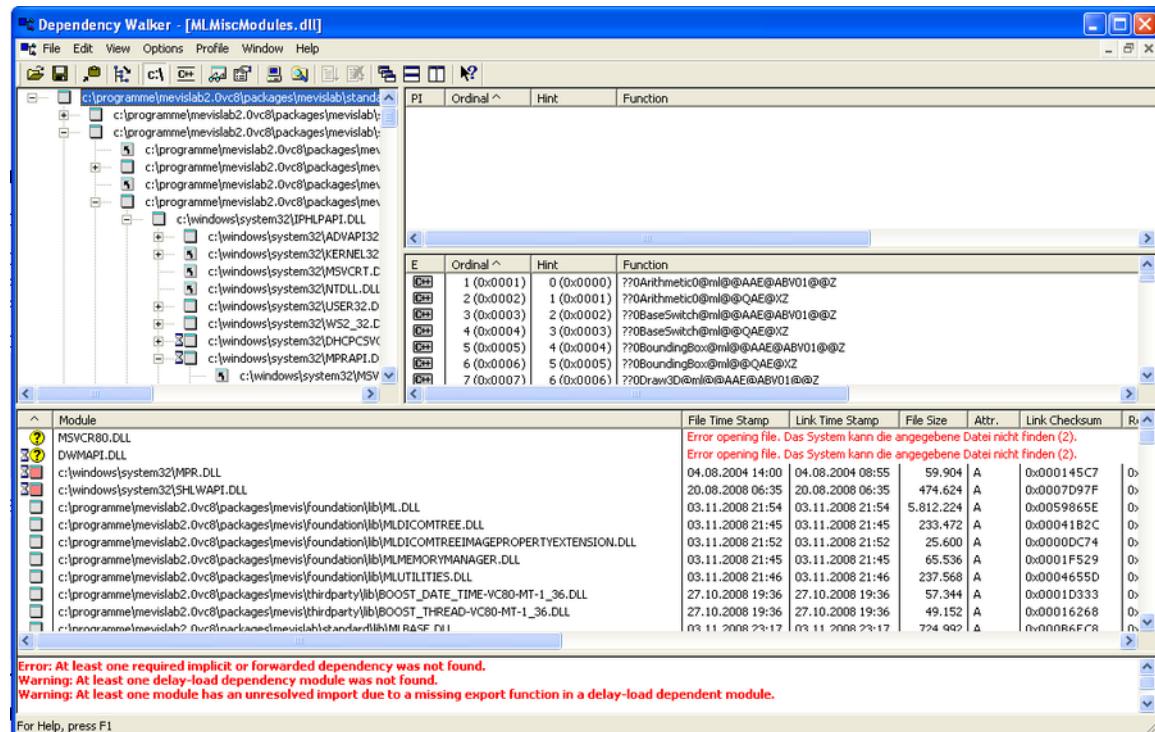
Edits the mhelp file in MATE. Use this option if fields have changed (renamed, new fields, removed fields) or if the module is new. This creates the initial mhelp file if it does not exist or refreshes an existing mhelp file with updated field information.

3.9.1.4. Extras

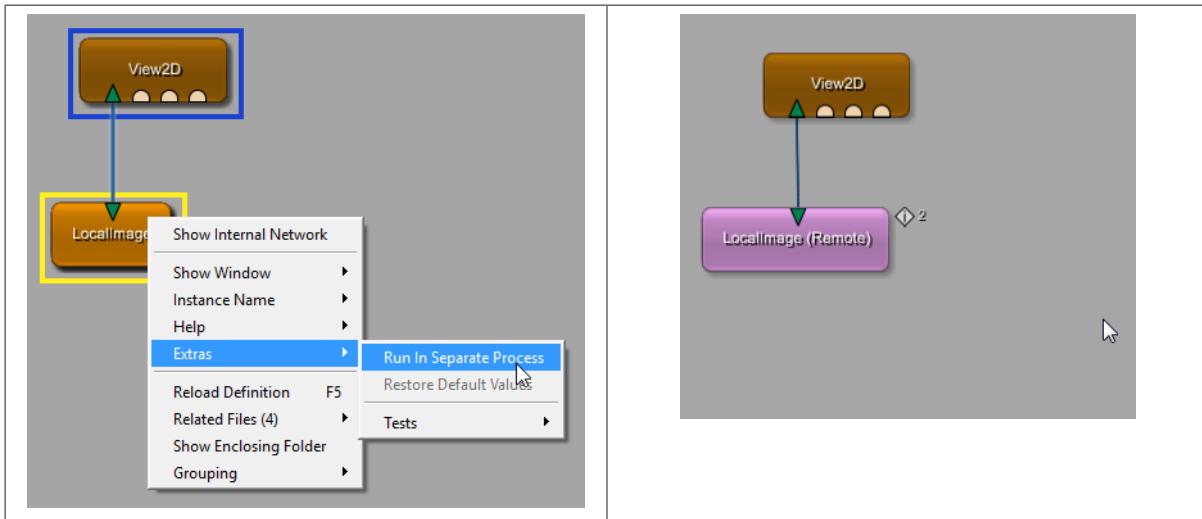
Show DLL Dependency (not on macro modules)

This option uses (on Windows) the Dependency Walker for checking and displaying all dependencies for the module. For more information, please refer to the help of the Dependency Walker. Linux has its own solution for displaying similar information.

Figure 3.17. Dependency Walker



Run In Separate Process (not on Inventor modules)

Table 3.36. Run In Separate Process

MeVisLab allows for running ML and macro modules in background processes, so-called worker processes. We call the underlying concept "Remote Modules". "Run In Separate Process" will replace the selected module in the network by a remote module and start a MeVisLab worker process that loads the replaced module. Field and image changes are transmitted asynchronously between MeVisLab and the worker process.

Remote Modules are an alternative approach to multithreading for utilizing multiple CPUs and asynchronous processing. For example, a remote module can be used to move long running calculations into the background to keep the GUI responsive.



Note

Restrictions:

- You can only move single modules.
- Image inputs are not supported. It is not worth to load an image in the main process and then move the image data to the worker process. Load it directly in the worker process instead; you might need to create a macro module for this.
- Image outputs do not support image extension information. Normally you don't need this.
- Base fields are only supported through special handlers. At the moment there are only handlers for XMarkerList(Container) and some specialized remote Base types like RemoteRendering, RemoteFileTransfer, RemoteCallInterface and AbstractItemModel. Other Base types will just result in an empty Base field.
- Inventor nodes do not work.
- Scripting doesn't work if it accesses the GUI or is called from the GUI (since the GUI lives in another process than the scripting context). Controls may also not use fields of sub-modules.
- Use an ItemModelView instead of a ListView if you need to display lists or tables in your GUI.
- Module fields will not update immediately after some other field was changed, since updates are transmitted asynchronously.

Restore Default Values

Sets all fields whose values differ from their default value to this default value. (This are the fields whose value is displayed in bold font in the automatic panel.)

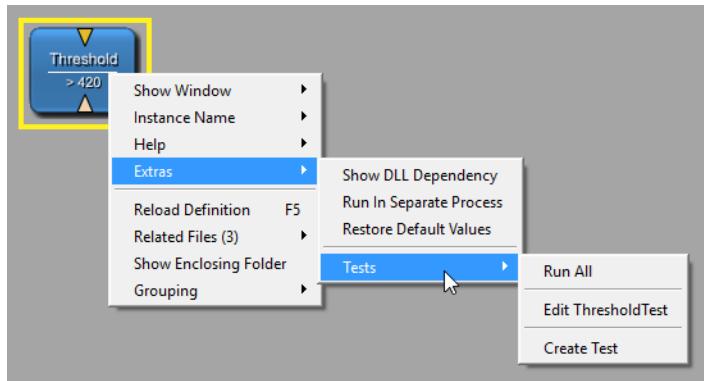
This is easier than creating a new module to replace an existing module with it.

Set Open Inventor Override Flag (only on Inventor modules)

See [Section 27.8, “Set Open Inventor Override Flag \(Inventor Modules\)”](#).

Tests

Figure 3.18. Module Context Menu — Tests



In the Tests submenu, testing options are available.

Run All

Starts all available tests for the module. In case of `Threshold`, the generic test case “Formal” and the “Functional” test case are executed. When the tests are finished, a test report window is opened. (See also [Section 4.6.5, “Run Module Tests...”](#), the TestCenter Reference, and the Getting Started, chapter 16, “Using the TestCenter”).

Edit <AssociatedTest>

Opens the files of a test associated with the selected module.

Create Tests

Opens the TestCaseManager on the tab to create a new functional test.

3.9.1.5. Reload Definition

Reloads the module definition (.script and optional .py file). This is necessary when layouting panels and windows or when working on the scripting.



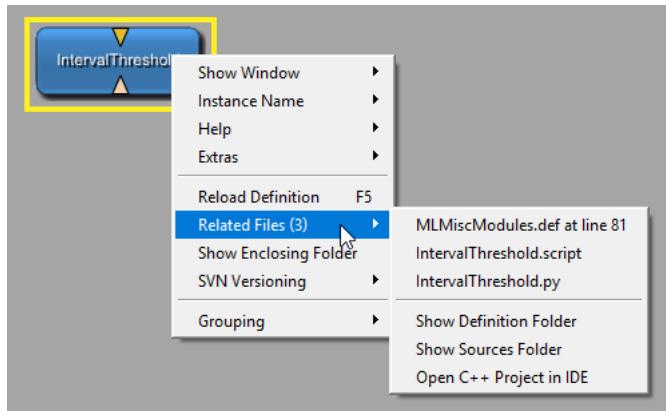
Tip

A single selected module can also be reloaded by pressing the according shortcut key for the OS. Refer to [Section 4.3.10, “Preferences — Shortcuts”](#).

If modules are being reloaded, an animation (module turns white and slowly gains its color back) indicates that the modules' definitions have indeed been reloaded.

3.9.1.6. Related Files

Figure 3.19. Module Context Menu — Related Files



Related Files: Lists all files belonging to the module. Possible file types are `.def/.script` (MDL definition files), and `.py` (scripting files). Select a file to open it in the default editor (as set in [Section 4.3.4, "Preferences — Supportive Programs"](#)).

Show Definition Folder: Opens the definition folder of the module which contains the `.def` and the `.script` file. If the module is augmented by scripting, the `.py` files can also be found there.

Show Source Folder: Opens the folder containing the source code files of the module.

3.9.1.7. Show Enclosing Folder

Shows the directory where the definition file of the module is located.

3.9.1.8. Groups

For the Groups functions, see [Section 3.11, "Using Groups"](#).

3.9.2. Additional Inputs

Modules may have more inputs and outputs than at first visible, to keep the module display as uncluttered as possible. An example for a module with possibly hidden inputs is the `View3D` module. It offers the additional context menu entry `View3D Options → Show Inventor Inputs`. If enabled (which is the default), three Inventor input fields are displayed. The option can also be enabled in the Settings panel.

Figure 3.20. View3D With Visible Inventor Inputs (Default)



Figure 3.21. View3D With Hidden Inventor Inputs



Tip

The three Inventor inputs of `View3D` have certain positions in the scene rendering, i.e. the first input is before LUT and volume renderer, the second between LUT and volume renderer

and the third after LUT and volume renderer. This can be seen when the Inventor inputs are displayed and the internal network is opened, see next paragraph.

Depending on the programming, the number of inputs may be dynamically set. For example, this is the case for the `Switch` module.

For a supporting visualization while interactively drawing connections, see [Section 3.3, “Connector and Connection Types”](#)

3.9.3. Show Internal Network (Macro Modules)

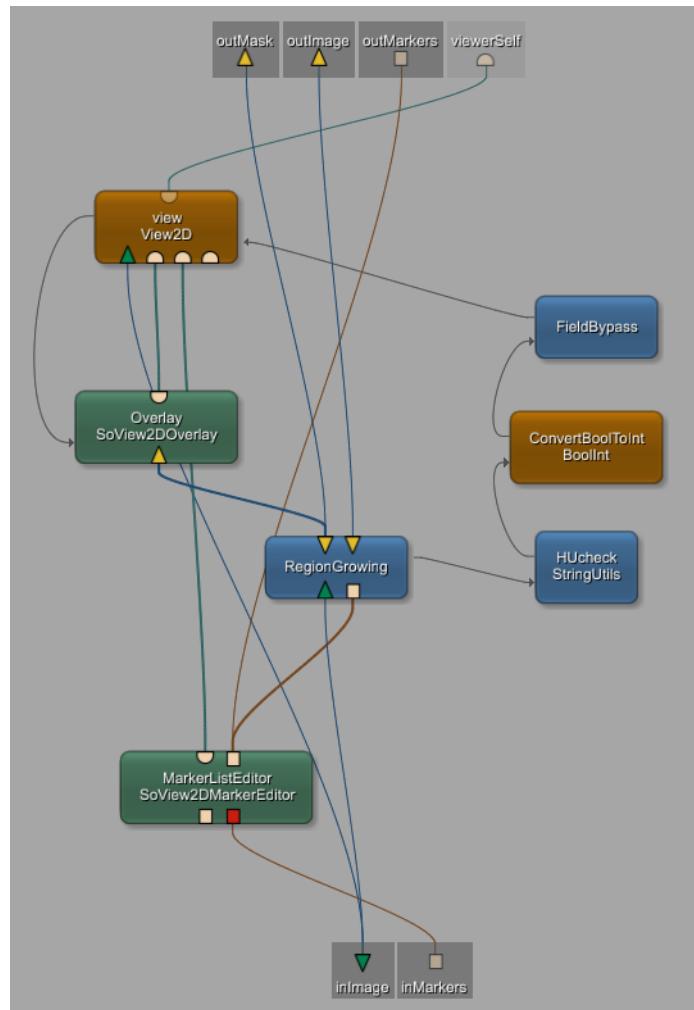
In the context menu of macro modules, the option **Show Internal Network** is available. If selected, the network of the macro is opened on another network tab.



Tip

Refer to chapter [Section 4.3.10, “Preferences — Shortcuts”](#) for a shortcut to open a macro's internal network.

Figure 3.22. RegionGrowingMacro — Internal Network



The pseudo-connectors shaded in gray are placeholders and indicate the input (bottom) and output (top) parameters of the macro, which constitute the connectors of the macro module. They are automatically drawn at the edges of the bounding box of the network. Important points about them:

- They cannot be moved or removed interactively but can only be changed in the script.

- They cannot be selected in a rectangle but each of them can be clicked, in which case the input/output square, the connection(s), and the connected module(s) are highlighted.



Note

Modules in an internal network of a macro that are connected to the macro's input / output fields (visualized by being connected to the pseudo-connectors) cannot be removed interactively from the network.

On an attempt to remove such a module, a window with a warning pops up. If such a module needs to be removed, the according connection needs to be removed in the scripting and the macro reloaded first.

If the macro module only has visible inputs, only those are drawn. An example for inputs only are the `View2D` and `View3D` modules.



Note

The tab of the internal network remains connected to the module from which it was opened. When the module is deleted or its network closed, the internal network is also closed.

3.10. Network Handling

Parts of a network can be selected by pressing the mouse button and dragging the mouse over the network (also called “rubber-band selection”). A selection rectangle appears that selects all modules that currently touch this rectangle.

While selecting modules with that rubber-band rectangle, the number of so-far selected modules is shown at the mouse cursor.

To select multiple nonadjacent modules, press **SHIFT** and click the modules.

To deselect all modules, press **ESC** or select an empty area of the network tab.

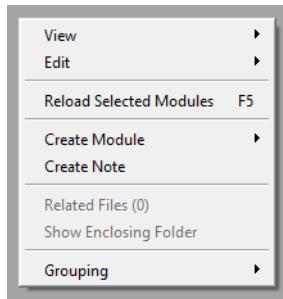
More than one network can be opened. They are displayed as tabbed view. To close the tab (and its network), click the respective Close (x) button. Alternatively, a network can be closed by clicking its tab with the middle mouse button.

A network quick search is available, see [Section 3.14, “Network Quick Search”](#).

3.10.1. Network Context Menu

Right-click the workspace to open the network context menu.

Figure 3.23. Network Context Menu



- For the Edit functions, see [Section 4.2, “Edit Menu”](#).
- For the View functions, see [Section 4.9, “View Menu”](#).

- For creating Notes, see [Section 3.12, “Using Notes”](#).
- For creating Groups, see [Section 3.11, “Using Groups”](#).

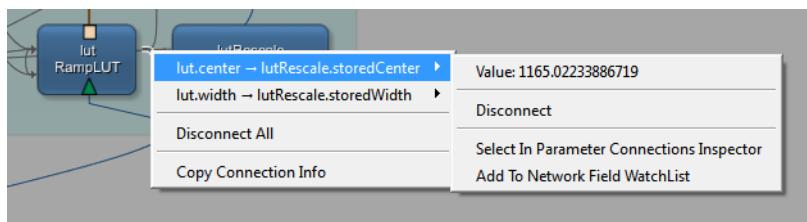
3.10.2. Connections Context Menus

3.10.2.1. Context Menu of Parameter Connections

Parameter connections are connections between the fields of modules. They may be created within the same module or between modules. For details on parameter connections, see Getting Started, chapter “Parameter Connection for Synchronization”.

The context menu of parameter connections in the network offers the following options:

Figure 3.24. Parameter Connection Context Menu



Parameter connections are always bundled between two connected modules. Therefore, several connections may be listed in the context menu of one parameter connection.

Outgoing parameter connections are positioned at the upper third of the module's left or right border, and incoming parameter connections are positioned at the lower third of the module's border.

If a module has parameter connections within its own fields, the parameter connection visualization forms a small loop at the border of that module.

For each connection, the following options are available:

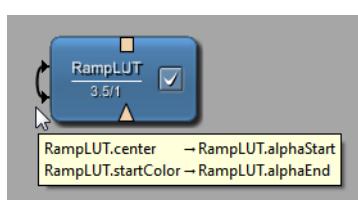
- **Value:** Shows the current value of the field.
- **Disconnect:** Disconnects the field connection.
- **Select in Parameter Connections Inspector:** Selects the field connection in the *Parameter Connections Inspector*.
- **Add to Network Fields WatchList:** Adds the connected fields to the *Network Fields WatchList*.

Additional options are:

- **Disconnect All:** Disconnects all listed parameter connections (can be undone/redone)
- **Copy Connection Info:** Copies the connection info string to the paste buffer

To disconnect internal parameter connections, click the small loop at the left side of the module and open the context menu to disconnect the internal parameter connections. Alternatively, internal parameter connections are also listed in the *Parameter Connections Inspector* and can be disconnected there.

Figure 3.25. Module with Internal/Self-Connected Parameter Connection



3.10.2.2. Context Menu of Data Connections

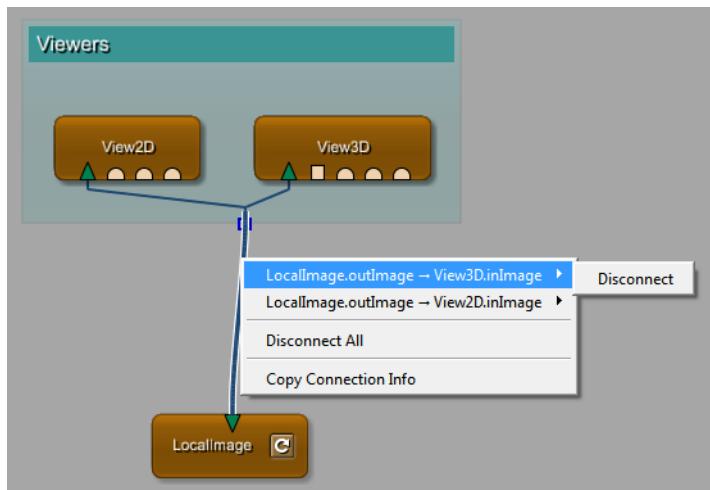
The context menu of data connections in the network only contains the **Disconnect** option for each item.

Additional options are:

- **Disconnect All:** Disconnects all listed data connections (can be undone/redone)
- **Copy Connection Info:** Copies the connection info string to the paste buffer

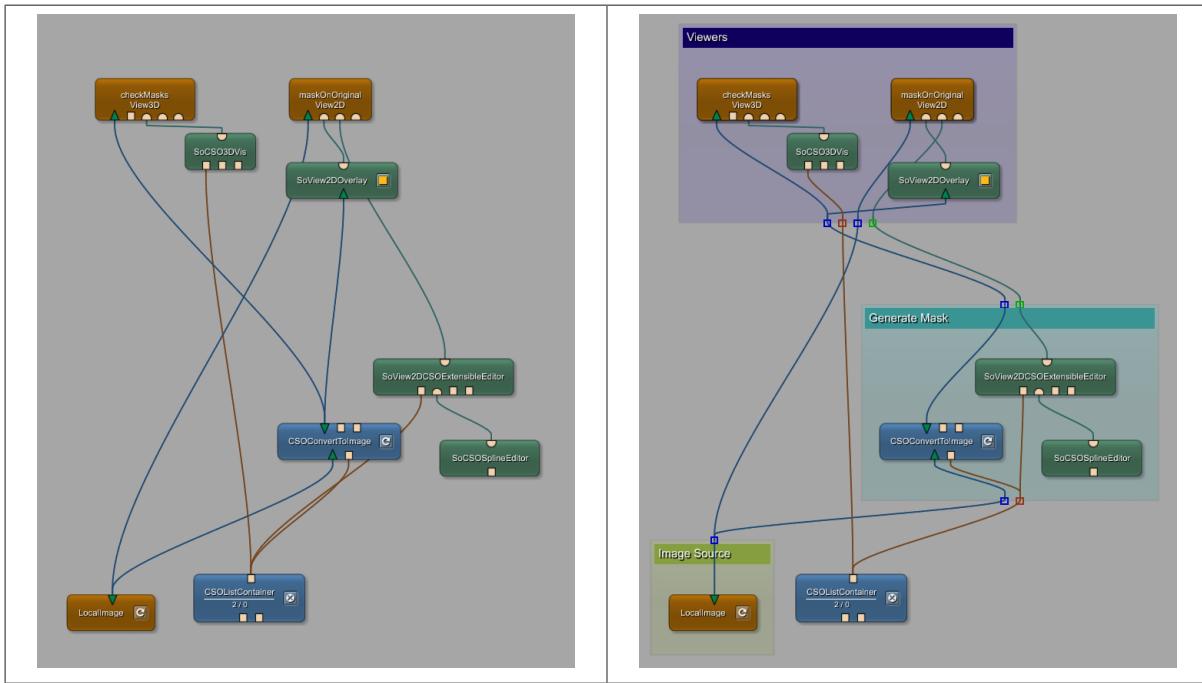
In case of grouped modules, data connections are bundled and more than one connector is listed in the context menu.

Figure 3.26. Data Connection Context Menu



3.11. Using Groups

Modules can be grouped. A group is helpful for organizing the network in the workspace, as the group can be moved as one unit. The default color of groups can be set in the Preferences, see [Section 4.3.7, "Preferences — Network Appearance"](#).

Table 3.37. Modules in Groups

Data connections are bundled optically as square, color-coded connectors at the bottom (input) or top (output) of the group. The colors correspond to the connection types: blue for ML, green for Open Inventor, brown for Base. Parameter connections are not bundled for groups.

The size of the group is set automatically by the bounding box of the modules and cannot be changed explicitly. To adjust it, move the modules within the group.



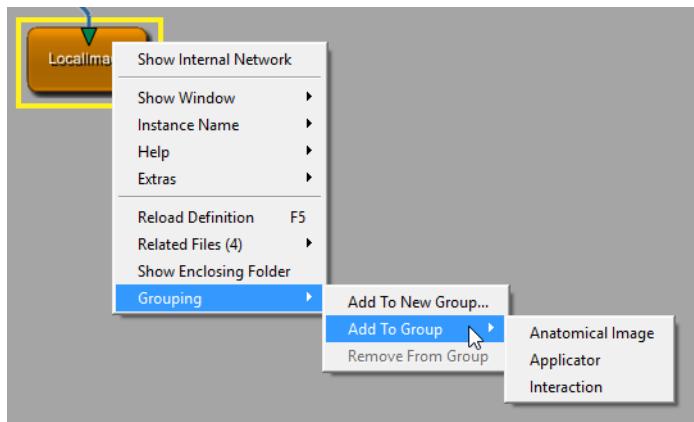
Note

Besides the optical appearance as “group”, the modules are not connected to each other in any special way. Groups are only a visual tool for improving the network handling. Consequently, the group feature should not be used excessively to organize complex networks; for this, groups should be converted to macros which are the recommended way to reduce complexity in the MeVisLab context, see [Section 3.11.2, “Editing, Converting, and Deleting Groups”](#).

The name and the color of a group can be scripted, see the Scripting Reference, `MLABNetworkModelItemGroup`.

3.11.1. Creating Groups and Adding/Removing Modules

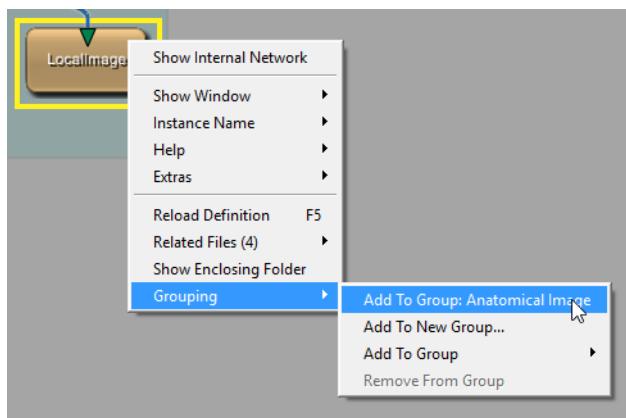
Creating groups and adding to/removing modules from groups is done via the context menu of the selected module(s).

Figure 3.27. Network Context Menu — Adding Groups**Tip**

If a single module is dropped over an existing module group, it is automatically added to that group.

**Note**

The color of modules that are positioned over a group but that are not part of any group is rendered slightly more saturated and bright.

Figure 3.28. Network Context Menu — Adding to a Specific Group**Add to New Group**

Creates a new group for the selected modules. Also allows adding the modules to an existing group by entering an existing group name.

The additional option **Add to Group: <TargetGroup>** is available when the module is already positioned within the target group's bounding box.

Add to Group

Adds the selected modules to one of the existing groups, which can be selected in the submenu.

Remove from Group

Removes the selected modules from the group.



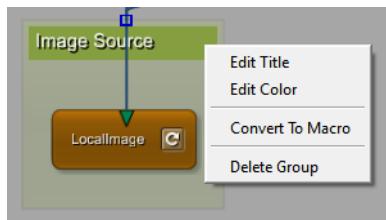
Tip

To move modules from one group to another, simply select them and add them to another or a new group. Since modules can only be part of one group, this course of action will effectively move the modules.

3.11.2. Editing, Converting, and Deleting Groups

A group can be edited, converted to a macro, or deleted via the context menu of its title.

Figure 3.29. Group Context Menu



Edit Title

Allows editing a new group title/name. The title must be unique within the current network.



Tip

Refer to chapter [Section 4.3.10, “Preferences — Shortcuts”](#) for a shortcut for editing a group's title.

Edit Color

Allows editing the color of the group. This has no effect on the default color which is set in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).



Note

When changing the color setting, the alpha value is 255 by default, that means the group is opaque. To give the group the appearance of standard groups, enter the original alpha value of 38.

Convert To Macro

Creates a (local) macro from the group, see [Section 4.1.12, “Create Local Macro”](#). The network has to be saved for this.

Delete Group

Deletes the group. Can be undone/redone. Does not remove the modules in that group.



Tip

To remove a group and all its modules, double-click the group's title bar; this selects the group and all its modules. Now the group and its modules can be removed by pressing DEL.

3.11.3. Copying Groups Including Modules

For copying a complete group:

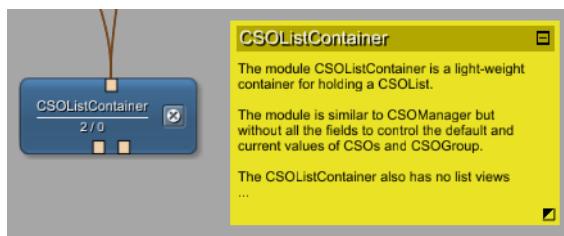
1. Double-click the group title bar to select all modules of the group.
2. Duplicate the group via the Edit menu or the respective keyboard shortcuts see [Section 4.3.10, “Preferences — Shortcuts”](#).

A number is added automatically to the title of the group copy, for example “title2”, “title3”.

3.12. Using Notes

Notes allow for adding annotations and additional information to a network or group. In contrast to the Comment module, notes are immediately visible and readable. The default color of notes can be set in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).

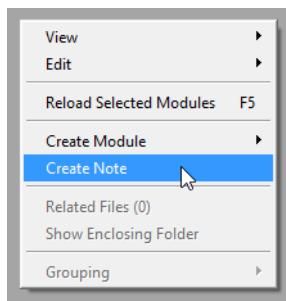
Figure 3.30. Note (expanded)



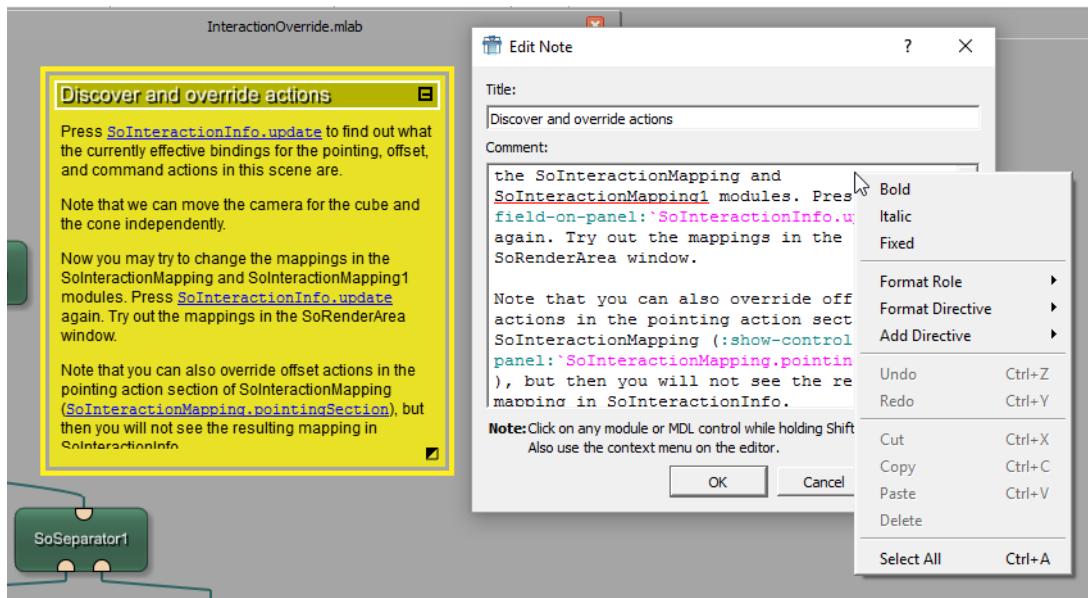
3.12.1. Creating Notes

Notes can be created via the network context menu.

Figure 3.31. Creating a Note



Click **Create Note** and create a new note item by entering a title and a comment. The width of the note title defines the minimum width of the note in the display. Note titles do not have to be unique, so there can be more than one note with the same title. Note comments are not limited in size. If the full note text cannot be displayed, three dots (“...”) are displayed to signal that more text is available. Text with more complex formatting (bold, lists, etc.) may be pasted into the note editor but the formatting will be lost. Instead one can use the formatting options also available for the help editor as described in [Section 26.9.2, “Formatting”](#).

Figure 3.32. Dialog for editing of notes, with context menu**Tip**

Since notes do not offer scrolling, it is recommended to adjust the note size and/or the amount of information so that all note text is visible at once. Otherwise, it may be necessary to resize the note or go via **Edit Text** to read all text.

The formatting of the text also changes slightly while zooming the network because of dynamic font alignment, therefore as a rule-of-thumb you should make the note slightly bigger than needed for the current zoom level.

Notes can be scripted, see the Scripting Reference, MLABNoteItem.

3.12.2. Handling Notes

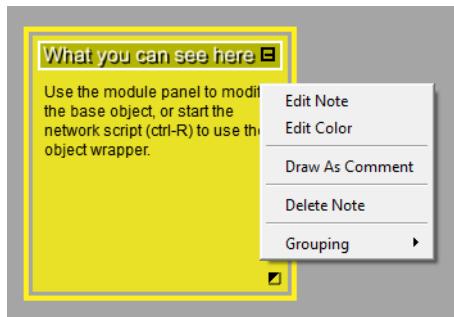
Notes can be collapsed/expanded by clicking the minus/plus buttons (⊖/⊕) or by double-clicking the note title.

Figure 3.33. Note (collapsed)

Notes can be resized by dragging the resize icon at the lower right (□).

3.12.3. Editing and Deleting Notes

A note can be edited or deleted via the context menu of its title bar.

Figure 3.34. Note Context Menu**Tip**

Double-click the text area to edit the note. For further note-related shortcuts, see [Section 4.3.10, “Preferences — Shortcuts”](#).

Edit Note

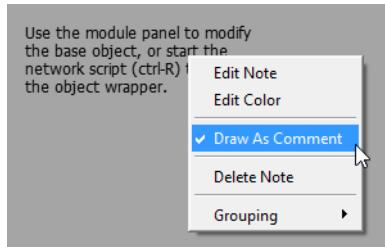
Allows editing the note title and comment.

Edit Color

Allows editing the color of the note. This has no effect on the default color which is set in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).

Draw As Comment (toggle)

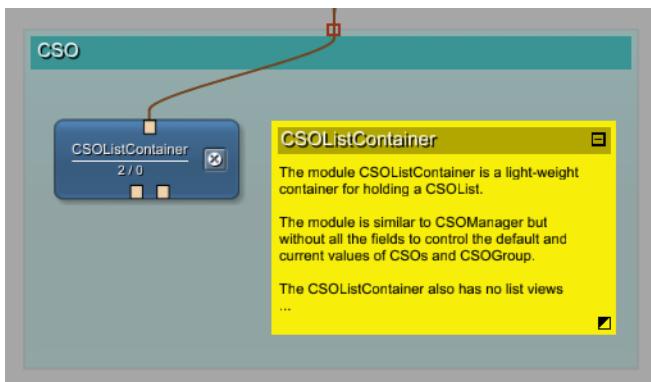
If this option is active, the note is displayed without title and without a surrounding box, directly on the network background.

Figure 3.35. A note displayed as a network comment**Delete Note**

Deletes the note. Can be undone/redone.

Grouping

Notes can also be assigned or removed from groups just like modules (see above, [Section 3.11, “Using Groups”](#)). The bounding box of the group is adjusted accordingly. Notes in a group will be moved with the group; notes in general can be moved anywhere in the workspace.

Figure 3.36. Note in a Group

3.12.4. Copying Notes Including Text

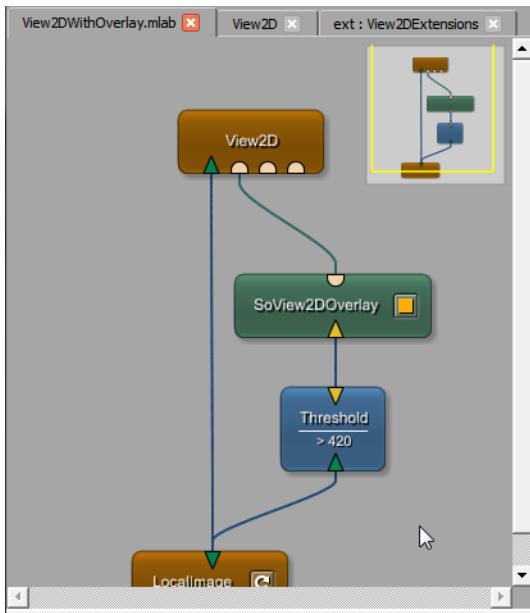


Note

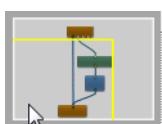
Notes can be copied or cloned, just like a module.

3.13. Using the Mini Map

When zooming into a network so that it is not fully visible in the workspace, a mini map is displayed (default setting). This allows navigating in the network. The settings and appearance of the mini map can be edited in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).

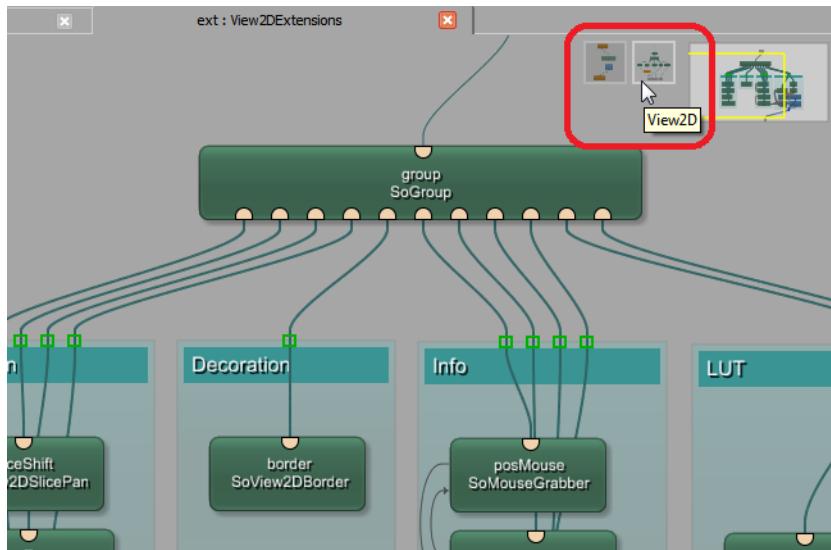
Figure 3.37. Mini Map

The highlighted area of the map can be dragged with the mouse. The network display is adjusted accordingly.

Figure 3.38. Navigating in the Mini Map

In case of macro modules, the network(s) from which the internal network is opened can be displayed in a hierarchy when the option **Show parent navigation frames** is set, see [Section 4.3.7, “Preferences — Network Appearance”](#) for details.

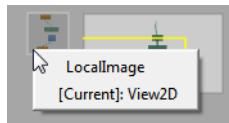
Figure 3.39. Parent Navigation Frame for Macro Modules



Parent navigation frames (PNF) can be used for navigating to networks higher in the hierarchy or to open internal macro networks of a parent frame.

- Click a PNF to display its network.
- Right-click a PNF to open a context menu that lists the names of all macros in that network. Click a macro to open and activate its network. The currently displayed macro network is marked as “[Current]”.

Figure 3.40. Parent Navigation Frame Context Menu



If a parent network is not open, the PNF is rendered as plain, dark gray square. Upon clicking it, the network is opened and the PNF is updated to show the small mini map rendering.

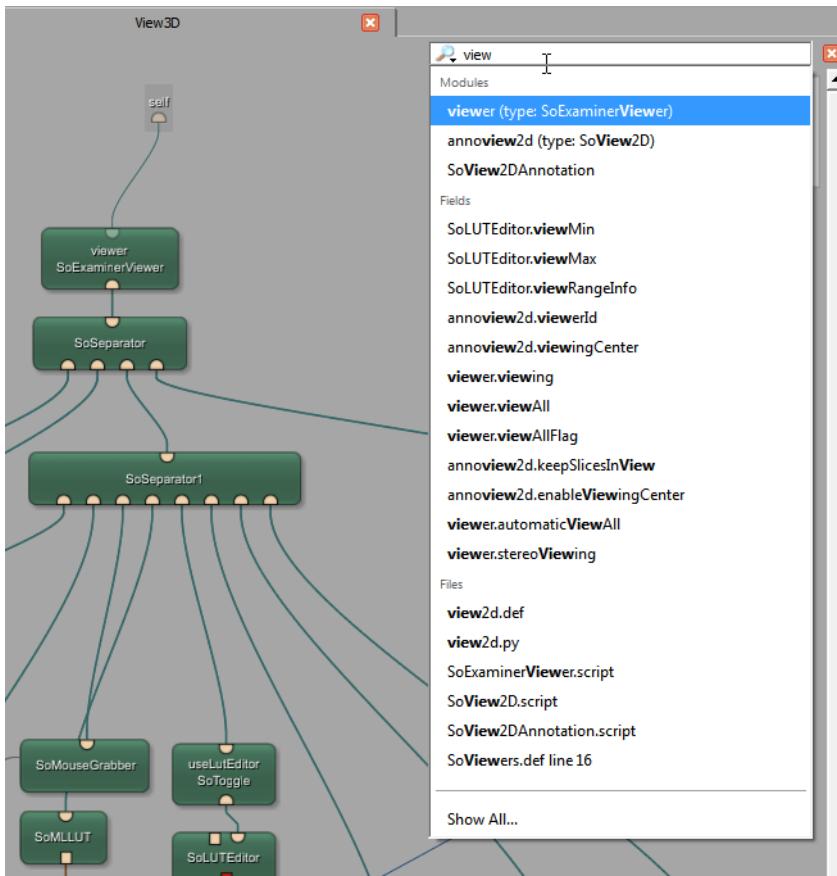


Tip

The PNF shows not only the modules and their connections but also the highlighting state of modules. Make sure to select a macro before opening its internal network so that the PNF displays that macro as highlighted.

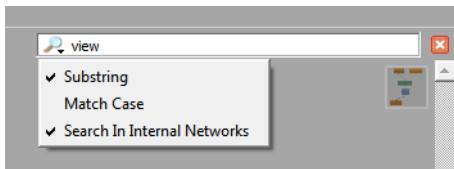
3.14. Network Quick Search

A quick search for networks is available with the keyboard shortcut "Find" found in [Section 4.3.10, “Preferences — Shortcuts”](#). It opens on the top right of the network.

Figure 3.41. Network Quick Search

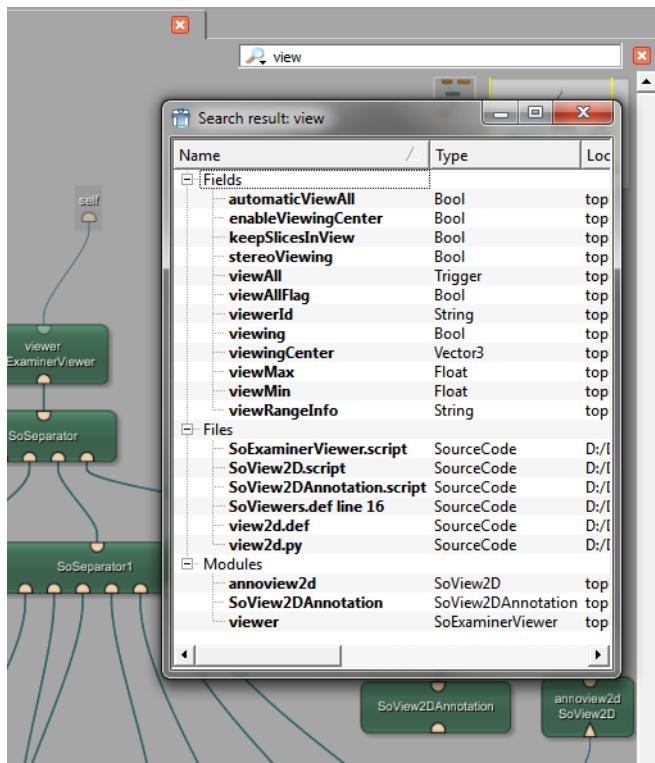
The search results are sorted by subjects (modules, fields, files).

Search options are available by clicking the magnifier button:

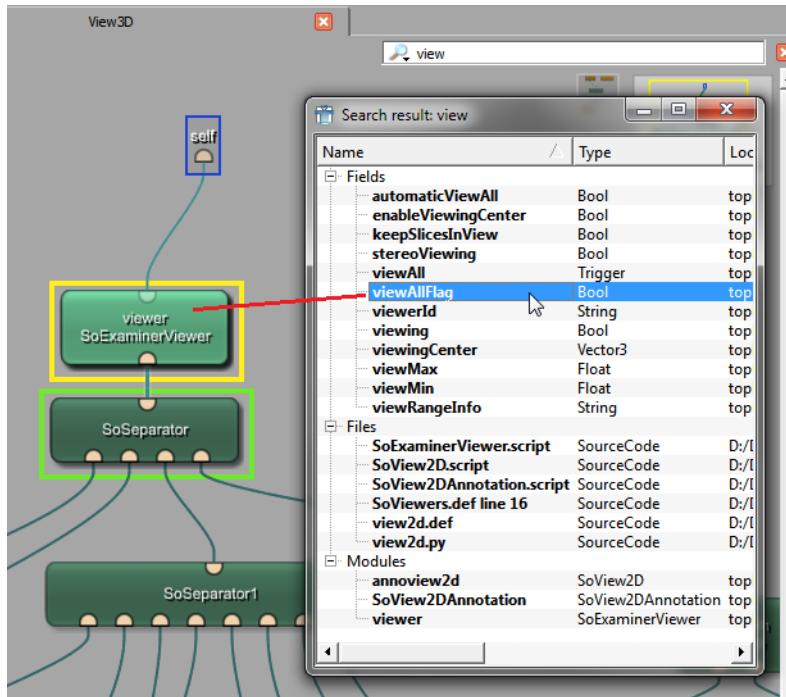
Figure 3.42. Network Quick Search — Options

- **Substring:** If selected, extends the search to module name substrings. Effectively works as if adding wildcards, for example *image*.
- **Match Case:** If selected, the search differentiates between lower and upper case letters.
- **Search In Internal Networks:** Searches in the internal networks of macro modules.

Clicking **Show All** opens the results in a new, persistent window. This way, you can have several search result lists open.

Figure 3.43. Network Quick Search — Show All Results

Double-clicking a search result has different effects depending on the result type:

Figure 3.44. Network Quick Search — Highlight Results

- Double-clicking a field highlights and zooms to the module that contains the field.
- Double-clicking a module highlights and zooms to the module.
- Double-clicking a file opens it in the integrated text editor MATE.

3.15. Network Selector

MeVisLab offers a network selector with a network preview similar to the Task Switcher or Flip on Windows systems.

The use of the network selector can be toggled in the **Preferences** on the **General** tab: [Section 4.3.1, “Preferences — General”](#).

To open the network selector, hold down the right mouse button and turn the mouse wheel. With an open network selector, turning the mouse wheel selects a next or a previous network, depending on the direction in which the mouse wheel is being turned. On releasing the right mouse button, the currently selected network in the selector is set as the current network in MeVisLab. Also, the selected network is maximized in the IDE.

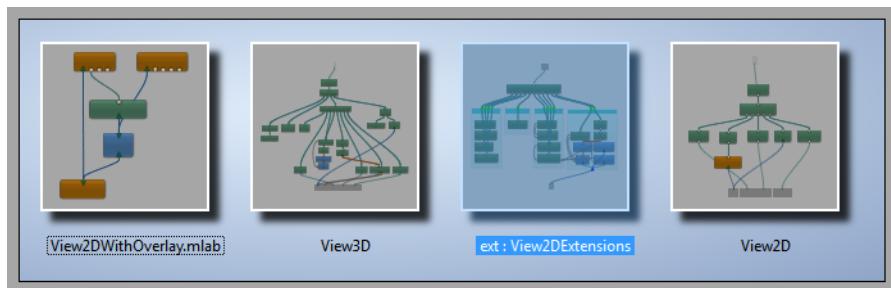
The network selector also opens on pressing CTRL+Tab or CTRL+SHIFT+Tab. Consecutive presses of those shortcuts navigate the available networks in the preview. On releasing the key combination, the last selected network is made active in MeVisLab.



Note

The mentioned shortcuts are the defaults, refer to [Section 4.3.10, “Preferences — Shortcuts”](#) for the current shortcuts.

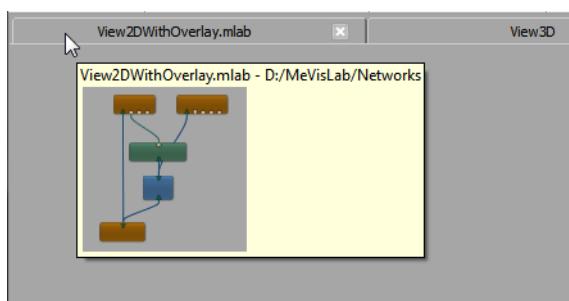
Figure 3.45. Network Selector in Action



3.16. Network Preview

When hovering with the mouse cursor over the tab of a network, a network preview is rendered as a tool tip.

Figure 3.46. Network Selector in Action



3.17. Network Mouse Gestures

The Windows version of MeVisLab supports mouse gestures for network interaction.

The use of network gestures can be toggled in the **Preferences** on the **Network Interaction** tab: see [Section 4.3.8, “Preferences — Network Interaction”](#).

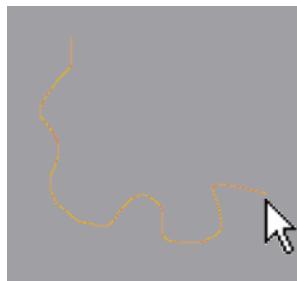
A mouse gesture is performed by holding the right mouse button down while the mouse cursor is over the network background, and moving the mouse. The trail of the mouse while performing a gesture is rendered on the network for a better orientation.

The trail of the mouse gesture is color coded. If the gesture is not yet recognized, the trail is rendered in a yellow color, on recognizing a gesture the color changes.

The trail is cleared on releasing the right mouse button. No context menu is shown in case of having performed a gesture, not even if the gesture has not been recognized.

A gesture can be reset by pressing **ESC** while still holding the right mouse button down; the trail so far is then cleared as well.

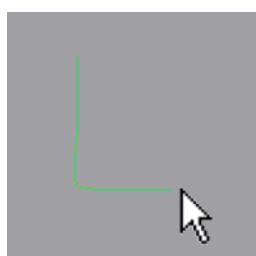
Figure 3.47. Trail of Unrecognized Mouse Gesture



3.17.1. Gesture for Closing the Current Network

The current network can be closed by a gesture that consists of a stroke down and then a stroke to the right, like drawing the letter 'L'. Once the gesture has been recognized by the system, the trail is rendered in a green color. To complete the command, simply release the right mouse button.

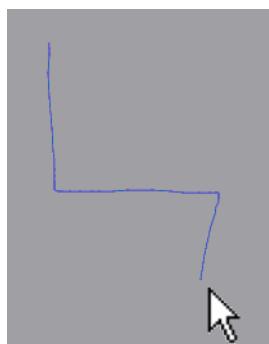
Figure 3.48. Mouse Gesture for Closing the Current Network



3.17.2. Gesture for Closing the Current Network Without Requester

The current, changed and unsaved network can be close without showing the requester asking whether the network should be saved by a gesture that starts as the gesture before (drawing an 'L'), but with an additional stroke down at the end. Once the gesture has been recognized by the system, the trail is rendered in a blue color. To complete the command, simply release the right mouse button.

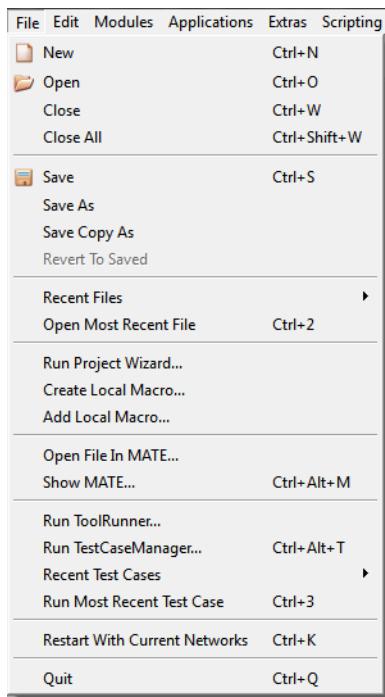
Figure 3.49. Mouse Gesture for Closing the Current Network Without Requester



Chapter 4. Menu Bar

4.1. File Menu

Figure 4.1. File Menu



4.1.1. New

Creates a new MeVisLab network document.

4.1.2. Open

Opens an existing MeVisLab network from file (extension .mlab). Alternatively, the .mlab file can be dragged onto the workspace to open it in a new network window.

Some other file types may also be opened directly by dragging them onto the workspace (no network needs to be open for that):

- A dragged image file (like .dcm, .tif, .png, etc.) creates an `ImageLoad` module which automatically loads the file.
- A dragged file readable by WEM modules (like .wem, .off, .obj, .ply, etc.) creates a `WEMLoad` module which automatically loads the file. See the HTML help for the `WEMLoad` module for more information.
- A dragged file readable by CSO modules (typically .cso) creates a `CSOLoad` module which automatically loads the file.

4.1.3. Close

Closes the current network.

4.1.4. Close all

Closes all open network.

4.1.5. Save

Saves the current network.

For saved networks, the AutoSave files are discarded. See [Section 4.3.1, “Preferences — General”](#).

4.1.6. Save As

Writes the current network to file with a new name.

For saved networks, the AutoSave files are discarded. See [Section 4.3.1, “Preferences — General”](#).

4.1.7. Save Copy As

Writes a copy of the current network to file with a new name.

4.1.8. Revert To Saved

Reverts to the last saved version of the current network. (Option is only available if network was changed.)

4.1.9. Recent Files

Allows selecting recently opened network from file. The maximum number of recent files is twenty. The list of recent files is not deleted upon installing a new version of MeVisLab. For more information about storing preferences, see [Section 4.3, “Preferences”](#).

4.1.10. Open Most Recent File

Opens the most recent network file. This function can be triggered with a keyboard shortcut, see [Section 4.3.10, “Preferences — Shortcuts”](#).

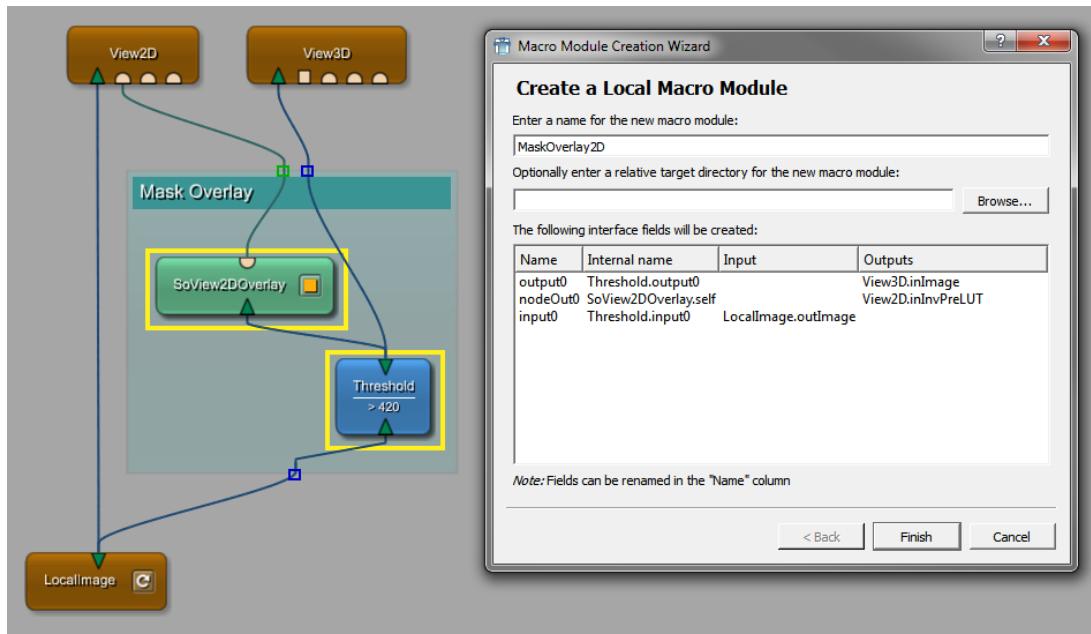
4.1.11. Run Project Wizard

Starts a wizard for creating new modules, packages and installers (ADK license required for the latter). See [Chapter 25, Project Wizard](#).

4.1.12. Create Local Macro

Creates a new macro module locally to the current network path, based on the currently selected modules or a module group, see [Section 3.11.2, “Editing, Converting, and Deleting Groups”](#). The necessary inputs and outputs will be added automatically. The display names of interface fields can be changed but not the internal names.

Figure 4.2. Local Macro Creation



Locally defined macro modules may be used in complex networks to encapsulate sub-networks as independent functional units with a defined interface to other network components. This way they carry out an application-specific function which would not be of use for any other application. Therefore they are not added to the common MeVisLab module database, i.e. they are not declared in a `.def` file.

The following items are created:

- the files `<ModuleName>.script` and `<ModuleName>.mlab` in the current network path directory
- the new local macro module on the current network workspace



Note

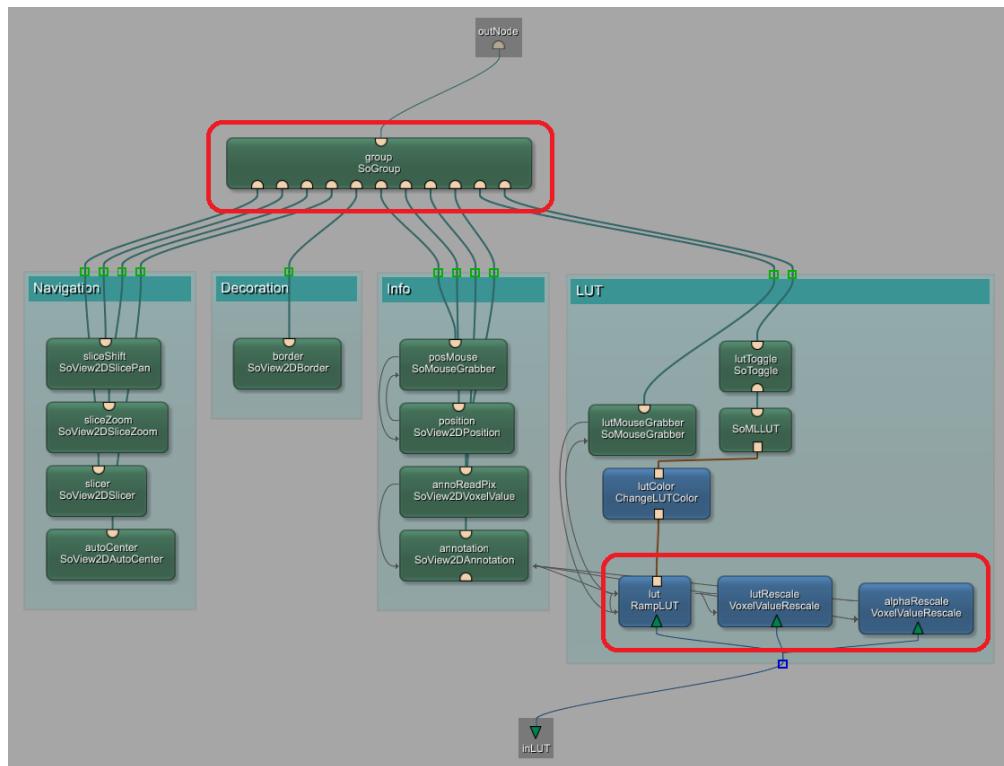
A local macro module is not available in the MeVisLab module database, as no `.def` file is created. The module cannot be accessed via the **Modules** menu or the **Modules Search**. Local modules can only be added to networks in the same network path, see [Section 4.1.13, “Add Local Macro”](#).

To differentiate local macros from global macros, `./` is prepended to the module name/type in the network view to hint at the fact that this module only exists relative to the location of the network file.



Note

During the conversion to local macro, modules need to be disconnected and reconnected. Connections that are set by script, i.e., forwarded from an outer macro, cannot be disconnected, so an alert will be issued: “Unable to remove the module [module name] with connections set by script.” A similar alert is issued when trying to remove a module connected like this. In the next figure, modules within a red rectangle can neither be deleted, disconnected from the inputs/outputs, or added to a local macro.

Figure 4.3. Modules Connected to Outer Macros

For an introduction on macros, read Getting Started, chapter “Introduction to Macro Modules”.

4.1.13. Add Local Macro

Adds a locally defined local macro module to the current network, see [Section 4.1.12, “Create Local Macro”](#). Choose the `<ModuleName>.script` file in the file dialog to add the local macro module. The local macro needs to be defined in the same folder as the network it should be added to, or in a sub-folder of this folder.

4.1.14. Open File in MATE

Opens MATE with a file dialog in which one or more files can be opened, see [Chapter 26, MATE](#).

4.1.15. Show MATE

Opens MATE without files, see [Chapter 26, MATE](#).

4.1.16. Run ToolRunner

Starts the ToolRunner, see the ToolRunner documentation.

4.1.17. Run TestCaseManager

Starts the TestCaseManager of the TestCenter, see the TestCenter Reference and the Getting Started, chapter 16, “Using the TestCenter”.

4.1.18. Recent Test Cases

Lists the last test cases that have been run from the TestCaseManager. Selecting an entry from this list will open the TestCaseManager and run the selected test case.

4.1.19. Run Most Recent Test Case

This will open the TestCaseManager and run the top-most entry of the recent test cases list.

4.1.20. Restart with Current Networks

Restarts MeVisLab with all networks currently opened. This is necessary for a complete DLL update and useful when developing new ML and Inventor modules. Alternatively, you can press a shortcut to restart MeVisLab with the current network. Refer to [Section 4.3.10, “Preferences — Shortcuts”](#).



Note

MeVisLab restarts in the same mode as it was started originally, regarding the `-quick` option.

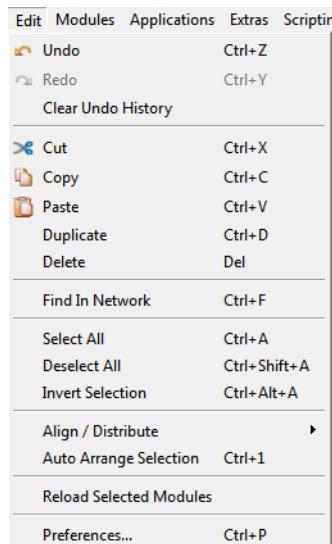
4.1.21. Quit

Quits MeVisLab. If unsaved changed networks are open, a message appears.

4.2. Edit Menu

For editing modules, module groups and connections, the typical text edit shortcuts for each platform can be used. See [Section 4.3.10, “Preferences — Shortcuts”](#).

Figure 4.4. Edit Menu (Windows example)



4.2.1. Undo

Undoes the last edit action.

4.2.2. Redo

Redoes the last undo action.

4.2.3. Clear Undo History

Clears the undo cache.

4.2.4. Cut

Cuts the selected subnetwork from the current network. The subnetwork is cached with all connections and field values of the contained modules.

4.2.5. Copy

Copies the selected subnetwork in the current network. The subnetwork is cached with all connections and field values of the contained modules.

4.2.6. Paste

Pastes the copied or cut subnetwork into the current network.

4.2.7. Duplicate

Duplicates (copies and pastes) a subnetwork within the current network.

4.2.8. Delete

Deletes the selected subnetwork from the current network.

4.2.9. Select All

Selects all modules and their connections in current network.

4.2.10. Deselect All

Deselects all modules in the current network.

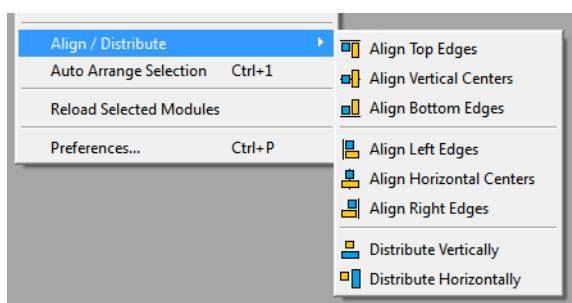
4.2.11. Invert Selection

Inverts the selection by selecting all currently unselected modules and deselecting all selected modules in the current network.

4.2.12. Align / Distribute

Aligns along centers and edges of modules, or distributes selected modules evenly in their bounding box. See the icons for the effect each option has.

Figure 4.5. Align / Distribute



A toolbar with these buttons is available via **View → Toolbars**.

4.2.13. Auto Arrange Selection

Arranges the selected modules of the current network automatically with an animation. This is very useful for automatically generated networks where all modules are placed onto one spot otherwise.

If no modules are selected, the whole network is being automatically arranged.

Modules with only inputs are arranged at the top of the network and modules with only outputs are arranged at the bottom of the network. Other modules are arranged in layers in between. Groups are arranged internally as if they were networks and the groups themselves are arranged in their context as if they were just large modules. Parameter connections do not influence the arrangement.

The horizontal and vertical spacings between modules can be set with the **Module Arrangement Spacing** settings described in [Section 4.3.8, “Preferences — Network Interaction”](#).

The automatic arranging of a network (selection) can be triggered with a keyboard shortcut, see [Section 4.3.10, “Preferences — Shortcuts”](#).

4.2.14. Reload Selected Modules

Reloads the module database of the selected modules in the current network document. Changes in the .def, .script, and .py files are updated and applied to the selected modules.

The successful reloading of modules is indicated by a short color-flashing animation of the module(s).



Note

If a macro is reloaded, its internal network is not reloaded, just the macro's GUI definition and scripting files.

4.3. Preferences



Note

Settings in the Preferences panel overwrite the corresponding settings in the mevislab.prefs file.

The Preferences (and other information, like the list of recent files or the stored user layouts) are saved in a way that they are not overwritten when updating/reinstalling MeVisLab:

- On Windows: in the Registry in `Workstation/HKEY_CURRENT_USER/Software/Mevis/MeVisLab`.
- On Mac OS X: in `$HOME/Library/Preferences/de.mevis.MeVislab.plist`.
- On Linux: in `$HOME/.config/MeVis/MeVisLab.conf`.

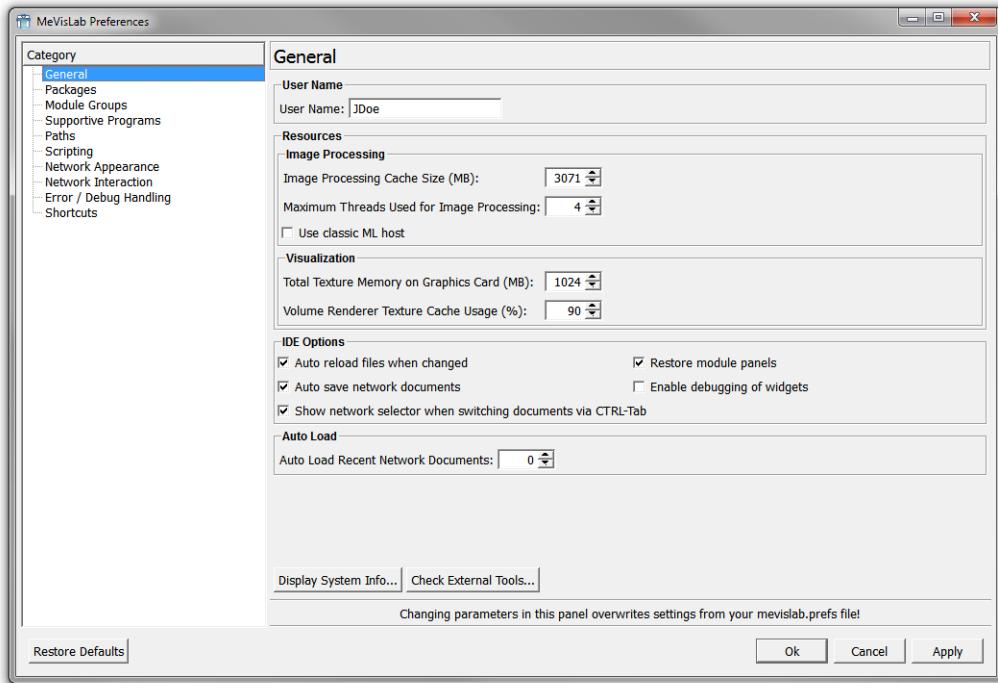


Tip

For many options in the Preferences, a mouse-over tip is available.

4.3.1. Preferences — General

Figure 4.6. Preferences — General



User Name

The user currently signed in at this computer.

Resources

The optimal Resources settings depend on the system and platform. Use the default settings if in doubt.

Image Processing

Image Processing Cache Size (MB)

Defines the memory available for caching (intermediate) ML image tiles/pages within a network of image processing modules. Reducing the cache size will slow down the image processing pipelines because images will be recalculated more frequently in individual modules. Cache sizes too large might cause a collapse of your system because of reduced memory for other programs. For 2 GB RAM, a value of 512 MB is well-tested. For details, see the ML Programming Guide, “Optimizing Data Flow in Module Networks”.

Maximum Threads Used for Image Processing

Gives the number of parallel threads for image processing. For more details on multi-threading, see [Section 27.7, “Multi-threading in MeVisLab”](#).

Use classic ML host

If checked, the classic ML host (the only available option before MeVisLab 2.5) is used. Otherwise, a new ML host is used that implements an optimized multi-threading for the ML.

Visualization

Total Texture Memory on Graphics Card (MB)

Gives the amount of texture memory (texture RAM, TRAM) on the graphics card to be used for MeVisLab texture processing, for example in the `View2D` module.

Volume Renderer Texture Cache Usage (%)

Defines the percentage of texture memory that the GVR volume renderer may use.

IDE Options

Auto reload files when changed

If selected, the `.def`, `.script`, and `.py` files of a module are reloaded when reloading the module panel (by double-clicking the module or selecting **Show Window → Panel** from the module context menu).



Note

Networks `.mlab` files (for macro modules) are not reloaded.

Auto save MeVisLab documents

If selected, MeVisLab networks are auto-saved as `<NetworkName>.mlab.auto` upon major changes. This allows for restoring in case of system crashes. Auto-saved copies are deleted when the according networks are saved.

Show network selector when switching documents via CTRL-Tab

If selected, pressing **CTRL+TAB** (or **SHIFT+CTRL+TAB**) does not switch directly to the next or previous network document, but shows a network selector preview. For more details, see [Section 3.15, "Network Selector"](#).

Restore module panels

If selected, opening a network restores all module panels (including window size and position) to the state (open, minimized, positions, etc.) when the network was last saved.

Enable debugging of widgets

Enables/disables debugging module panels. When enabled, **CTRL+left-clicking** a GUI control in a module panel opens the `.script` file in the default text editor at the line in which the GUI control is defined.

Auto Load Recent Files

Gives the number of recent networks to be loaded automatically on MeVisLab startup. This may considerably slow down the startup.



Note

If loading a network leads to a crash, this option can be harmful, because on the next start of MeVisLab, that network will automatically be loaded, which in turn will lead to a crash! Use this option with care!

Display System Info...

Displays system information regarding the Open GL vendor, the available GL extensions, and more in an extra window.

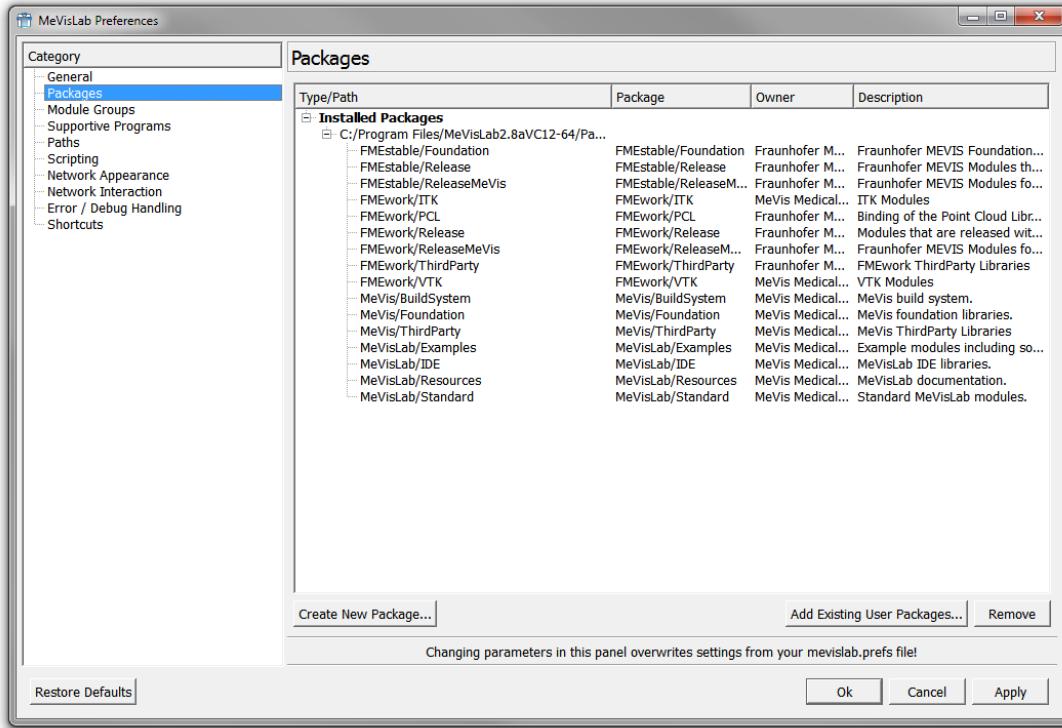
Check External Tools...

Starts the Tools Check tool that checks for software necessary for certain build tasks. This tool is also part of the ToolRunner. See the ToolRunner Documentation for details.

4.3.2. Preferences — Packages

MeVisLab modules are organized in packages. These are defined as certain folder structures (see the Package Structure documentation for details). The Packages category gives an overview over the available and active packages.

Figure 4.7. Preferences — Packages



The packages are separated into:

- **User Packages:** packages found in the user path (packages in other paths can be added manually). These are the default packages for user-defined modules.
- **mevislab.prefs:** packages resulting from the paths given in the prefs file.
- **Installed Packages:** packages resulting from an installation of, e.g., the MeVisLab SDK.

If a package with the same PackagelIdentifier is found more than once, the last package found will overwrite the previously loaded packages (in the order given above, see the Package Structure documentation for details). These will be grayed out and labeled “(Overwritten)”.

Create New Package

Opens the Package Wizard, see [Section 25.5, “Packages”](#).

Add Existing User Packages

Opens the default file browser to add a user package. Folders are read recursively and all packages below them are automatically included.

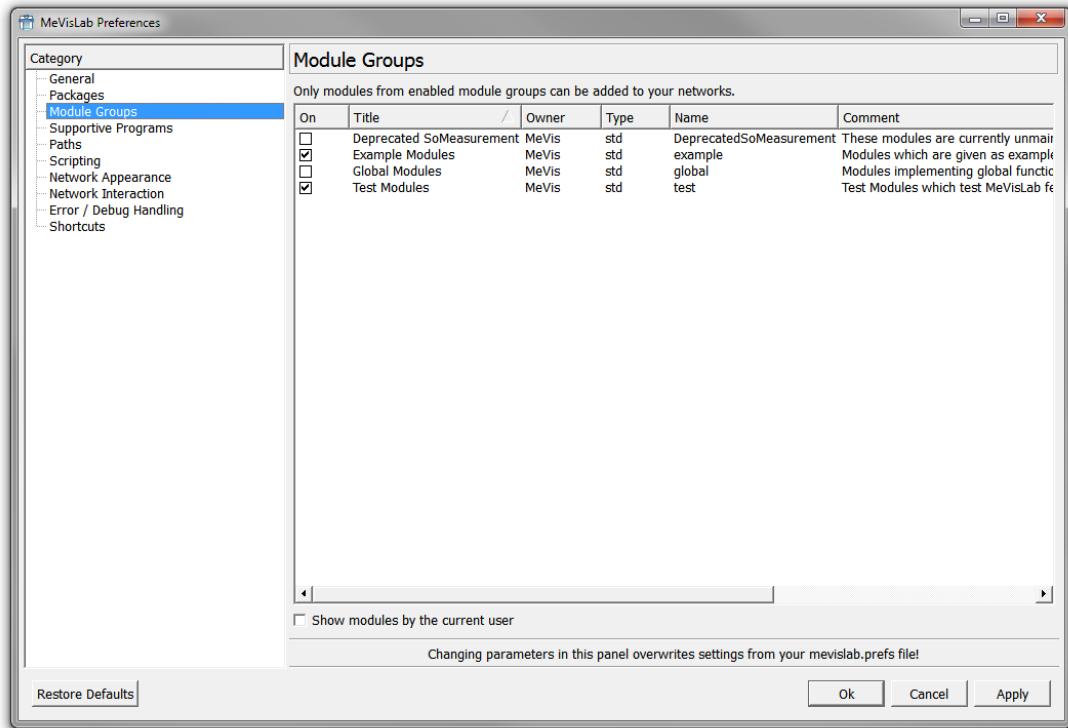
Remove

Removes the selected user package from the path of MeVisLab. (Installed packages cannot be removed.) Removed user packages can always be re-added later.

4.3.3. Preferences — Module Groups

The Module Groups category lists optional groups of modules that are not loaded by default. Check the corresponding group to get access to modules of the group.

Figure 4.8. Preferences — Module Groups

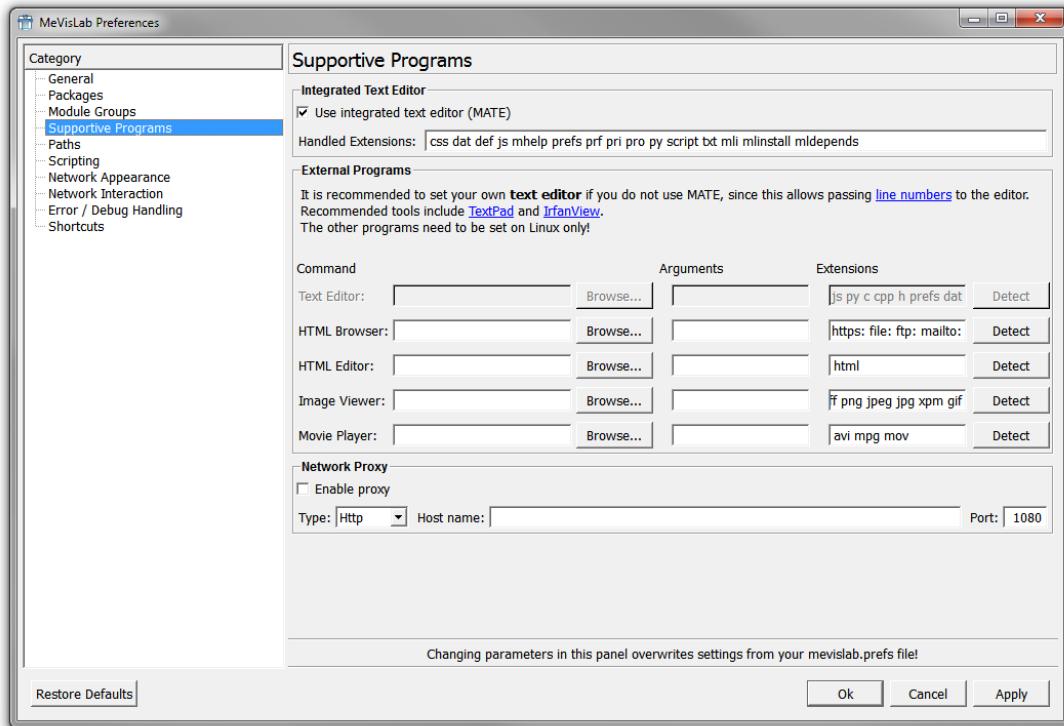


Scroll to the right to see additional comments and the number of modules for each group.

After confirming the selection with **OK**, the package groups of the selected modules and the user packages are scanned and loaded.

4.3.4. Preferences — Supportive Programs

Figure 4.9. Preferences — Supportive Programs



Internal Text Editor

Use internal text editor (MATE)

If selected, the internal text editor MATE is used, which offers many useful features for MeVisLab files. Recommended! See [Chapter 26, MATE](#) for details.

Handled Extensions

Allows to define a list of file extensions for which MATE is automatically opened.

External Programs

Allows to define external applications plus program arguments for file types used in MeVisLab. Click **Browse** to select the applications manually, or **Detect** to autodetect applications that work especially well with MeVisLab.



Note

On Linux it can happen that an external program depends on a third party library that MeVisLab provides. If problems occur, e.g., because a KDE program loads an incompatible Qt library from MeVisLab, then use a shell script that clears LD_LIBRARY_PATH before it calls the external program.

For example, to safely run konqueror from MeVisLab, place the following script in ~/bin and make sure ~/bin is the first entry in PATH before running MeVisLab.

```
#!/bin/sh
LD_LIBRARY_PATH= konqueror
```

Arguments are options added when starting the program. They can be entered manually or are added by the Detect feature.

Extensions lists the extensions for which the assigned program will be used. This overrides your system settings, but only in the MeVisLab context.

- **TextEditor:** Although it is recommended to use the internal text editor MATE, other text editors can be used in conjunction with MeVisLab. With the argument %F(%1), a file and a line number in it will be passed to the text editor. The Detect feature will check for TextPad on Windows (see the web link for installing) and set the options accordingly.
- **ImageViewer:** Graphic applications may be used in conjunction with MeVisLab. If none is set, the system default will be used. The Detect feature will check for IrfanView on Windows (see the web link for installing) and set the options accordingly.
- **MoviePlayer, HTMLBrowser, HTMLEditor:** may primarily needed to be set on Linux.

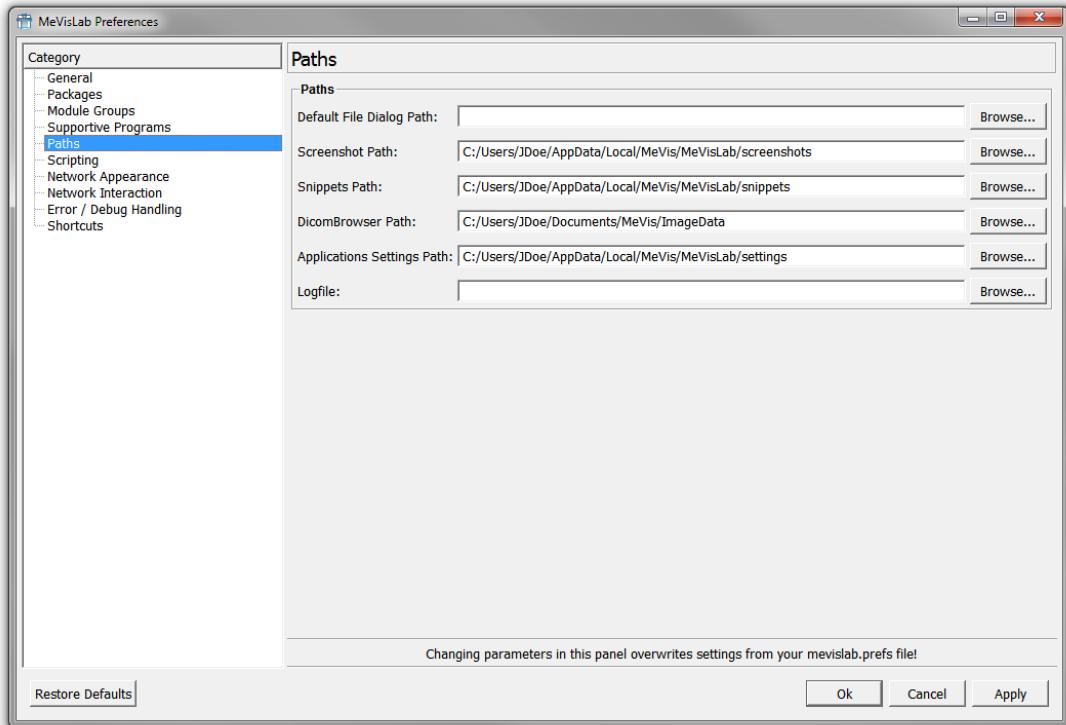
Network Proxy

Allows to set a proxy server for HTTP connections of the WebKit implementation of Qt.

- **Enable proxy:** If selected, a proxy is used by the WebKit implementation.
- **Type:** Sets the type of the used protocol.
- **Host name:** Sets the host name of the proxy server.
- **Port:** Sets the port of the proxy server.

4.3.5. Preferences — Paths

Figure 4.10. Preferences — Paths



Default File Dialog Path

Sets the default path in the file dialog. If none is entered, the path last used will be offered.

Screenshot Path

Sets the path for files of the View Screenshot Gallery, see [Chapter 19, Screenshot Gallery](#).

Snippets Path

Sets the path for network snippets, see [Chapter 24, Snippets List](#)

DicomBrowser Path

Sets the path where the `DICOMBrowser` module and the `OpenImage` module look for DICOM data by default.

Applications Settings Path

Sets the path in which MeVisLab applications save their settings.

Logfile

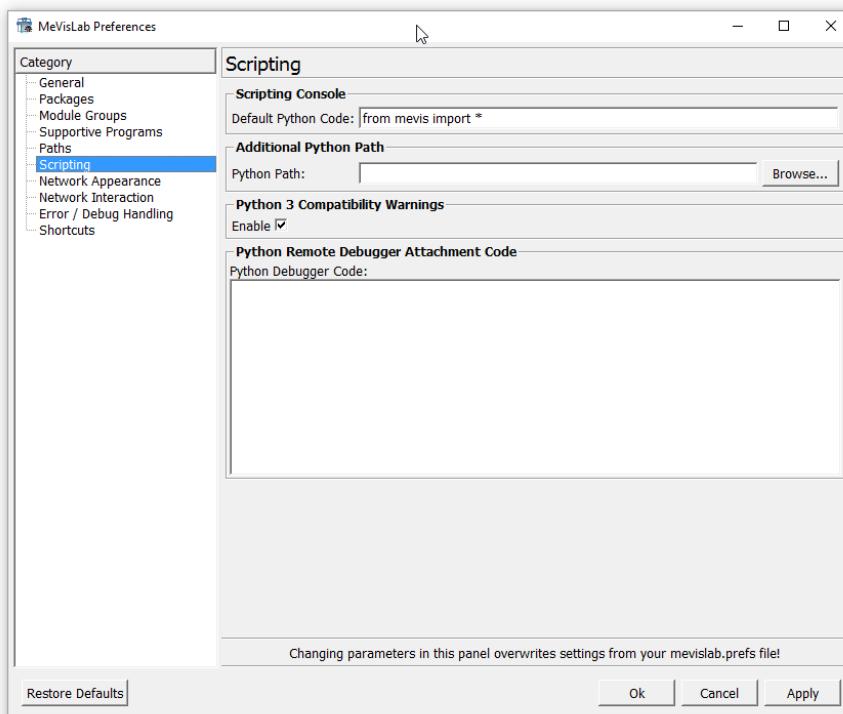
Sets the path to the logfile written by MeVisLab.

**Note**

You can also set the path to the logfile in the `.prefs` file with `logfile = <absolutePath/file.log>`.

4.3.6. Preferences — Scripting

Figure 4.11. Preferences — Scripting

**Scripting****Default Python Code**

Sets a default Python code snippet that is used in the scripting console as default.

Additional Python Path

Add the path to an additional Python package so it can be found in the import statement.

Python Debugger Code

Sets Python debugger code as described in the Scripting Reference.



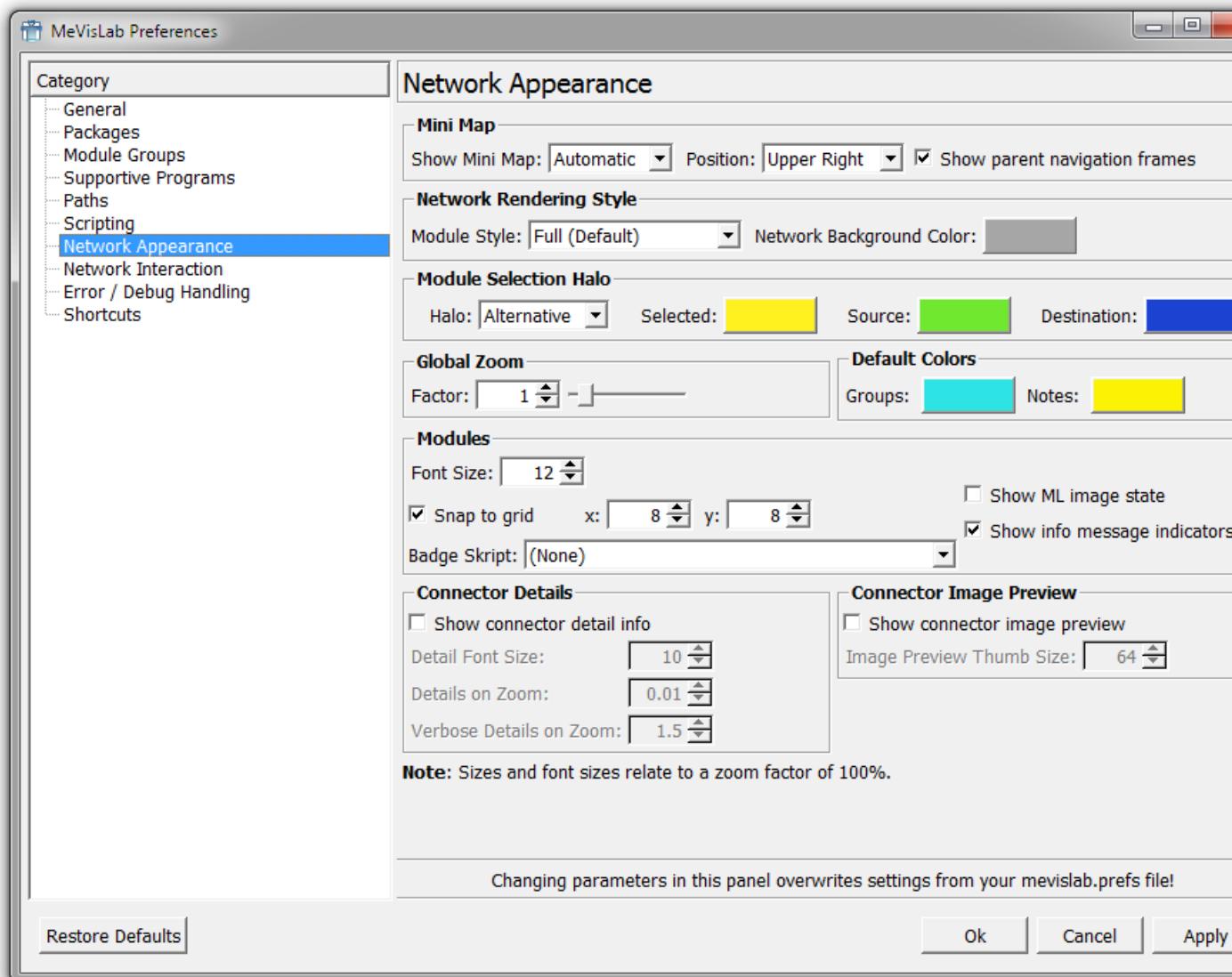
Note

The 'Additional Python Path' and the 'Python Debugger Code' are for attaching an external debugger to Python.

Using an external Python debugger is obsolete since MATE has a built-in debugger for Python (see [Section 26.8, "Python Debugger"](#)).

4.3.7. Preferences — Network Appearance

Figure 4.12. Preferences — Network Appearance



Mini Map

Show Mini Map

The Mini Map is an overview map. See also [Section 3.13, "Using the Mini Map"](#).

The following settings for the Mini Map are available:

Show Mini Map

- **Automatic:** Is displayed when parts of the network are outside the workspace.
- **Never:** Is never displayed.
- **Always:** Is always displayed.

Position

Defines the position of the Mini Map: **Upper Right** (default), **Lower Right**, **Upper Left**, **Lower Left**.

Show parent navigation frames

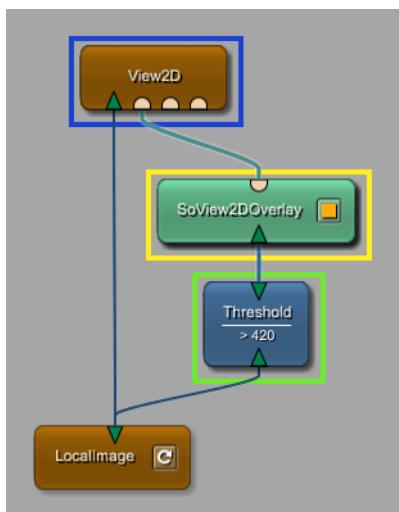
(For macro modules only) Shows the hierarchy of the opened networks. For example, when adding a View2D module, opening its internal network and there the internal network of View2DExtensions, the hierarchy of involved macro modules will be displayed in small frames next to the usual mini map (which might not be shown, depending on its mode).

Network Rendering Style

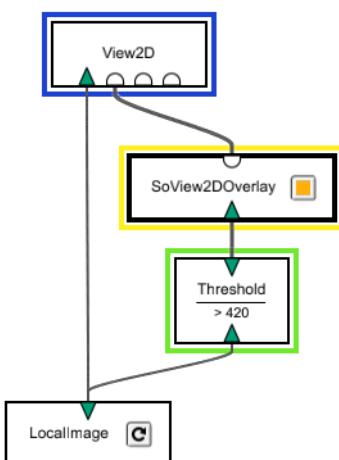
Style

Offers four options for styling the network rendering:

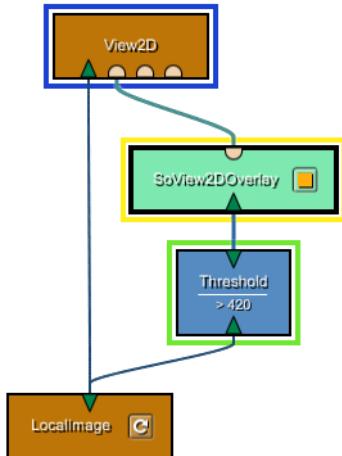
- **Full (Default):** Network is rendered in color with 3D and highlighting effects.



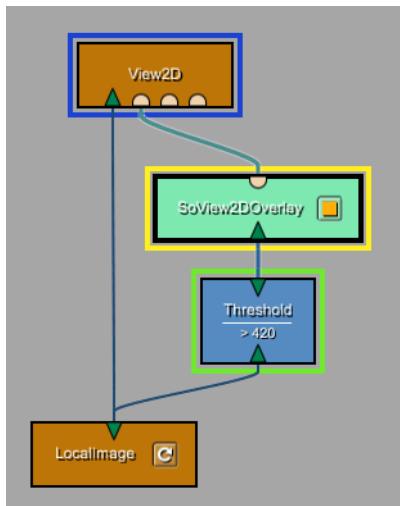
- **Print (Black & White):** Network is rendered as simple black-and-white drawing.



- **Print (Color)**: Network is rendered as a simple color drawing without effects and a white background.



- **Comic**: Network is rendered as a simple color drawing without effects but with a gray background.



All four styles are fully functional in terms of editing, connecting modules, displaying a Mini Map, etc. However, it is recommended to use the print styles only when screencapturing and printing the network.

Halo

Offers two options for styling the halo of highlighted modules:

- **Classic (Default)**: The halo is rendered as classic halo effect.
- **Alternative**: The halo is rendered as rectangle around the module.

To change the halo colors of the selected modules and the modules attached to them, edit the settings of **Selected**, **Source**, and **Destination** by clicking on the respective color field.

Global Zoom

Factor

Sets a global zoom factor. Only applicable to modules and networks that do not fill the network space automatically. Other modules/networks will be displayed with the given global zoom factor upon double-clicking the networks space or using the button (Show the whole network) in the toolbar.

Default Colors

Groups

Sets the background color for module groups, see [Section 3.11, “Using Groups”](#). The default is green.

Notes

Sets the background color for notes, see [Section 3.12, “Using Notes”](#). The default is yellow.

Modules

Font Size

Sets the font size of the module name in the display (number referring to a zoom of 100%).

Snap to grid

Sets the grid size in the workspace to which the modules snap when moved.

Show ML image state

Shows the image states by coloring the connectors.

- **Green:** Valid and updated ML image.
- **Yellow:** Valid but not updated ML image.
- **Red:** Invalid ML image.

Badge Script

Select a script that provides a badge icon for modules. Badge icons are intended to make users aware of certain properties of a module. See [Section 4.3.7.1, “How to define your own badge scripts”](#) for how to define your own badge scripts.

Connector Details

Show connector detail info

Enables detailed information for ML image, Inventor, or Base objects currently available on the connectors when a single module is selected.

Detail Font Size

Sets the font size for the connector detail info.

Details On Zoom

Sets the threshold zoom factor below which the details are not displayed.

Verbose Details On Zoom

Sets the threshold zoom factor below which the verbose details are not displayed.

Connector Image Preview

Show Connector Image Preview

Shows an image preview for an image connector when a single module is selected.

Image Preview Thumbnail Size

Sets the size of the image preview thumbnails in pixel.



Note

All sizes and fonts relate to a zoom factor of 100%.

4.3.7.1. How to define your own badge scripts

A badge script consists of a single `.py` Python file and a `.def` file (which just points to the Python file and declares a title for display in a combo box).

The script is treated like an invisible macro module. For each module type the `run()` function of the script is called with the module's type as string argument. The run function may return a dictionary with the entries "icon", which should be the path of an icon (preferably `.png`), "description", which is a text that is displayed in the tooltip of the module, and possibly "icon_size", if the icon must be scaled down for network display - this should be something like "20x20". When no icon should be displayed for a module, `None` can be returned.

Example MyBadgeScript.def file:

```
BadgeScript {
    title = "Some example badge script"
    source = "${LOCAL}/MyBadgeScript.py"
}
```

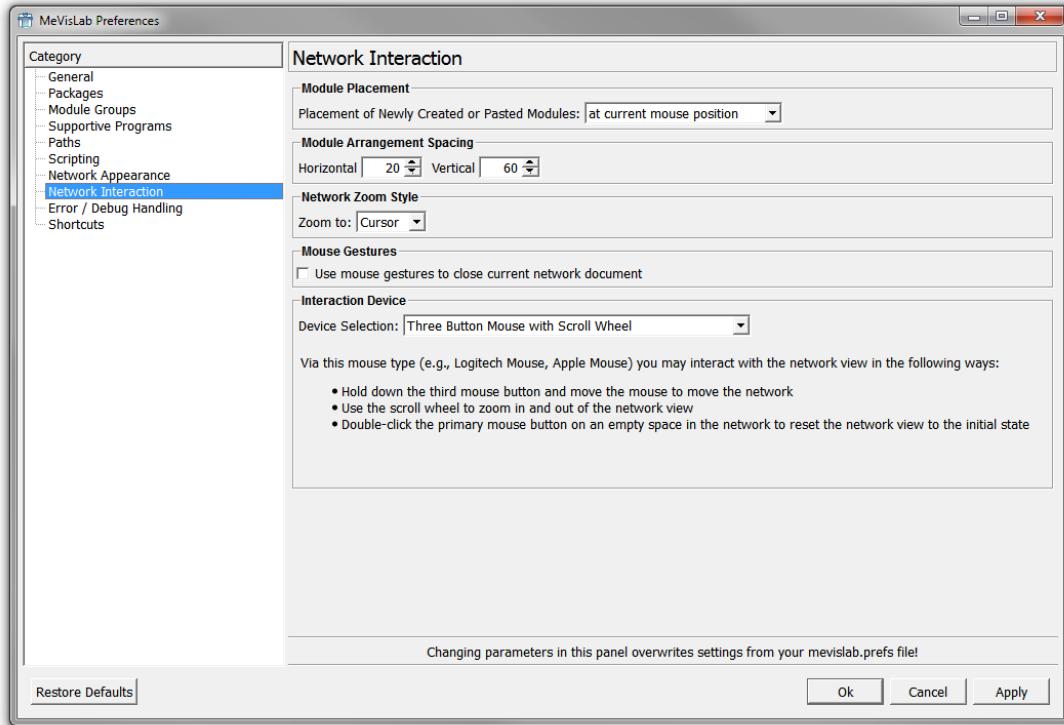
Example MyBadgeScript.py file:

```
from mevis import *

def run(moduleName):
    author = MLAB.moduleInfo(moduleName).get("author")
    if author and "John" in author:
        return {
            "icon": "${LOCAL}/johns.png",
            "description": "Module written by a John"
        }
    return None
```

4.3.8. Preferences — Network Interaction

Figure 4.13. Preferences — Network Interaction



Module Placement

Placement of newly created or pasted modules

Defines where newly created or pasted modules are placed in the network:

- **into the middle of a network:** Modules are inserted in the middle of the network space; multiple modules are inserted with a slight offset in position.
- **at current mouse position (Default):** Modules are inserted at mouse position; multiple modules are inserted in a cascading manner. If the mouse cursor is outside of the network window, the modules are placed into the middle of the network.

Module Arrangement Spacing

Horizontal / Vertical

Enter arbitrary values to adjust the distances between modules for the autoArrange scripting command. (The settings do not directly translate to pixels but depend on the zoom and other factors.)

Network Zoom Style

Zoom to

Defines to which position the zoom should be centered around. Available options are to zoom to the center of the network or to zoom to the current cursor location.

Network Mouse Gestures

Use mouse gestures to close current network document

Enables or disables the use of mouse gestures (see [Section 3.17, “Network Mouse Gestures”](#)).

This option only has an effect on Windows systems because on Linux and Mac OS X, the context menus are opened on right-click and not on right-button release.

Interaction Device

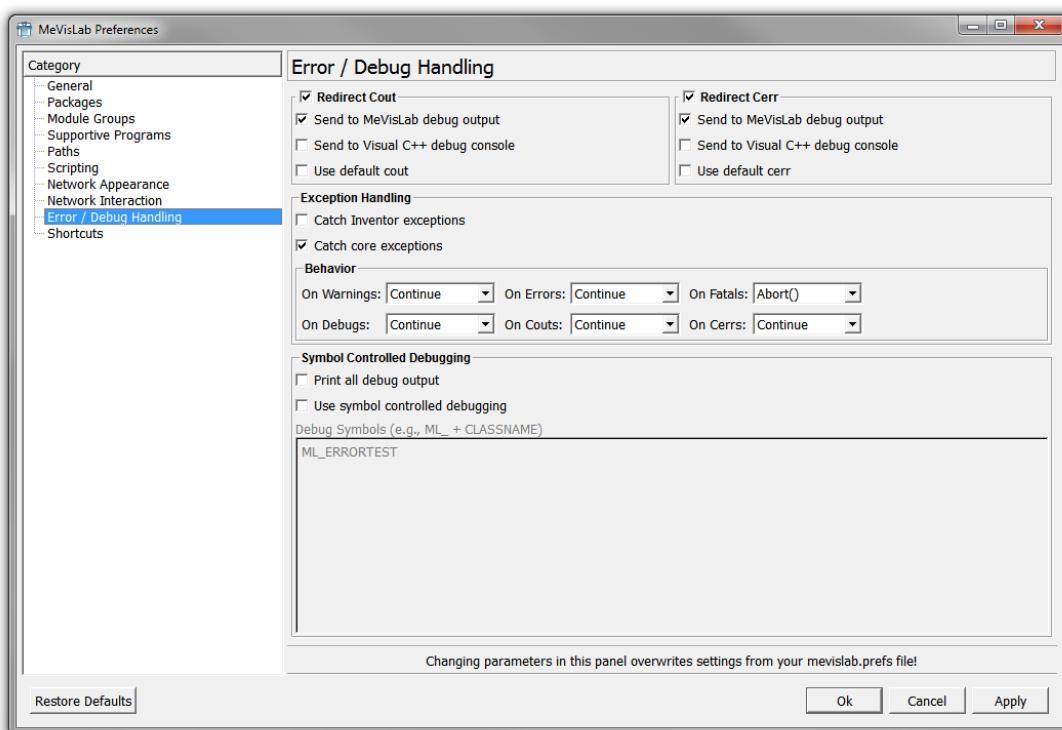
Device Selection

Allows selecting special interaction devices (mice, touch pads, especially Apple devices). For each selected option, the available features are listed beneath.

- **Three button mouse with scroll wheel** (Default)
- **Two button mouse with scrolling in multiple directions**, for example Apple Magic Mouse
- **Multi-touch pad**, for example Apple MacBook Touchpad

4.3.9. Preferences — Error / Debug Handling

Figure 4.14. Preferences — Error / Debug Handling



`cout` (standard output stream) and `cerr` (standard output stream for errors) are standard outputs in C++. The difference is that `std::cout` is a buffered stream especially useful for general output, while `std::cerr` is not buffered and therefore especially useful for error messages. The outputs are independent of each other, so both can be directed to other targets.

Redirect cout

Enables the redirect of the `cout` stream.

Send to MeVisLab debug output

Redirects `cout` to the Debug Output (default).

Send to Visual C++ debug console (Windows only)

(Only if MeVisLab is started from within Visual C++) Redirects `cout` to the Visual C++ Debug Console (on Windows).

Use default cout

Redirects `cout` to the default output (for example a console). Needs to be activated explicitly, otherwise only the redirect will be output.

Redirect cerr

Enables redirect of the `cerr` stream.

Send to MeVisLab debug output

Redirects `cerr` to the Debug Output. (default).

Send to Visual C++ debug console (Windows only)

(Only if MeVisLab is started from within Visual C++) Redirects `cerr` to the Visual C++ debug console.

Use default cerr

Redirects `cerr` to the default output (for example a console). Needs to be activated explicitly, otherwise only the redirect will be output.

Exception Handling

Catch inventor exceptions and **Catches core exceptions**

Exceptions are handled as selected for each kind of exception:

On Warnings, On Errors, etc.

Defines what is to happen upon warnings, errors, etc. Especially helpful if no source code for real tracing is available. Possible settings: **Continue**, **Abort**, **Exit(0)**, **Exit(ErrCode)**.

Symbol Controlled Debugging

Print all debug output

Prints out all debug outputs from the code (for example everything tagged with `ML_DEBUG`).

Use symbol controlled debugging

The debugging is done based on symbols, that is special classes. Enter the debug symbols to filter for in the text field, for example `ML_ERRORTEST`.

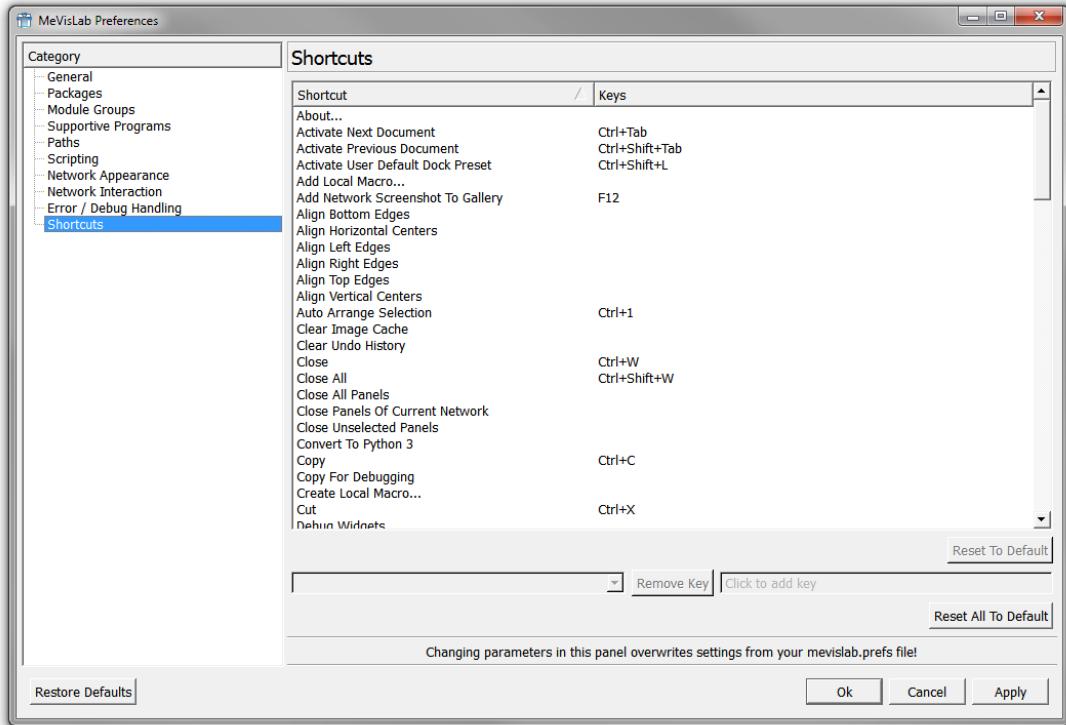


Note

Refer to the ML Guide for detailed information on symbol-controlled debugging.

4.3.10. Preferences — Shortcuts

Figure 4.15. Preferences — Shortcuts



The shortcut editor allows to set and/or change shortcuts for various actions. It also shows which shortcuts are active at the moment.

To remove a shortcut for an action, select the action, select its shortcut key (if there are multiple), and press **Remove Key**.

To set a keyboard shortcut, select the action and click into the field that says **Click to add key**. Then, press the key or key combination you want to use for this action. This can be done multiple times, adding a shortcut for the action at each repetition.



Note

The same shortcut editor is available in MATE.

Reset To Default

Resets the keyboard shortcut for the selected action to its default.

Remove Key

Removes the currently shown shortcut key for the selected action.

Reset All To Default

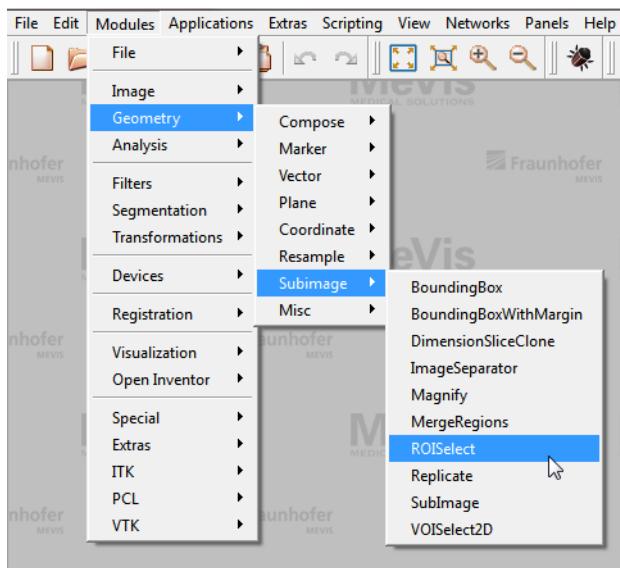
Resets all keyboard shortcuts to their defaults.

4.4. Modules Menu

Displays a tree of all modules currently available in the MeVisLab module database, sorted both by genres and by DLLs (projects). Includes all MeVisLab SDK modules and all user-defined modules.

In the genre section, the listing of a module is based on its genre and the pre-defined genre structure given in the `Genre.def` file, see the MDL Reference, chapter “Module Genre Definition”. This way, new modules are automatically displayed in the correct submenu.

Figure 4.16. Modules Menu



4.5. Applications Menu

Lists available applications. Applications are macro modules that have the genre tag “Applicationsmenu” in their definition file.

A typical example is ImageViewer. Start it as application from the menu or insert the corresponding macro module “ImageViewer” via the module search.

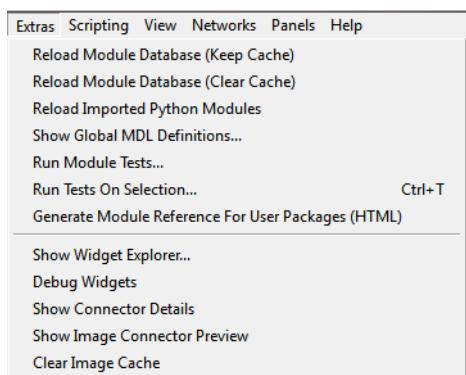


Note

Applications can only be created with the special ADK license.

4.6. Extras Menu

Figure 4.17. Extras Menu



4.6.1. Reload Module Database (Keep Cache)

Reloads the .def, .script, and .py files of modules that have been changed after the last reload.
Use this to

- add newly defined modules to the module database.
- update changes on module interfaces and scripting.

4.6.2. Reload Module Database (Clear Cache)

Reloads all modules in the database like [Section 4.6.1, “Reload Module Database \(Keep Cache\)”](#) but clears the cache.



Note

This may be slow, especially if many module panels are currently open in the network(s). To update current changes on module interfaces and scripting faster, use [Section 4.6.1, “Reload Module Database \(Keep Cache\)”](#).

4.6.3. Reload Imported Python Modules

This feature is only relevant when using the Python import functionality and working on the imported modules.

Reloads imported Python modules (not MeVisLab modules that use Python). This was previously only possible via a manual reload() call or a MeVisLab restart.

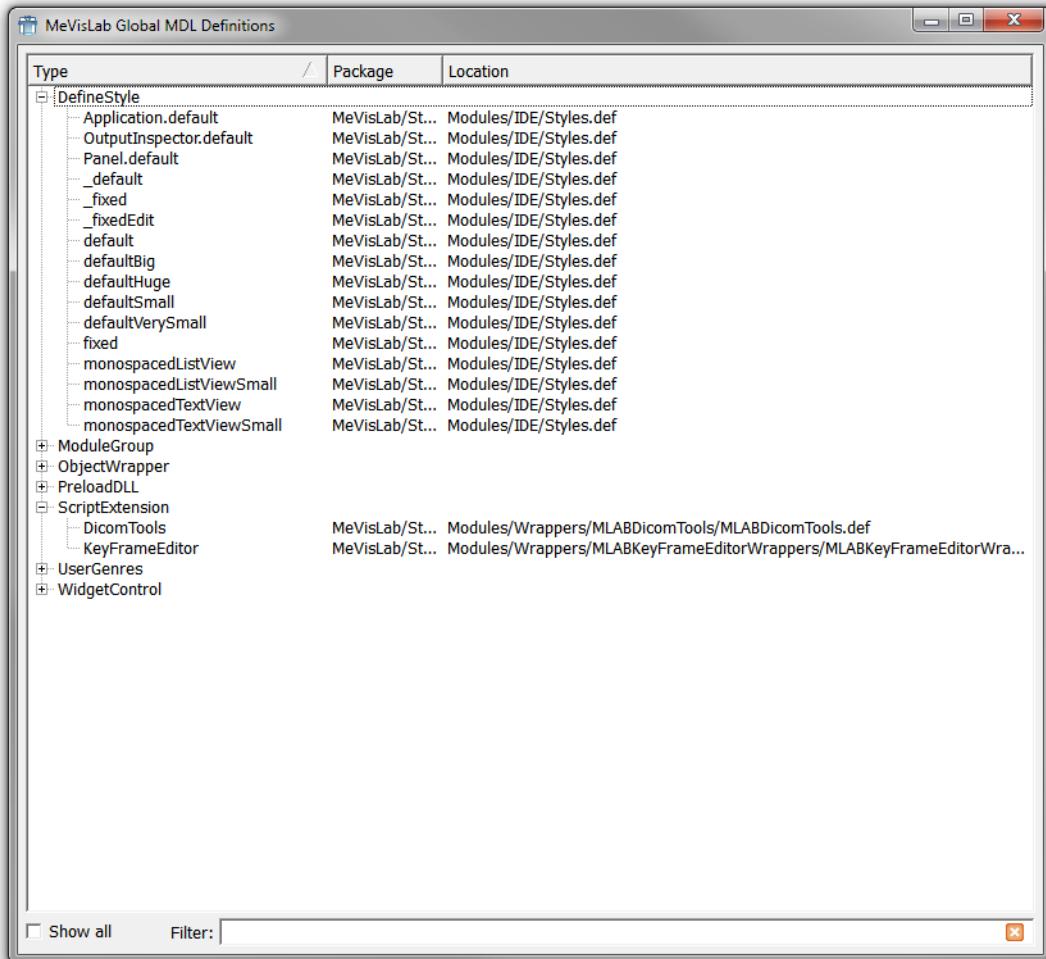


Note

After the Python modules have been reloaded, reload the MeVisLab modules that make use of the Python module(s). Otherwise the MeVisLab modules will still see the previously imported Python modules.

4.6.4. Show Global MDL Definitions...

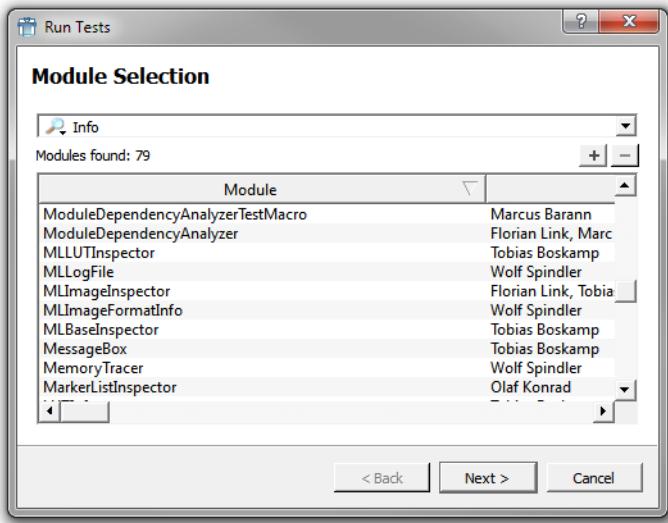
This entry displays a list of special objects defined in the MDL, sorted by object category. This is useful to, e.g., find special control types that might not be listed in the MDL Reference.

Figure 4.18. MeVisLab Global MDL Definitions

By default the list only contains the most useful object categories. Checking the "Show All" option shows (almost) all object categories, but most of them are not immediately useful since they contain MeVisLab internals.

4.6.5. Run Module Tests...

Starts the TestCenter for a module selection, by default for the modules selected in the *Module Search* browser window (see [Chapter 13, Module Search](#)). By changing the filter settings in the extra window, other modules can be selected for testing. For all modules, the test cases associated with the module(s) are listed here. In case of `TestWebView`, only the generic test case “Formal” that is associated with all modules are available. When **Finish** is clicked, the test cases are run and test reports are available.

Figure 4.19. Module Selection**Tip**

To run tests on several modules in a network, select them, open the network context menu and select **Run Tests On Selection...**; or use the menu entry of the same name in the Extras menu. For single modules, start the tests via the module context menu.

For further information, see the TestCenter Reference and the Getting Started, chapter 16, “Using the TestCenter”.

4.6.6. Run Tests On Selection...

Selects the tests associated with the currently selected modules and shows a dialog from which these tests can be started.

4.6.7. Generate Module Reference for User Packages (HTML)

Creates an HTML index for the help files of modules in the user packages (one index for each PackageGroup).

4.6.8. Show Widget Explorer

The **Widget Explorer** is useful for developing Qt style sheets. It can also be used for debugging module panels.

On the left side are

- the **Widget** view, which displays all existing windows and widgets of the MeVisLab process hierarchically (see [Figure 4.20, “MeVisLab Widget Explorer - Attributes Inspector”](#))
- the **Update** button, which can be clicked to update the view when the GUI changes and new windows are shown
- the **CSS Selector** area, which shows the class hierarchy of the currently selected widget (this can be useful for writing CSS rules)
- the **Highlight Selected Widget** checkbox, which toggles if the background color of the currently selected widget is temporarily changed to yellow to ease locating the widget in the GUI (note that this does not work for all widgets, because not all draw their background themselves)

On the right side are

- the **Attributes** inspector, which shows the widget attributes
- the **StyleSheet** editor, which allows for viewing and testing style sheet rules (see [Figure 4.21, "MeVisLab Widget Explorer - Style Sheet Editor"](#))

Figure 4.20. MeVisLab Widget Explorer - Attributes Inspector

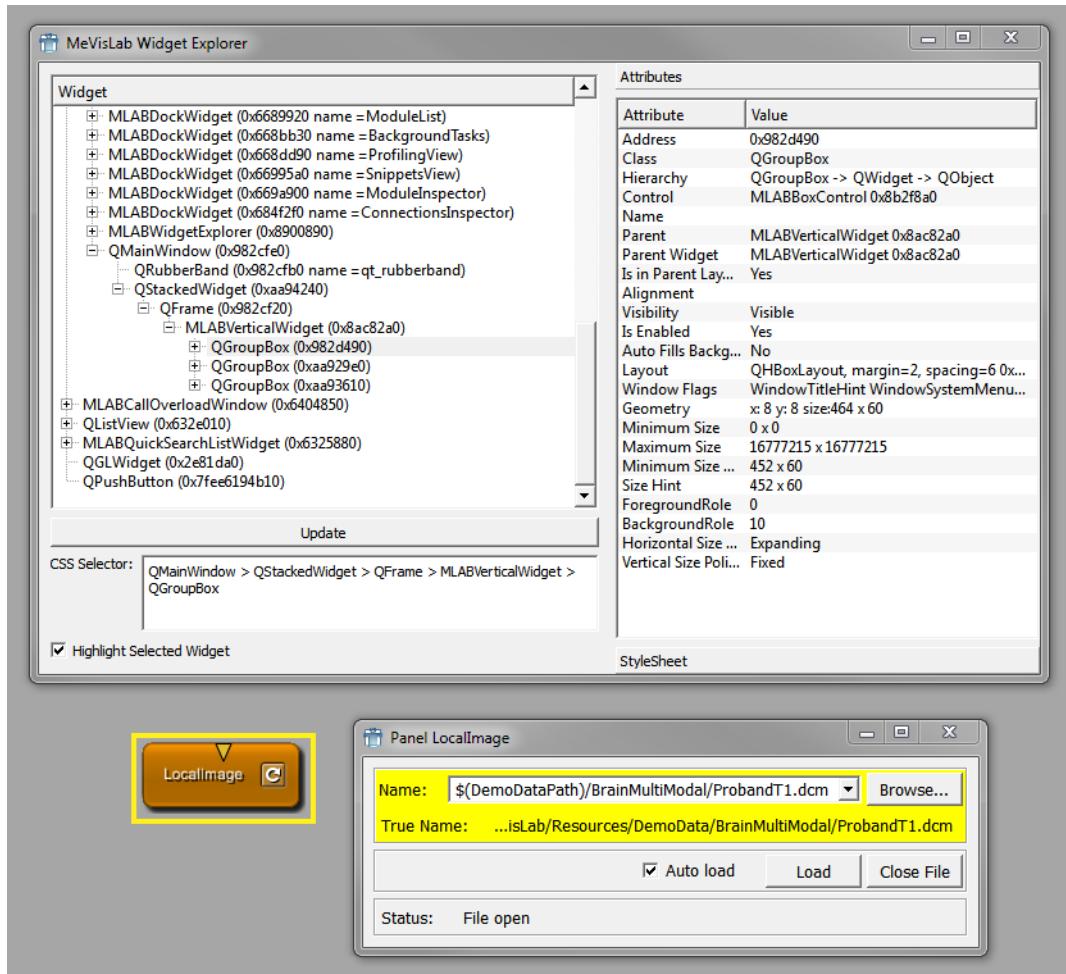
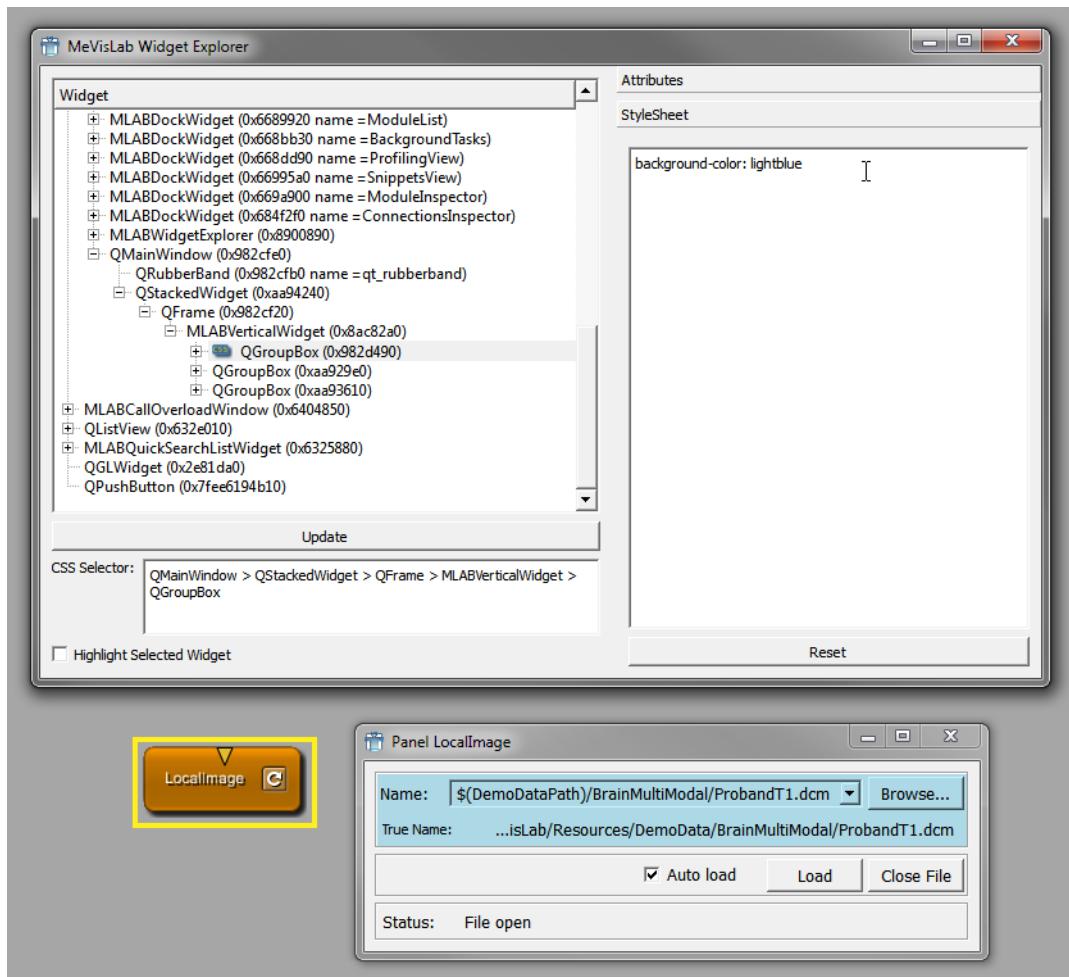


Figure 4.21. MeVisLab Widget Explorer - Style Sheet Editor



4.6.9. Debug Widgets

Enables/disables debugging module panels. **CTRL+left-clicking** a user interface control in a module panel opens the `.script` file (MDL source code of the GUI) in MATE at the line where this GUI control is defined.



Note

To be able to jump to the definition line of the user interface control with any other text editor than MATE, the parameter `%f(%1)` has to be set in the Preferences, see [Section 4.3.4, “Preferences — Supportive Programs”](#).

4.6.10. Show Connector Details

Shows detailed info about image, Inventor or base object properties currently pending on module's connectors. Activated when selecting a single module in the network. Same option as **Connector Detail Info** in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#)

4.6.11. Show Image Connector Preview

Shows image preview on a module's image connectors when selecting a single module in the network. Same option as **Connector Image Preview** in the Preferences, see [Section 4.3.7, “Preferences — Network Appearance”](#).

4.6.12. Clear Image Cache

Frees cached image pages of all ML modules in currently opened networks. All image pages that are currently not in request in any module pipeline are cleared and have to be recalculated the next time they are requested. See the ML Guide for details.

4.7. Scripting Menu

Offers features for script editing and for running user scripts.

For details on user scripts, see [Section 4.8, “User Scripts”](#)

Scripting is used to implement the dynamic functionality of module user interfaces and applications (which are defined as macro modules) in MeVisLab.

Scripting is done in the context of a module. Via the script context variable of the module (`ctx`), access is given to the module instance itself as well as to all members of the module like fields, input/outputs, and GUI controls. This way, getting/setting field values (module parameters), connecting/disconnecting fields, implementing dynamic user interfaces, and much more can be done dynamically .

In the context of an `MLABMacroModule`, access to the contexts of all modules contained in the modules macro network is available (recursive descend).

For scripting in MeVisLab one can use the Python language, via an object-binding with `PythonQt`, an inhouse-development by MeVis.

This object-binding uses the Qt Meta Object System to find out about the `MLAB` object features.

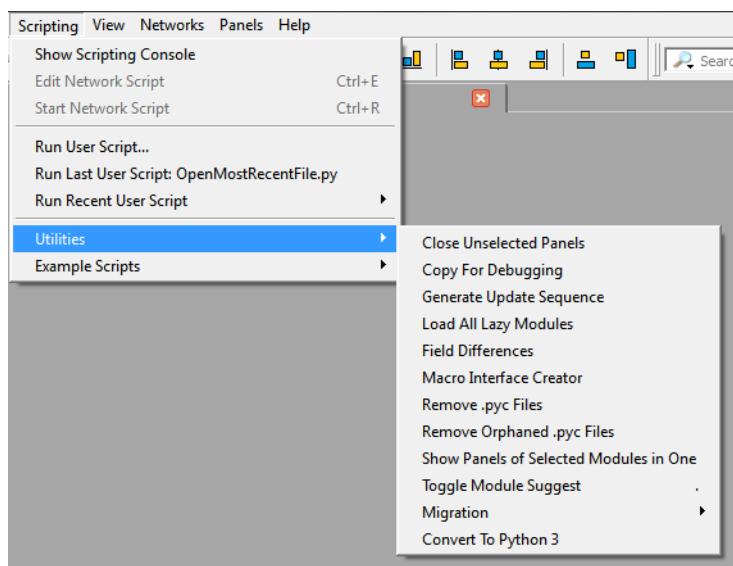
For the doxygen documentation of the scripting interface, see the MeVisLab Scripting Reference.



Tip

It is possible (but not recommended) to include single line script statements in MDL script files.

Figure 4.22. Scripting Menu



4.7.1. Show Scripting Console

Opens a command line console to type in Python code. Useful for debugging when programming user interfaces dynamically. The console is opened in the context of the current network. Access is possible to

- the contexts of all modules contained in the current network
- (for macro modules) to all objects of included modules in a recursive descend



Note

If the current network is a macro network, the script console is opened in the context of the macro module. The same context is reached when opening the script console via the macro module's context menu, **Debugging** → **Show Script Console**.

Figure 4.23. Scripting Editor



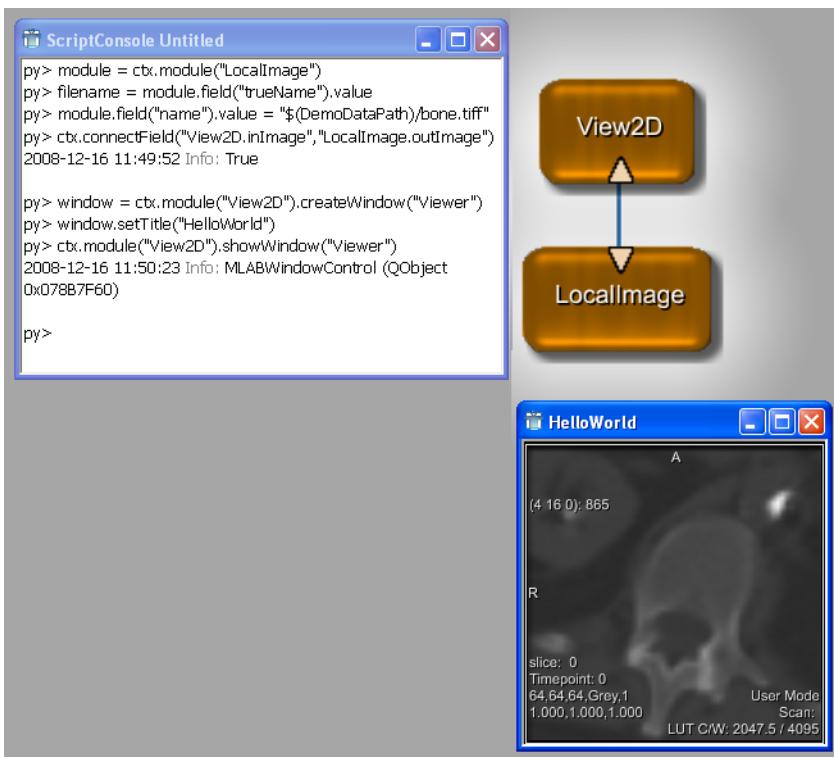
Example listing in Python. The code has to be entered one line at a time, and without any indents. Make sure that the modules LocalImage and View2D exist in the current network, and that the modules have those specific instance names:

```
# get module context
module = ctx.module("LocalImage")

# set/get module field value
filename = module.field("trueName").value
module.field("name").value = "${DemoDataPath}/bone.tiff"

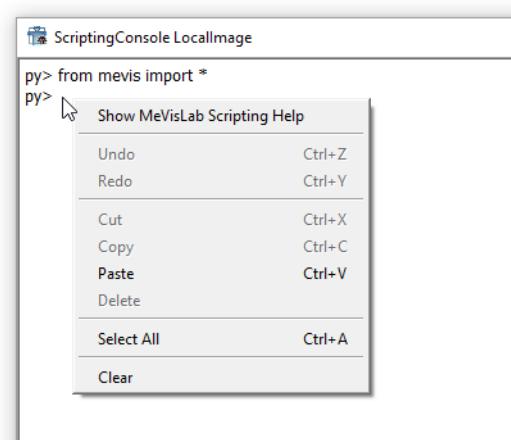
# connect fields
ctx.connectField("View2D.inImage", "LocalImage.outImage")

# open GUI window control
window = ctx.module("View2D").createWindow("Viewer")
window.setTitle("HelloWorld")
ctx.module("View2D").showWindow("Viewer")
```

Figure 4.24. Scripting Example

4.7.2. Scripting Context Menu

The context menu of the Scripting View contains the usual commands for text editing, see also [Section 4.2, “Edit Menu”](#).

Figure 4.25. Scripting Context Menu

In addition, three options are available:

Show MeVisLab Scripting Help

Opens the Scripting Reference documentation (HTML) in the default web browser.

Scripting Language

Can be switched from the default (as defined in the Preferences) to the other script language. We recommend to stay with Python (see note above).

Clear

Clears the scripting console.

4.7.3. Edit Network Script

(Macro modules only) Opens the interface definition file (.script or .def) in the default text editor.

4.7.4. Start Network Script

(Macro modules only) Parses the interface definition file (.script or .def). Has the same effect as adding the macro module to a network and opening the module panel.

4.8. User Scripts

A user script is either a single .py Python file or an MDL .script file (with a .py and .mlab file, like any local MacroModule).

When a user script is started, MeVisLab creates a local macro module from the script and calls the `run()` function, which needs to be declared in the Python script. The run function takes a single argument, which is the context (macro module) that surrounds the currently active MeVisLab network.

Using this argument, the script can work on the active network, get the selected modules, etc.

For instance, the following script just prints the selected modules to the console:

```
def run(macro):
    print macro.network().selectedModules()
```

A user script can even show MDL windows from the `run()` method, which are declared in the .script file of the user script. The user script is destroyed after the last window it creates is closed.

User scripts are added to the **Scripting** menu by defining `Action` and `SubMenu` entries in a `UserIDE_menus` section in a user script's definition file. The user script's definition file can have any name, but it must have the extension `.def` and it needs to be placed below the `Modules` directory of any package, because MeVisLab scans only the `Modules` directories recursively for the `*.def` files. It is preferred to place the user scripts into a folder named `UserScripts` below the `Modules` directory of your package.

Please have a look at the user script definition file `MeVisLab/IDE/Modules/IDE/UserScripts.def` and the example user scripts in `MeVisLab/IDE/Modules/IDE/UserScripts/`.

4.8.1. Example Scripts

- **Replace Inventor Group**

Replaces a single selected Inventor group node (e.g., `SoSeparator`, `SoGroup`) by specifiable Inventor group. All connections to the original Inventor group node are restored to the new Inventor group node.

4.8.2. Run User Script...

Opens a browse dialog to load and run a user script. By default, the folder `MeVisLab/IDE/Modules/IDE/UserScripts` is opened, where a number of commonly used scripts are located.

4.8.3. Run Last User Script:

Runs the last chosen user script. This menu entry is only active if a user script has been executed before.

4.8.4. Run Recent User Script

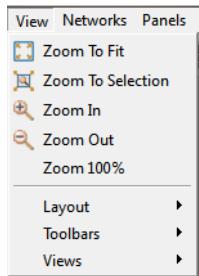
Offers a list of recently executed user scripts to choose from.

4.8.5. Example Scripts

Offers a list of pre-installed example user scripts.

4.9. View Menu

Figure 4.26. View Menu



4.9.1. View All

Displays the complete network.

4.9.2. Zoom To Selection

Zooms the selection to 100%.

4.9.3. Zoom In

Zooms in for displaying more network details.

4.9.4. Zoom Out

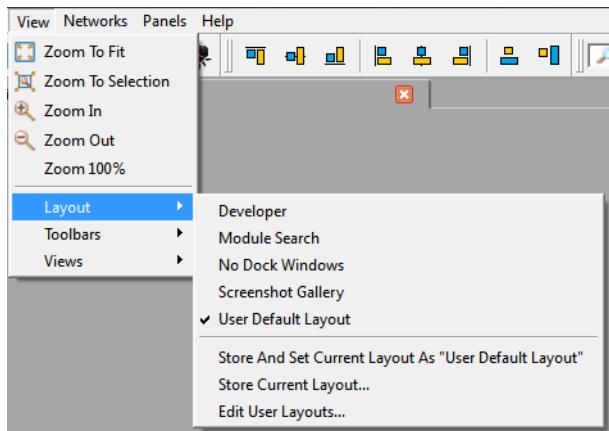
Zooms out for displaying less details and more of the full network.

4.9.5. Zoom 100%

Zooms the network to 100% (size based on the standard module design).

4.9.6. Layout

Figure 4.27. View — Layout Submenu



In this menu, the MeVisLab interface layout concerning the visible Views and their arrangement is defined, see [Section 4.9.8, “Views”](#).



Tip

The Layout menu is also available from the bottom bar, see [Chapter 6, Bottom Bar](#).

The following pre-defined settings are available:

Developer

Opens Output Inspector, Module Inspector, Module List, and Debug Output.

Module Search

Opens Module Inspector, Module List, and Module Search.

No Dock Windows

Hides all docking windows, that is all Views. This leaves only the network workspace visible.

Screenshot Gallery

Opens the Screenshot Gallery, see [Chapter 19, Screenshot Gallery](#).

User Default Layout

Opens the layout saved as “User Default Layout”. If none was saved, opens the last used layout.

The following options for layout handling are available:

Store and Set Current Layout as "User Default Layout"

Saves the current configuration of Views as the user default and activate this user default layout.



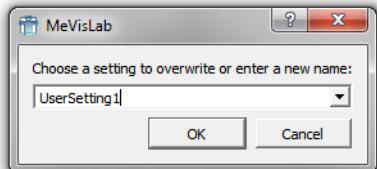
Note

The default user layout is a persistent setting. If it is the currently active layout, the configuration of Views last changed by the user is saved as “Default User Layout”. Therefore, changes done on the layout will “overwrite” the default user layout.

Store Current Layout

Opens a window to save the current configuration of the Views under another name. Stored user layouts are not overwritten when updating/reinstalling MeVisLab but are saved to the places listed in the Preferences chapter per operating system, see [Section 4.3, “Preferences”](#).

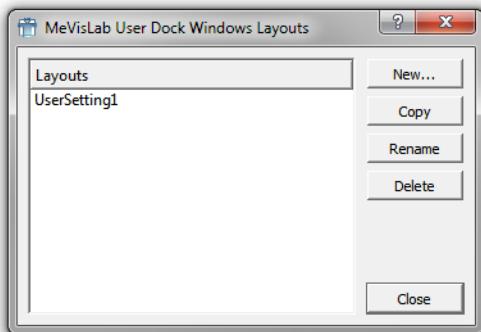
Figure 4.28. Store Current Layout



Edit User Layouts

Opens a window to copy, rename or delete saved user layouts.

Figure 4.29. Edit User Layouts

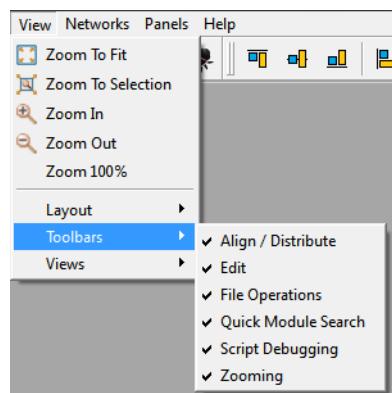


User layouts cannot be edited via the menu. For editing, open the user layout in MeVisLab, edit its Views configuration and save it under its old name.

4.9.7. Toolbars

In the Toolbars menu, toolbar elements can be enabled and disabled.

Figure 4.30. View — Toolbars Submenu



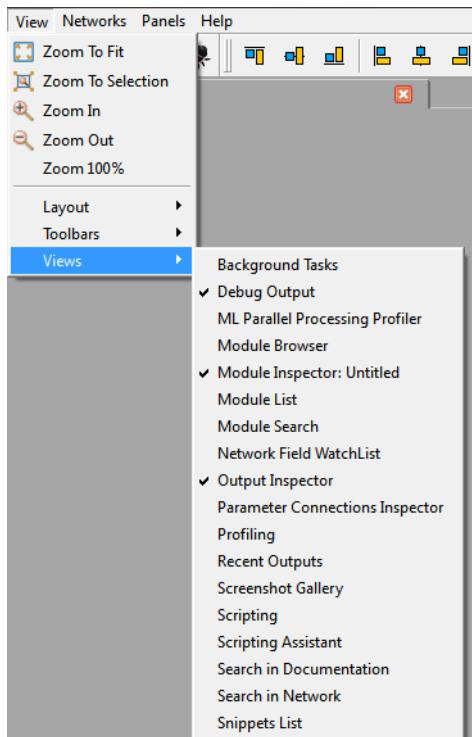
- **Align / Distribute:** see [Section 4.2.12, “Align / Distribute”](#).
- **Edit:** see [Section 5.2, “Edit”](#).

- **File Operations:** see [Section 5.1, “File Operations”](#).
- **Quick Module Search:** see [Section 5.5, “Quick Search”](#).
- **Script Debugging:** see [Section 26.8, “Python Debugger”](#).
- **Zooming:** see [Section 5.3, “Zooming”](#).

4.9.8. Views

In the Views menu, Views elements can be enabled and disabled.

Figure 4.31. View — Views Submenu



Available Views:

- **Background Tasks:** see [Chapter 7, *Background Tasks*](#).
- **Debug Output:** see [Chapter 8, *Debug Output*](#).
- **ML Parallel Processing Profiler View:** see [Chapter 9, *ML Parallel Processing Profiler View*](#).
- **Module Browser:** see [Chapter 10, *Module Browser*](#).
- **Module Inspector:** see [Chapter 11, *Module Inspector*](#).
- **Module List:** see [Chapter 12, *Module List*](#).
- **Module Search:** see [Chapter 13, *Module Search*](#).
- **Network Field WatchList:** see [Chapter 14, *Network Field WatchList*](#).
- **Output Inspector:** see [Chapter 15, *Output Inspector*](#).
- **Parameter Connections Inspector:** see [Chapter 16, *Parameter Connections Inspector*](#).
- **Profiling:** see [Chapter 17, *Profiling*](#).

- **Recent Outputs:** see [Chapter 18, Recent Outputs](#).
- **Screenshot Gallery:** see [Chapter 19, Screenshot Gallery](#).
- **Scripting Console:** see [Chapter 20, Scripting Console](#).
- **Scripting Assistant:** see [Chapter 21, Scripting Assistant](#).
- **Search in Documentation:** see [Chapter 23, Search in Documentation](#).
- **Search in Network:** see [Chapter 22, Search in Network](#).
- **Snippets List:** see [Chapter 24, Snippets List](#).

4.10. Networks Menu

In the Networks menu, functions for closing the current or all networks are available, as well as a list of all open networks and their status.

Figure 4.32. Networks Menu



At the bottom of the menu, a list of all currently open networks is displayed. Networks with unsaved changes are marked with * (asterisk).

4.10.1. Close

Closes the current network. In case of unsaved changes, a message appears.

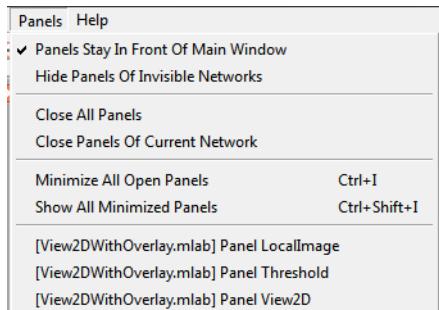
4.10.2. Close All

Closes all networks. In case of unsaved changes, a message appears.

4.11. Panels Menu

Manages all module panels currently opened in any of the open networks.

Figure 4.33. Panels Menu



If networks with open panels are available in the workspace, the panels are listed at the bottom of the Panels menu. The panels are named in the pattern **[NetworkName] PanelName**. For not yet

saved networks (“untitled”), [-] is displayed as network name without a distinction for different unsaved networks.

For the behavior of the panel list, see [Section 4.11.7, “Working with the Panel List”](#).

4.11.1. Panels Stay In Front Of Main Window

If selected, panel windows are tied to the MeVisLab application window and always stay in front of it. Otherwise, panels are independent windows that may be used even if the MeVisLab application window is hidden or minimized.



Note

Independent panels may accidentally get hidden behind MeVisLab or other applications.

4.11.2. Hide Panels Of Invisible Networks

If selected, panels of networks currently not visible are hidden (which is not the same as minimized). Otherwise, open panels of all networks are visible.

4.11.3. Close All Panels

Closes all panels of all networks.

4.11.4. Close Panels Of Current Network

Closes all panels of modules of the current network only.

4.11.5. Minimize All Open Panels

Minimizes all open panels at once. Can be combined with the option [Section 4.11.2, “Hide Panels Of Invisible Networks”](#).

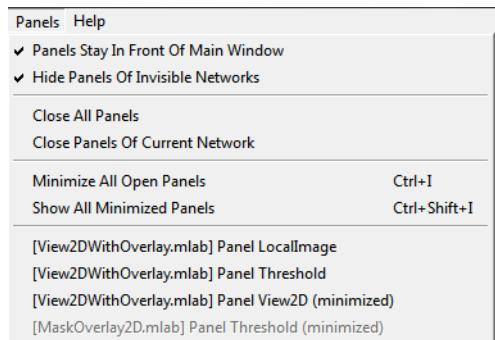
4.11.6. Show All Minimized Panels

Restores all open panels to the state they have been before minimizing. Can be combined with the option [Section 4.11.2, “Hide Panels Of Invisible Networks”](#).

4.11.7. Working with the Panel List

If networks with open panels are available in the workspace (and none are hidden), all open panels are displayed and listed at the bottom of the Panels menu.

Figure 4.34. Panels Menu — Listing all Open Panels



Usually, having all panels open will clutter the workspace. By combining the hiding and minimizing options, currently unnecessary panels can be “removed” temporarily from the workspace.

If **Hide Panels Of Invisible Networks** is selected, panels of currently not displayed networks are hidden and grayed out in the panel list. They cannot be selected directly. Only if their network is visible again, the hidden panels are displayed at their original size and position and can be selected in the list.

Panels can be minimized by selecting the **Minimize All Open Panels** option or by clicking the minimize button at the top right of a single panel window.



Note

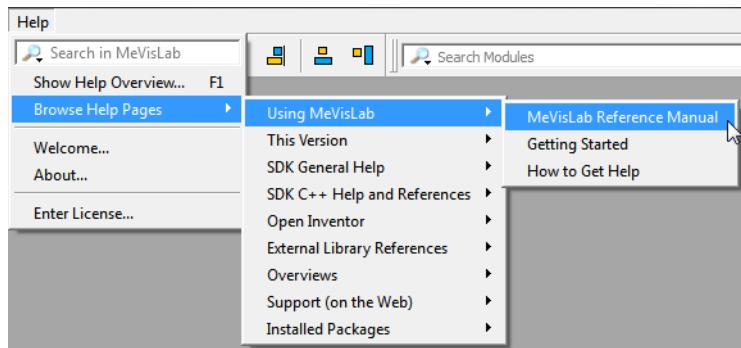
If **Hide Panels Of Invisible Networks** is selected, **Minimize All Open Panels** has no effect on any hidden panels of invisible networks. This way, you can selectively minimize the panels of only the currently visible network.

Minimized panels are tagged with “(minimized)” in the panel list. When selecting a minimized panel in the list, it is restored to its original size and position. Alternatively, select **Show All Minimized Panels** to restore all panels.

4.12. Help Menu

Offers important information and links around MeVisLab.

Figure 4.35. Help Menu



4.12.1. (Search in documentation and menu entries)

The input field in the help menu does an immediate search in all menu entries and the documentation of MeVisLab, see [Chapter 23, Search in Documentation](#).

4.12.2. Show Help Overview

Opens a HTML file with an overview over all MeVisLab help resources in the default browser.

4.12.3. Browse Help Pages

Allows browsing and opening a specific help page directly in the default browser.

4.12.4. Welcome

Opens the Welcome screen which is displayed when first starting MeVisLab. It offers a plethora of links to tutorials, documentation, demos, user forums, recently opened networks, latest information about MeVisLab, and a “Tips + Tricks” section.

Toggle the option **Don't show this at MeVisLab launch** to have this window displayed at start-up or not.

4.12.5. About

Displays a window with information about the installed version and license, involved developers, and license information about all built-in technologies.

4.12.6. Enter License

Opens a file browser to select a license file (.dat). Depends on your MeVisLab distribution (free, basic, SDK, etc.), see installation guide.

Chapter 5. Toolbar

The toolbar below the menu bar in the GUI offers some editing and zooming features, a quick search, and the align/distribute feature.

Figure 5.1. Toolbar



All offered toolbar groups can be enabled and disabled via **View → Toolbars**.

5.1. File Operations

See [Section 4.1, “File Menu”](#).

5.2. Edit

See [Section 4.2, “Edit Menu”](#).

5.3. Zooming

See [Section 4.9, “View Menu”](#).

5.4. Script Debugging

This tool bar only contains one button showing a stylized bug. Clicking this button will start the MATE editor if it isn't open already, and activate or deactivate the integrated Python debugger.

5.5. Quick Search

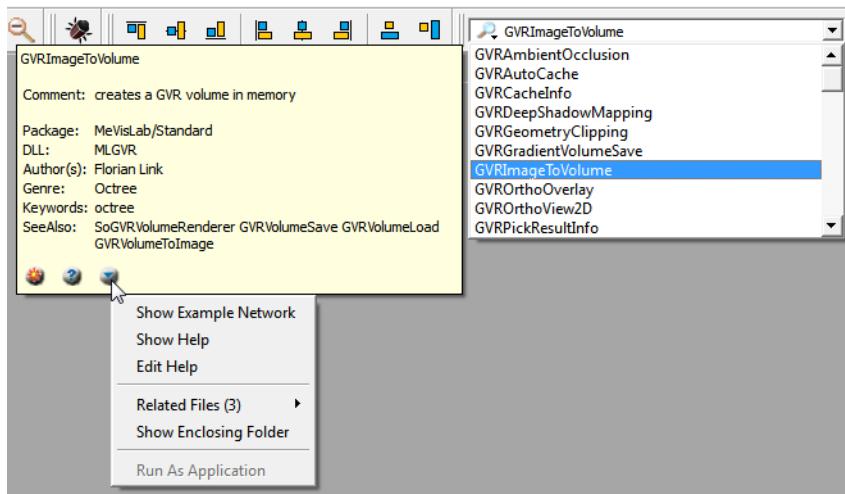
Searches modules by name. Click the magnifier button to select search criteria.

Figure 5.2. Quick Search Options

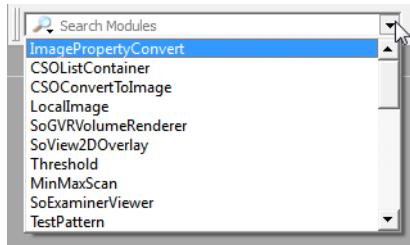


- **Substring:** If selected, extends the search to module name substrings. Effectively works as if adding wildcards, for example `*image*`.
- **Keywords:** If selected, extends search to module keywords (defined in the `<ModuleName>.def` file).
- **Match Case:** If selected, the search differentiates between lower and upper case letters.

Clicking on a quick search result opens an info box for the module. More buttons are available on the bottom, offering the functions “Create module”, “Show HTML help” and “More options”. The options are the same as for modules in the search results, see [Section 13.3, “Module Search Result Context Menu”](#).

Figure 5.3. Quick Search — Info Box

When opening the drop-down list without a search entry, it contains the search history with the latest added modules.

Figure 5.4. Quick Search History**Tip**

For more extended search possibilities, see [Chapter 13, Module Search](#).

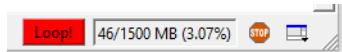
5.6. Align / Distribute

Allows aligning and distributing of selected modules in a network. See [Section 4.2.12, “Align / Distribute”](#).

Chapter 6. Bottom Bar

The bottom bar can be found at the lower right of the MeVisLab GUI. It offers two unique options and a quick way to the Layout menu.

Figure 6.1. Bottom Bar



6.1. Loop! indicator

This flashing indicator is only visible if a network with Inventor modules is causing constant field updates. This can happen because some viewer interaction makes constant updates necessary, but may also happen if Inventor field connections form a loop and at least one ML or macro module field is involved (because that foils the loop detection of Inventor). So if this indicator is visible without any interaction, you need to investigate your network, for example by using profiling, see [Section 17.2.2, “Fields”](#).

For a possible way to break notification loops see [Section 27.5, “Using SyncFloat to Reduce System Load”](#).

6.2. ML Cache

The ML Cache field displays the <used size / max. size> of the Image Processing Cache, and the percent of memory in use.

The Image Processing Cache Size in MB can be set in **Preferences → General → Resources**, see [Section 4.3.1, “Preferences — General”](#).

To clear the image cache of all ML modules in the currently opened networks, select **Extras → Clear Image Cache**, see [Section 4.6.12, “Clear Image Cache”](#).

These options can also be accessed from a context menu on this area.

6.3. Stop Button

The Stop button is used to terminate algorithms. This only works if the module is implemented with a stop button.

An explanation and example code for the implementation of the Stop button can be found in the MLGuide, chapter “Testing For Interruptions During Calculations”.

6.4. Toggle Layout

Opens the layout menu, see [Section 4.9.6, “Layout”](#).

Chapter 7. Background Tasks

The **Background Tasks** View displays running background tasks.

ML Background Tasks is a framework to execute long taking tasks in worker threads. The basic framework is independent of the ML, but convenience classes add access to ML images. For more on background tasks, see the ToolBox Reference, chapter “Background Tasks”.

Figure 7.1. ML Background Tasks

Status	Module	Task	Progress
Finished	GVRVolumeSav...	Saving...	100%
Canceled	GVRVolumeSav...	Saving...	34%
Finished	ImageSave(Ima...	Saving image...	100%
Finished	WEMIsoSurface...		100%
Finished	WEMIsoSurface...		100%

Four states are possible:

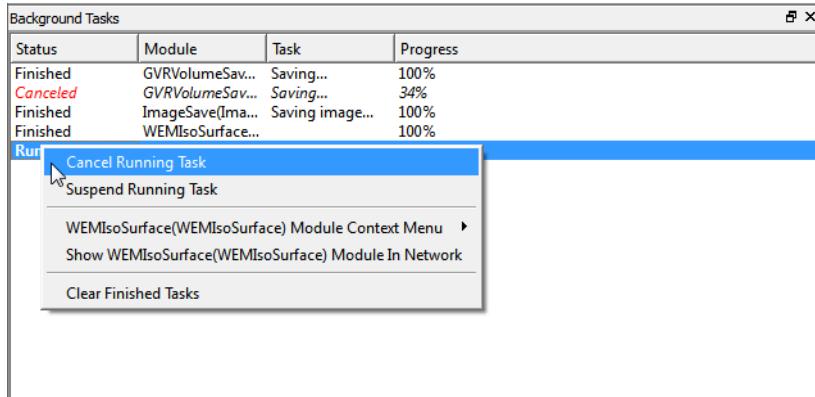
- Running
- Finished
- Canceled
- Suspended

Figure 7.2. ML Background Tasks — Context Menu

Status	Module	Task	Progress
Finished	GVRVolumeSav...	Saving...	100%
Canceled	GVRVolumeSav...	Saving...	34%
Finished	ImageSave(Ima...	Saving image...	100%
Finish	ImageSave(ImageSave)	Module Context Menu	>
Finish	ImageSave(ImageSave)	Show ImageSave(ImageSave) Module In Network	
		Clear Finished Tasks	

In following options are available in the context menu:

- **<ModuleName> Module Context Menu** : Opens the normal module context menu.
- **Show <ModuleName> Module In Network** : Selects the module and zooms to it.
- **Clear Finished Tasks**: Deletes all finished tasks from the list.

Figure 7.3. ML Background Tasks — Context Menu for Running Processes

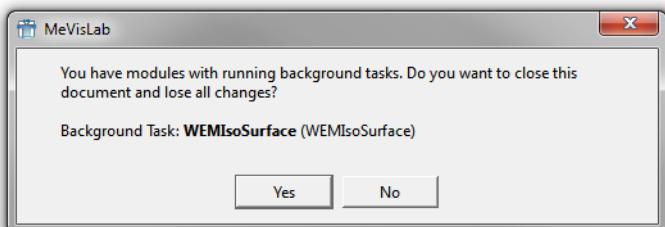
The following additional options are available for running processes:

- **Cancel Running Task:** Cancels the selected background task.
- **Suspend Running Task:** Pauses the selected background task (could be resumed).

The following additional options are available for suspended processes:

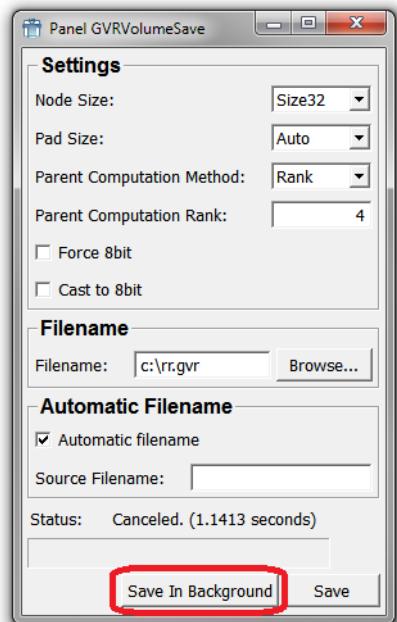
- **Cancel Suspended Task:** Cancels the suspended background task.
- **Resume Suspended Task:** Resumes the processing of the suspended background task.

If a network is closed while a background task is running, a warning is displayed:

Figure 7.4. Warning for Running Background Tasks

Background tasks are implemented for some modules, for example `GVRVolumeSave`, `GVRImageToVolume` and `ImageSave`. For other modules, check whether their panels offer features like "Save in Background".

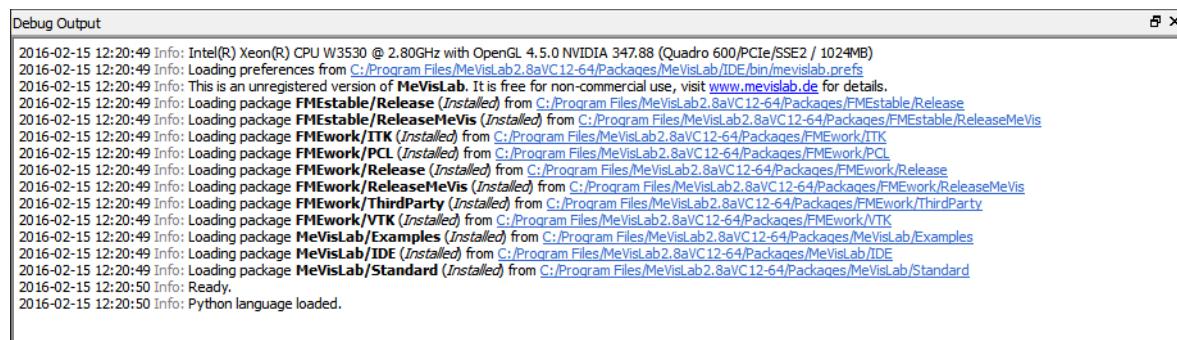
Figure 7.5. Save in Background for GVRVolumeSave



Chapter 8. Debug Output

The **Debug Output** View shows all events in the same way as they are written to the log file (see [Section 4.3.5, “Preferences — Paths”](#)).

Figure 8.1. Debug Output



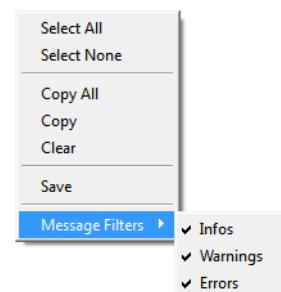
```
Debug Output
2016-02-15 12:20:49 Info: Intel(R) Xeon(R) CPU W3530 @ 2.80GHz with OpenGL 4.5.0 NVIDIA 347.88 (Quadro 600/PCIe/SSE2 / 1024MB)
2016-02-15 12:20:49 Info: Loading preferences from C:/Program Files/MeVisLab2.8aVC12-64/Packages/MeVisLab/IDE/bin/mevislab.prefs
2016-02-15 12:20:49 Info: This is an unregistered version of MeVisLab. It is free for non-commercial use, visit www.mevislab.de for details.
2016-02-15 12:20:49 Info: Loading package FMEstable/Release (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEstable/Release
2016-02-15 12:20:49 Info: Loading package FMEstable/ReleaseMeVis (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEstable/ReleaseMeVis
2016-02-15 12:20:49 Info: Loading package FMEwork/ITK (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/ITK
2016-02-15 12:20:49 Info: Loading package FMEwork/PCL (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/PCL
2016-02-15 12:20:49 Info: Loading package FMEwork/Release (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/Release
2016-02-15 12:20:49 Info: Loading package FMEwork/ReleaseMeVis (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/ReleaseMeVis
2016-02-15 12:20:49 Info: Loading package FMEwork/ThirdParty (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/ThirdParty
2016-02-15 12:20:49 Info: Loading package FMEwork/VTK (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/FMEwork/VTK
2016-02-15 12:20:49 Info: Loading package MeVisLab/Examples (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/MeVisLab/Examples
2016-02-15 12:20:49 Info: Loading package MeVisLab/IDE (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/MeVisLab/IDE
2016-02-15 12:20:49 Info: Loading package MeVisLab/Standard (Installed) from C:/Program Files/MeVisLab2.8aVC12-64/Packages/MeVisLab/Standard
2016-02-15 12:20:50 Info: Ready.
2016-02-15 12:20:50 Info: Python language loaded.
```

Paths and files for which standard programs are defined are marked as links (in blue and underlined). Click them to open the folder or file.

To clear the **Debug Output**, click it and press **L**. This has no effect on the log file, and clear actions are not being logged.

A context menu with editing options is available when right-clicking the **Debug Output**. The context menu has a submenu to manage message filters for info messages, errors, and warnings. The filters do not prevent messages from being printed, but only from being shown. This means that previously hidden messages will be shown when the message type is enabled again, unless the **Debug Output** has been cleared.

Figure 8.2. Context Menu



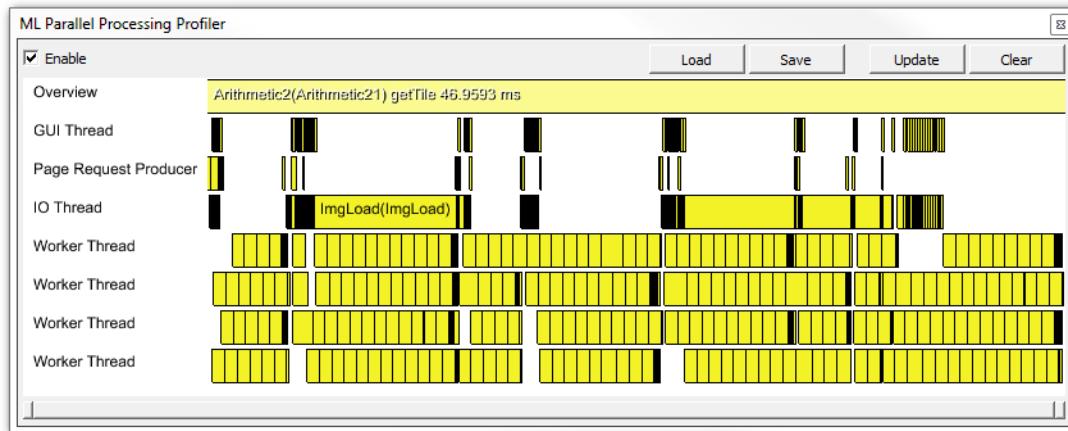
Chapter 9. ML Parallel Processing Profiler View

The **ML Parallel Processing Profiler View** shows the workload of all threads and how much time they needed.

The view shows a zoomable graph and has the following options:

- **Enable:** If checked, the view tracks the time consumption of each used thread. Press **Update** to refresh the view.
- **Load:** Loads a previously saved *.timeline file.
- **Save:** Saves a profiling result as a *.timeline file.
- **Update:** Updates the view with the latest results.
- **Clear:** Clears all profiling data and clears the view.

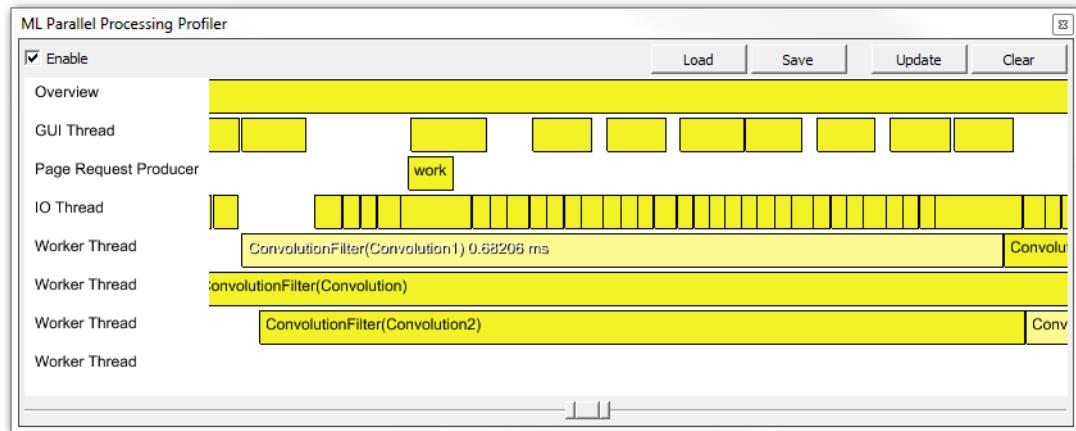
Figure 9.1. Parallel Processing View Overview



The graph shown in the view gives an overview about how many threads have been used and how long they have worked on single requests. The graph can be zoomed with the mouse-wheel or by interacting with the scrollbar.

On mouse-over, the single work packages show the module's name and instance name that produced the work, and the time it took to process the work package.

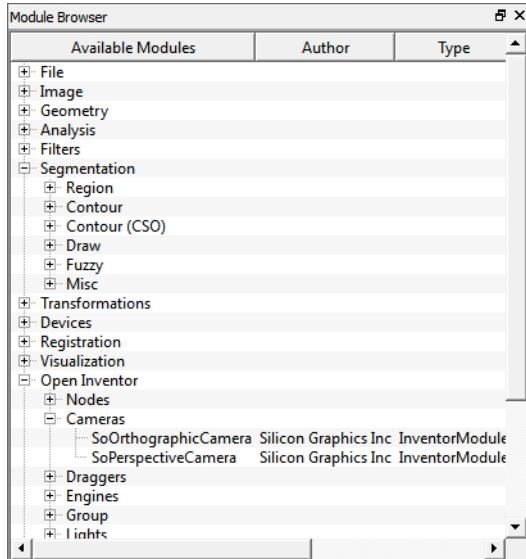
Figure 9.2. Parallel Processing View Details



Chapter 10. Module Browser

Like the [Section 4.4, “Modules Menu”](#), the **Module Browser** displays a tree of all modules currently available in the MeVisLab module database, sorted both by genres and by DLLs (projects). However, it allows browsing the modules, viewing the basic information of each module (author, package, etc.) and opening a context menu with various features (editing the definition file, opening the example network, etc.).

Figure 10.1. Module Browser



Double-click a module to create it in the current network.

The context menu is the same as for modules in the search results, see [Section 13.3, “Module Search Result Context Menu”](#).

Chapter 11. Module Inspector

The **Module Inspector** gathers and displays extensive module information. This information can also be found in other parts of the software.

11.1. Fields

The Fields tab lists all fields available in the module.

Figure 11.1. Module Inspector — Fields

Name	Type	In	Out	Value
instanceName	String			Threshold
threshold	Double			420
relativeThreshold	Bool			FALSE
comparisonOperator	Enum			Greater
conditionTrueWriteValue	Enum			UserDef
userConditionTrueValue	Double			1
conditionFalseWriteValue	Enum			UserDef
userConditionFalseValue	Double			0

The Fields tab contains the same information as the automatic panel's **Parameters** tab.

Figure 11.2. Automatic Panel

Name	Type	In	Out	Value
instanceName	String			Threshold
threshold	Double			420
relativeThreshold	Bool			FALSE
comparisonOperator	Enum			Greater
conditionTrueWriteValue	Enum			UserDef
userConditionTrueValue	Double			1
conditionFalseWriteValue	Enum			UserDef
userConditionFalseValue	Double			0

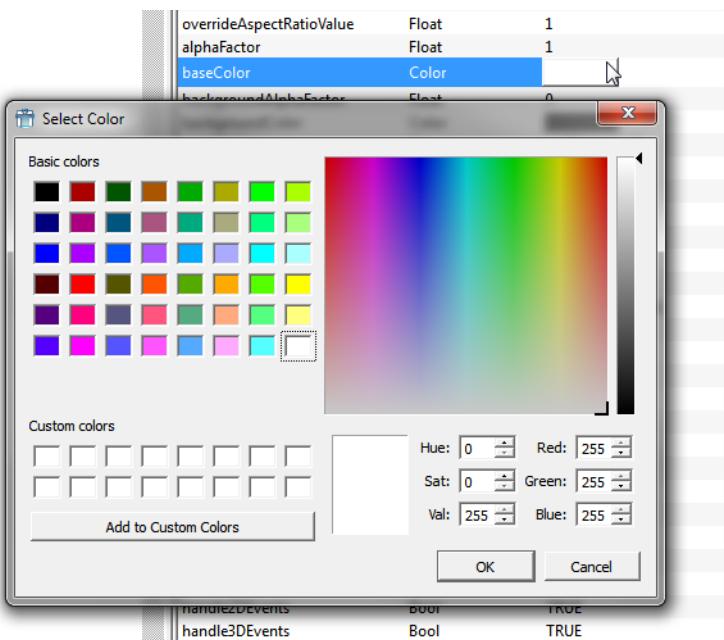
As they can be edited in the same way, they shall be discussed together here.

11.1.1. Editing Field Values

The values of fields can be edited. The options depend on the parameter type. Click the entry to edit. To finish the edit, press **RETURN** (to save the field value) or **TAB** (to save the field value and open the next field value for editing).

Figure 11.3. Module Inspector — Edit Boolean

numXSlices	Integer	1
maxSlice	Integer	0
keepSlicesInView	Bool	FALSE
slab	Integer	TRUE
lowerLeft	Vector2	FALSE
upperRight	Vector2	11

Figure 11.4. Module Inspector — Edit Color**Figure 11.5. Module Inspector — Edit Text**

instanceName	String	SoView2D
interactionProviderID	String	
vieweid	String	viewer
startSlice	Integer	0

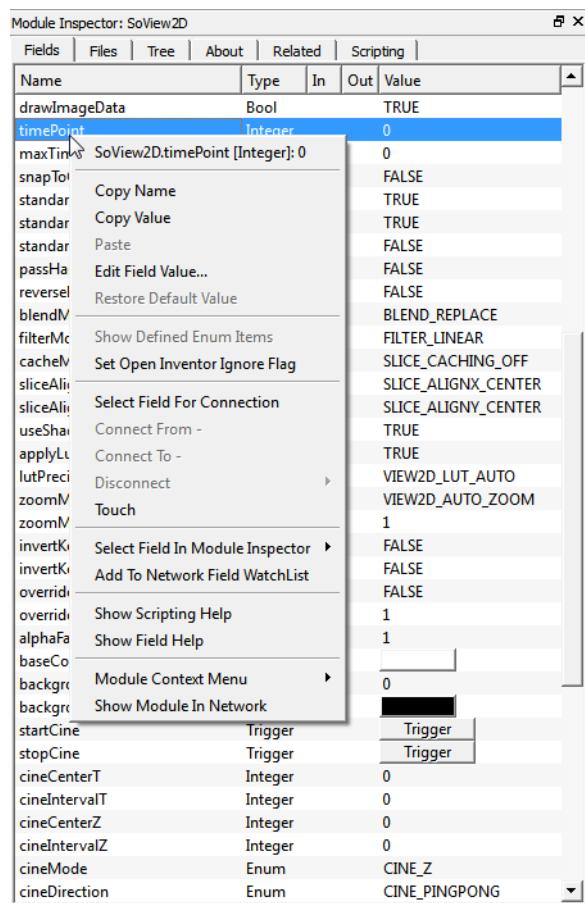
Figure 11.6. Module Inspector — Edit Values

overrideAspectRatio	Bool	FALSE
overrideAspectRatioValue	Float	1
alphaFactor	Float	≈0.644
baseColor	Color	[Color Swatch]
backgroundAlphaFactor	Float	0

Just like with automatic panels, parameter connections can be created by dragging the parameter of another module onto the Fields tab.

11.1.2. Module Inspector Fields Context Menu

For the fields of the **Module Inspector**, a context menu is available. The options depend on the type of the field and if it is part of a parameter connection.

Figure 11.7. Module Inspector Fields Context Menu

<ModuleName>.<FieldName> [FieldType]: <Value>

Displays the module name, the name of the field, its type, and its current value.



Tip

If the title is selected, the <ModuleName>.<FieldName> is copied to the paste buffer. This string can be pasted into any text editor, for example into MATE.

Copy Name

Copies the name of the field to the paste buffer.

Copy Value

Copies the value of the field to the paste buffer.

Paste

Pastes the field's value. The field's value can not only be pasted into an editor but also directly into another field.

Edit Field Value...

Opens an editor to edit the field's value.

Restore Default Value

Restore the default value of the field. This is only available if the current value is not the default value and the field is editable. (The field's value is displayed in a bold font in this case.)

Show Defined Enum Items...

Opens a window with four different representations of the enumeration's items:

- String list: the names of the items as strings
- MDL code: a snippet of MDL code that defines a field with the enumeration items
- Python code: a snippet of Python code that defines directives to react to each chosen item
- Detailed list: a list of the enumeration items with the item's name, the item's title, and the item's integer representation

This option is only available if the field is an enumeration field.

Set Open Inventor Ignore Flag

Toggles the field to be ignored. Only available for Open Inventor fields.

Select Field for Connection

Selects the field so that it can be connected to another field in the next step.

Connect From

Connects the selected field as source. Shows the field to connect from in the submenu.

Connect To

Connects the selected field as destination. Shows the field to connect to in the submenu.

Disconnect

Disconnects a parameter connection (to be selected from the submenu).

Touch

Touches the field value without changing it. This might trigger a recalculation of values or outputs.

Select Field In Module Inspector

Selects the field of a parameter connection to jump to in the display.

Add To Network Field WatchList

Adds the parameter to the **Network Field WatchList**, see [Chapter 14, Network Field WatchList](#).

Show Scripting Help

Opens the scripting reference for the field's type in the default web browser.

Show Field Help

Opens the field's mhelp documentation in the default web browser.

Module Context Menu

Opens the module's context menu in a submenu.

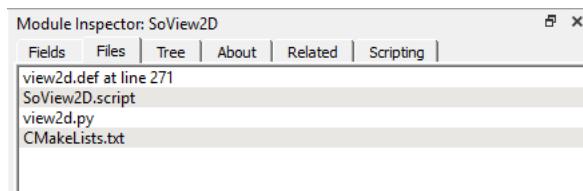
Show Module In Network

Selects the module and zooms to it in the network.

11.2. Files

The Files tab contains a list of related files. This is the same information as in the module context menu, **Related Files**.

Figure 11.8. Module Inspector — Files

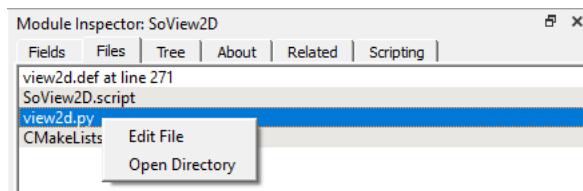


Double-click a file to edit it in the default text editor.

11.2.1. Module Inspector Files Context Menu

For files, only two options are available on the context menu:

Figure 11.9. Module Inspector Files Context Menu



Edit File

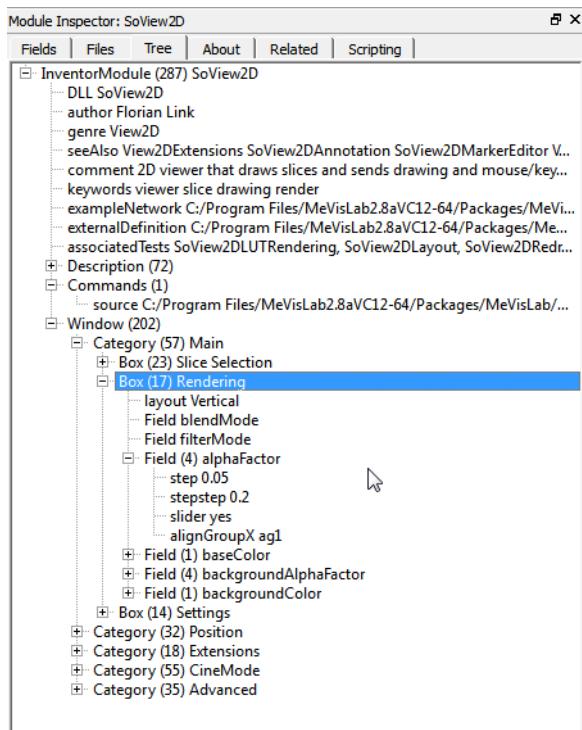
Opens the file in the default text editor (same effect as double-click).

Open Directory

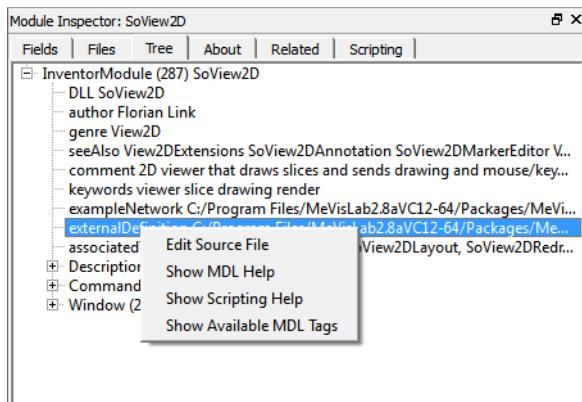
Opens the directory of the file in the standard file browser.

11.3. Tree

The Tree tab displays the module source file as an MDL tree structure.

Figure 11.10. Module Inspector — Tree

11.3.1. Tree Context Menu

Figure 11.11. Module Inspector Tree Context Menu

Edit Source File

Opens the .def file in the default text editor.

Show MDL Help

Opens the MDL help (HTML) in the default web browser and jumps to the according entry.

Show Scripting Help

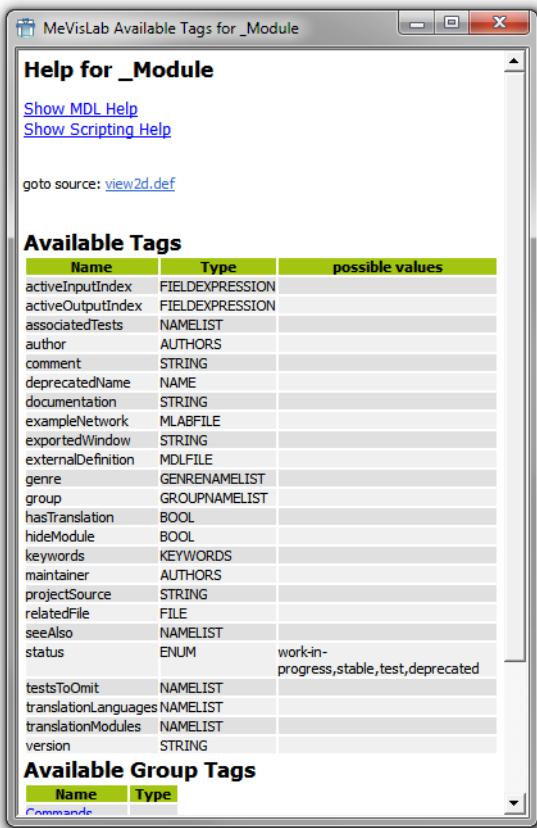
Opens the Scripting help (HTML) in the default web browser and jumps to the according entry.

Show Available MDL Tags

Opens a new window with a list of available tags and group tags. The tags depend on the module type.

The listed types of the MDL tags do not always correspond to the types listed in the MDL help. The MDL help lists simplified versions of the tag types.

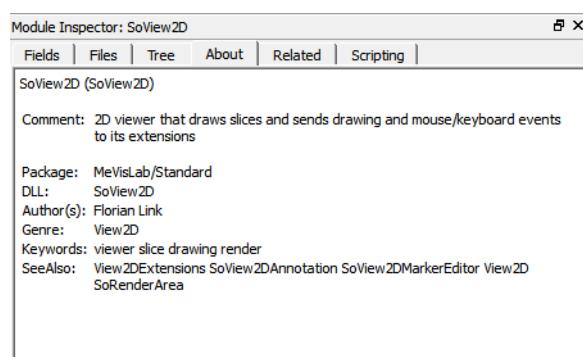
Figure 11.12. Show Available MDL Tags



11.4. About

The About tab displays the header tags and their contents from the module definition (the same information as shown on top of the module HTML help file).

Figure 11.13. Module Inspector — About



11.5. Related

The Related tab lists all modules related to the currently selected module.

The relatedness is shown as a blue bar in the right-most column. The related modules are sorted by default by their relatedness

The relatedness score is computed by the `seeAlso` and the keyword tags, and also by the modules' names. In case of the `seeAlso` tag, the relatedness works bidirectional: if a module A refers to a module B via the `seeAlso` tag, the relatedness score of module B to module A also raises. This way, the `seeAlso` tags of only one of the modules needs to be updated.

Figure 11.14. Module Inspector — Related

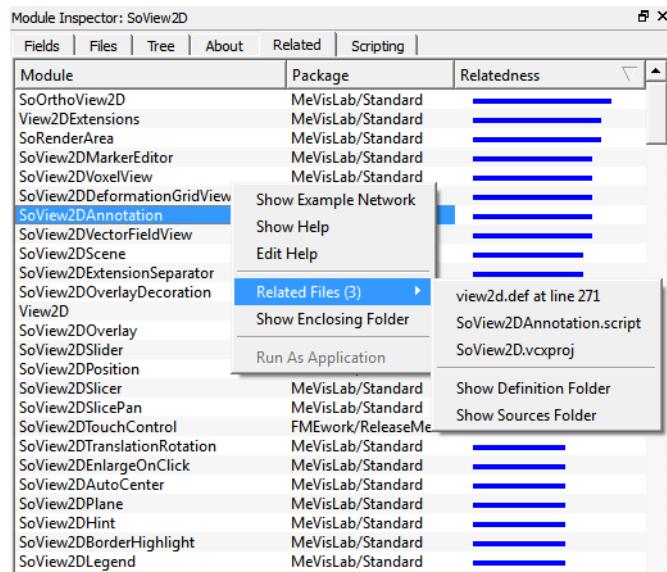
Module Inspector: SoView2D		
Fields	Files	Tree
About	Related	Scripting
Module	Package	Relatedness
SoOrthoView2D	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
View2DExtensions	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoRenderArea	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DMarkerEditor	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DValueView	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DDeformationGridView	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DAnnotation	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DVectorFieldView	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DScene	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DExtensionSeparator	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DOverlayDecoration	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
View2D	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DOverlay	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DSlider	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DPosition	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DSlicer	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DSlicePan	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DTouchControl	FMEwork/ReleaseMeVis	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DTranslationRotation	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DEnlargeOnClick	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DAutoCenter	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DPlane	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DHint	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DBorderHighlight	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DLegend	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DSliceZoom	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DMenu	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DOverlayMPR	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DLabel	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DMenuItem	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DExample	MeVisLab/Examples	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DRectangle	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DMenuItems	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DBorder	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DValue	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DContour	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
OrthoView2D	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
GVROrthoView2D	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DRigidRegistrationEditor	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DExtensionSampler	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
SoView2DCSOEditor	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>
ImageViewer	MeVisLab/Standard	<div style="width: 100%;"><div style="width: 100%;"></div></div>

Scroll to the right for more information.

Double-click a related module to add it to the workspace.

11.5.1. Related Context Menu

Figure 11.15. Module Inspector Related Context Menu



Show Example Network

Opens the example network in the workspace.

Show Example Network Folder

Opens the folder of the example networks in the default file browser.

Show Help

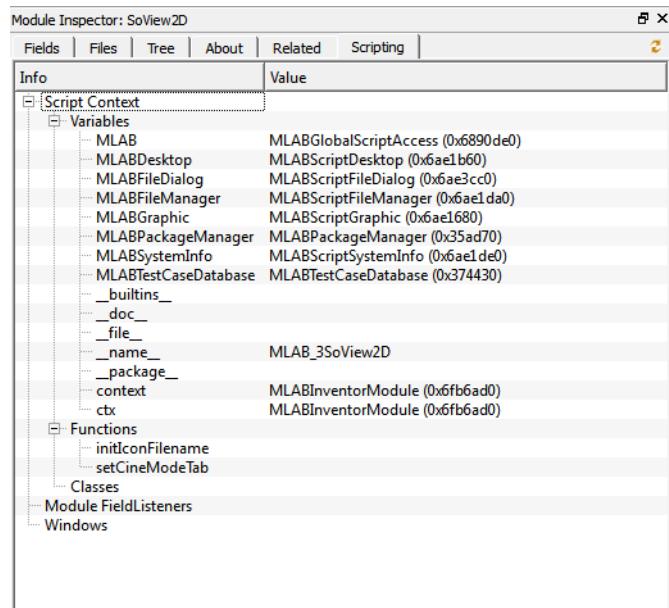
Opens the module help (HTML) in the default web browser.

Edit Module Definition File

Opens the .def file in the default text editor.

11.6. Scripting

The Scripting tab displays information about any scripts used within the module. The Script Context is for information purposes only. The ModuleFieldListeners can also be edited.

Figure 11.16. Module Inspector — Scripting

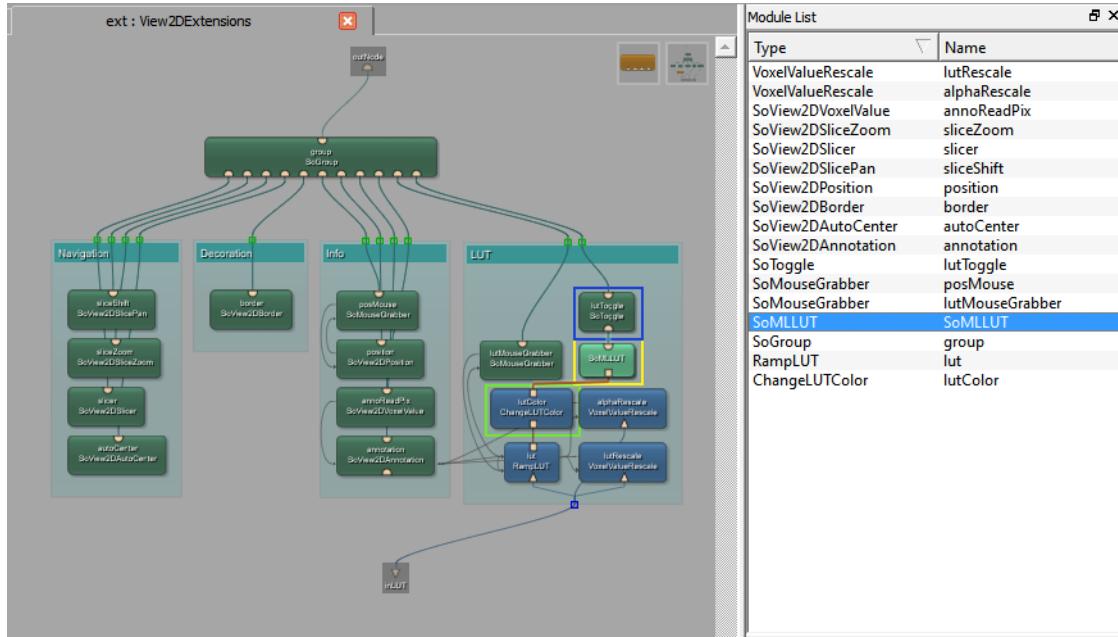
Click to refresh the scripting information. This is necessary if the script was changed outside of MeVisLab, either in MATE or in an external editor.

For the context menu of ModuleFieldListeners, see [Section 11.1.2, “Module Inspector Fields Context Menu”](#).

Chapter 12. Module List

The **Module List** lists all modules in the currently active network.

Figure 12.1. Module List



Select a module in the list to select and highlight it in the network and vice versa.

The context menu is the same as for the module in the workspace, see [Section 3.9.1, “Module Context Menu”](#).

Chapter 13. Module Search

13.1. Module Search

With the **Module Search** View, complex search filters can be constructed. When opened for the first time, it looks similar to the quick search in the toolbar, see [Chapter 5, Toolbar](#).

Figure 13.1. Module Search with Demo Entry

The screenshot shows the 'Module Search' view with the following interface elements:

- Search bar at the top with placeholder text 'example'.
- Text 'Modules found: 87'.
- Table with columns: Module, Author, Genre, Package, Type, Group, and seeAlso.
- Table data (partial list):

Module	Author	Genre	Package	Type	Group	seeAlso
AddImagesExample	Olaf Konrad		MeVisLab/Examples	MLModule		
AdvancedBackgroundTaskExample	Florian Link		MeVisLab/Examples	MLModule		
AlgorithmMacroModuleExample	Christian Kanthak	Examples	FMEstable/ReleaseMeVis	MacroModule		
AlgorithmModuleExample	Christian Kanthak	Examples	FMEstable/ReleaseMeVis	MLModule		
AlgorithmModuleExampleWithOutp...	Christian Kanthak	Examples	FMEstable/ReleaseMeVis	MLModule		AlgorithmM...
ApplicatorMacroExample	Olaf Konrad	Visualization	MeVisLab/Examples	MacroModule		
AsyncProcessAllPagesExample	Florian Link		MeVisLab/Examples	MLModule		
AsyncTileProcessingExample	Florian Link		MeVisLab/Examples	MLModule		
AsyncTypedProcessAllPagesExample	Florian Link		MeVisLab/Examples	MLModule		
BaseOwnerExample	Olaf Konrad		MeVisLab/Examples	MLModule		SoBaseRece...
ColorChooserExampleControlTest	Uwe Siems	Examples	MeVisLab/Examples	MacroModule	test	
Comment	Florian Link	Extras	MeVisLab/Standard	MacroModule		
DiagramExampleControlTest	Uwe Siems	Examples	MeVisLab/Examples	MacroModule	test	
ExampleCSOAlgorithmModule	Grzegorz Chlebus		FMEwork/ReleaseMeVis	MLModule		
ExampleGUIClipping	Stephan Kuhagen		MeVisLab/Standard	MacroModule	example	
FileSystemItemModelData	Uwe Siems		MeVisLab/Examples	MacroModule		FileSystemI...
FileSystemItemModelView	Uwe Siems		MeVisLab/Examples	MacroModule		FileSystemI...
GlobalPagedImageExample	Wolf Spindler, Ola...		MeVisLab/Examples	MLModule		
KernelIn2OutExample	Wolf Spindler	Kernel	MeVisLab/Examples	MLModule		Convolutio...
KernelExample	Wolf Spindler	Kernel	MeVisLab/Examples	MLModule		Convolutio...
MatplotlibMDLExample	Florian Link		MeVisLab/Examples	MacroModule		
OpenCVPlaygroundExample	Florian Link		MeVisLab/Examples	MacroModule		
ProcessAllPagesExample	Olaf Konrad		MeVisLab/Examples	MLModule		
PythonExampleControlsTest	Florian Link	Examples	MeVisLab/Examples	MacroModule	test	
PythonGetTileExample	Florian Link		MeVisLab/Examples	MacroModule	example	
PythonImageExample	Florian Link		MeVisLab/Examples	MacroModule	example	
PythonPagedImagingExample	Florian Link		MeVisLab/Examples	MLModule	example	
ReadDICOMTagExample	Olaf Konrad		MeVisLab/Examples	MLModule		
RemotePanelRenderingExampleMacro	Marcus Barann		MeVisLab/Standard	MacroModule		RemotePan...
RemoteRenderingCSOExampleModule	Uwe Siems		MeVisLab/Standard	MacroModule		RemoteRen...

Click the magnifier button to select search criteria.

The search starts immediately with each entered key (incremental search). The result list can be scrolled to the right for more information. Click a column header to sort the list by the contents of this column.

Double-click a module to add it to the network. If other Views are open, for example the **Module Inspector**, the module is also opened in this View.



Tip

A module can also be instantiated from the Module Search by dragging its name onto the network window.

13.2. Advanced Search

The advanced search allows for more complex search statements which can be combined. Click **[+]** to show advanced search entry fields.

Figure 13.2. Module Search — Advanced

The screenshot shows the 'Module Search' dialog with two search statements defined:

- name+keywords** contains example
- package** contains standard

Below the search area, it says "Modules found: 48". A table lists 16 modules, their authors, and genres.

Module	Author	Genre
Comment	Florian Link	Extras
ExampleGUIScripting	Stephan Kuhagen	
RemotePanelRenderingExampleMacro	Marcus Barann	
RemoteRenderingCSOExampleModule	Uwe Siems	
SoFlash	Felix Ritter	Visualizat
TestBatchCommand	Florian Link	
TestBorderlessWindow	Florian Link	
TextBoxLayout	Stephan Kuhagen	
TestButtonGroups	Marcus Barann	
TestComboBox	Florian Link	
TestControlVisibility	Florian Link	
TestCrossRefs	Tobias Boskamp	
TestDateTime	Uwe Siems	

In the first list, the area to be searched in has to be selected.

Figure 13.3. Module Search — Searching In

The screenshot shows the 'Module Search' dialog with two search statements defined:

- name+keywords** contains example
- genre** contains standard

A dropdown menu is open under the second search statement, showing options: name, group, Cc, type, seeAlso, Ex, name+keywords, Re, common tags, RemoteRenderingCSOExampleModule, and ExampleMacro. The 'name+keywords' option is highlighted.

Below the search area, it says "Modules found: 48". A table lists 16 modules, their authors, and genres.

Module	Author	Genre
Comment	Florian Link	Extras
ExampleGUIScripting	Stephan Kuhagen	
RemotePanelRenderingExampleMacro	Marcus Barann	
RemoteRenderingCSOExampleModule	Uwe Siems	
SoFlash	Felix Ritter	Visualizat
TestBatchCommand	Florian Link	
TestBorderlessWindow	Florian Link	
TextBoxLayout	Stephan Kuhagen	
TestButtonGroups	Marcus Barann	
TestComboBox	Florian Link	
TestControlVisibility	Florian Link	
TestCrossRefs	Tobias Boskamp	
TestDateTime	Uwe Siems	

In the second list, the operator has to be selected.

Figure 13.4. Module Search — Operators

The screenshot shows the 'Module Search' dialog with two search statements defined:

- name+keywords** contains example
- package** contains word contains any

A dropdown menu is open under the second search statement, showing options: contains, contains word, contains any, regexp, equals, and doesn't contain. The 'contains any' option is highlighted.

Below the search area, it says "Modules found: 48". A table lists 16 modules, their authors, and genres.

Module	Author	Genre
Comment	Florian Link	Extras
ExampleGUIScripting	Stephan Kuhagen	
RemotePanelRenderingExampleMacro	Marcus Barann	
RemoteRenderingCSOExampleModule	Uwe Siems	
SoFlash	Felix Ritter	Visualizat
TestBatchCommand	Florian Link	
TestBorderlessWindow	Florian Link	
TextBoxLayout	Stephan Kuhagen	
TestButtonGroups	Marcus Barann	
TestComboBox	Florian Link	
TestControlVisibility	Florian Link	
TestCrossRefs	Tobias Boskamp	
TestDateTime	Uwe Siems	

Only visible search statements are executed. By clicking , less search entries are visible (and active).

Search statements can be turned on or off with the check box in front of the entry.

Search statements offer two additional options at the end of the line:

: Negates the statement. For example, the search is done for “does not contain” instead of “contains” .

: Applies case-sensitive search.



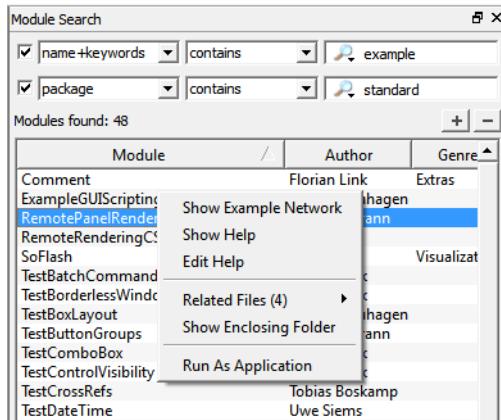
Note

Selections in the **Module Search** View are persistent and set as default the next time the View is used.

13.3. Module Search Result Context Menu

For the results of the module search, a context menu is available. The options depend on the module. For example, if no example network is available, this option is grayed out in the context menu.

Figure 13.5. Module Search Results — Context Menu



13.3.1. General Options

Show Example Network

Opens the example network in the workspace.

If a module has no example network, this option is grayed out.

If a module has multiple example networks, this entry is a menu with a submenu that shows a list of all available example networks by their names.

Show Help

Opens the module help (HTML) in the default web browser.

Edit Help

Opens the `.mhelp` file in the default text editor.

13.3.2. Additional Options for Macro Modules

Related Files

Lists all files belonging to the module. Possible file types are `.def/.script` (MDL definition files), `.mhelp/.html` (uncompiled/compiled help file), and `.py` (scripting files). Select a file to open it in the default editor (as set in [Section 4.3.4, “Preferences — Supportive Programs”](#)).

Show Enclosing Folder

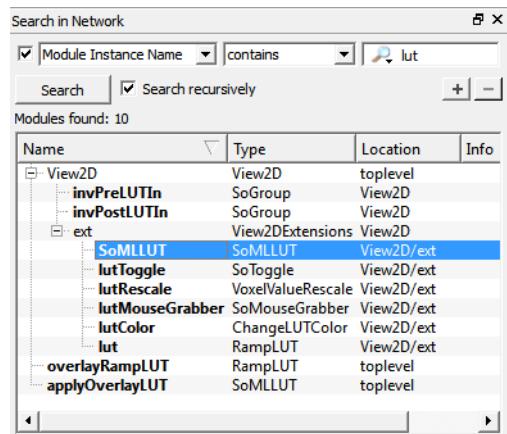
Opens the main folder of the module, which contains the `.def` file.

Run As Application

Runs the macro as an application in a stand-alone window.

13.4. Search in Network

Figure 13.6. Search in Network



The **Search in Network** View works the same as the module search but searches in network parameters/properties. For details, see [Chapter 13, Module Search](#).

Search recursively

If selected, the search also runs through the lower levels of embedded macro modules.



Note

Selections in the **Search in Network** View are persistent and set as default the next time the View is used.

Chapter 14. Network Field WatchList

In the **Network Field WatchList**, parameters of all modules in the network can be tracked.

Figure 14.1. Network Field WatchList

Name	Value
View2D.startSlice	40
LocalImage.trueName	C:/Program Files/MeVisLab2...
overlayRampLUT.center	0.5
overlayRampLUT.width	1
overlayRampLUT.alphaCenter	0.5
overlayRampLUT.alphaWidth	1
overlayRampLUT.computeLUTAnew	<input type="button" value="Trigger"/>

As the parameter values can be edited just like in the single module's **Module Inspector**, this allows for editing parameter values of potentially multiple modules at a single point, instead of toggling through modules.

For the options in the context menu, see [Section 11.1.2, “Module Inspector Fields Context Menu”](#).

In addition to these options, the following two are available:

Remove From Network Field WatchList

Removes the field from the WatchList.

Clear Network Field WatchList

Clears the complete WatchList (you are asked to confirm this).

Chapter 15. Output Inspector

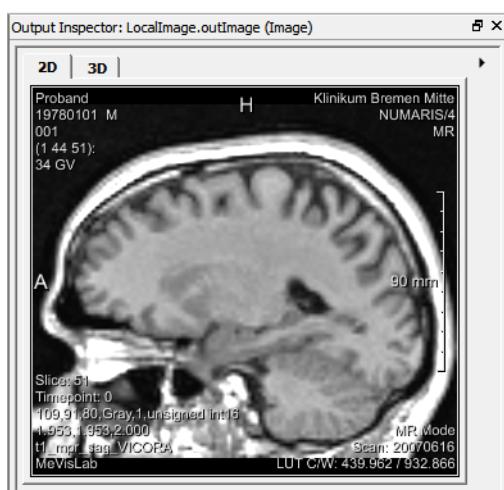
The **Output Inspector** displays the output of module connectors. In comparison to any defined view modules, the **Output Inspector** allows for a quick check of any inputs/outputs in the network by simply clicking them.

Output inspectors are based on inspector modules. They can be found with Quick Search. The following inspectors are available:

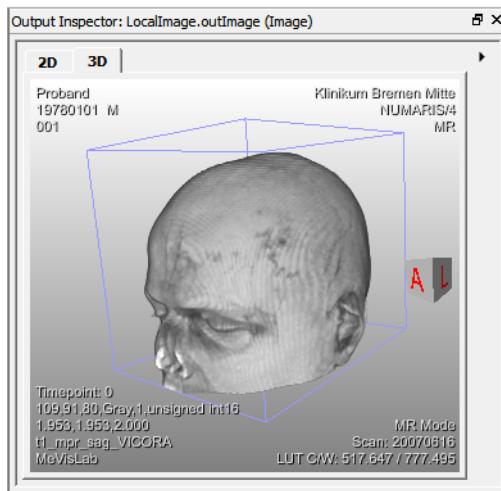
- **MLImageInspector:** for ML modules
 - **MLBaseInspector:** for **MLBase**
 - **MarkerListInspector:** for **Xmarker** list
 - **SceneInspector:** for Open Inventor scene
 - **CSOInspector:** for CSO lists
 - **MLLUTInspector:** for LUT function
 - **CurveInspector:** for curve data and curve list
 - **WEMInspector:** for all WEM modules
 - **GVRVolumeInspector:** for GVR volume
 - **StylePaletteInspector:** for a style palette
 - **ItemModelInspector:** for hierarchical item model

The panel and the ***Output Inspector*** of inspector modules are identical.

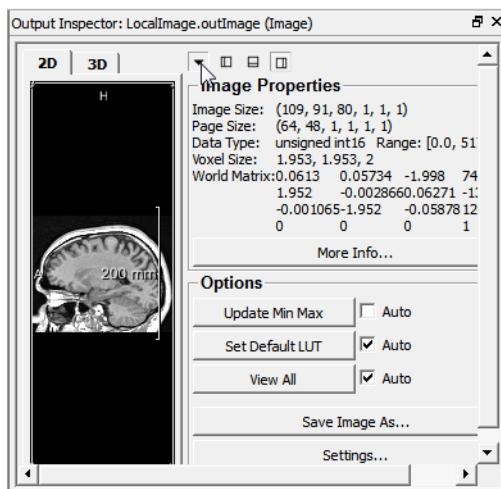
Figure 15.1. ML Image Inspector



Many, but not all inspectors have different views on the data:

Figure 15.2. ML Image Inspector: 3D View

Some inspectors offer more detailed information if clicking on the triangle symbol to the upper right hand side:

Figure 15.3. ML Image Inspector: Detailed Information

Which **Output Inspector** is to be used for which object type is defined in the `OutputInspectors.def` file in the MeVisLab IDE package. It is possible to develop new inspectors and add them to this definition file.

Chapter 16. Parameter Connections Inspector

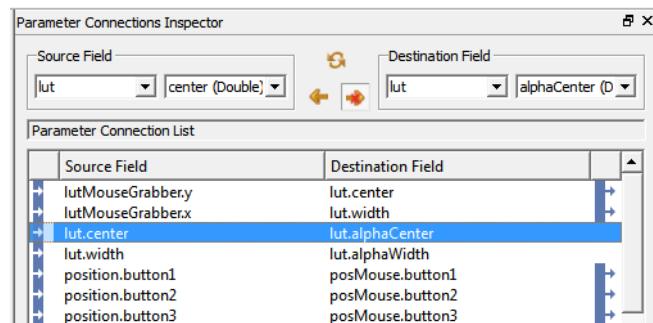
The **Parameter Connections Inspector** works only on parameter connections, not on data connections.

As explained in [Section 3.3, “Connector and Connection Types”](#), data connections may be of the types Base (square/brown), Inventor (half circle/green) and ML image (triangle/blue). Only connectors of fitting types may be connected, by clicking the connector and drawing a line to the other connector.

In case of parameter connections, field values are connected. This can be used basically for any type of fields. Modules with no input/output connectors can only be connected by parameter connections, for example calculation modules like `ComposeRotation`. For more examples on using parameter connections, see the Getting Started, chapter “Parameter Connection for Synchronization”.

16.1. Parameter Connections Inspector View

Figure 16.1. Parameter Connections Inspector View



Parameter connections need a source and a destination field. If the source field changes, for example a value increases from 1 to 20, the destination field value increases the same. Source and destination fields can be connected bidirectionally; in that case, increasing the value in the destination field would also increase the value in the source field (see [Figure 16.2, “Parameter Connection Example — view2D and view3D”](#)). A destination field can also be the source for another connection, effectively forwarding the parameter value. There is no limit to the number of parameter connections, it is only given by the available fields per module.

A parameter connection can be created in three ways:

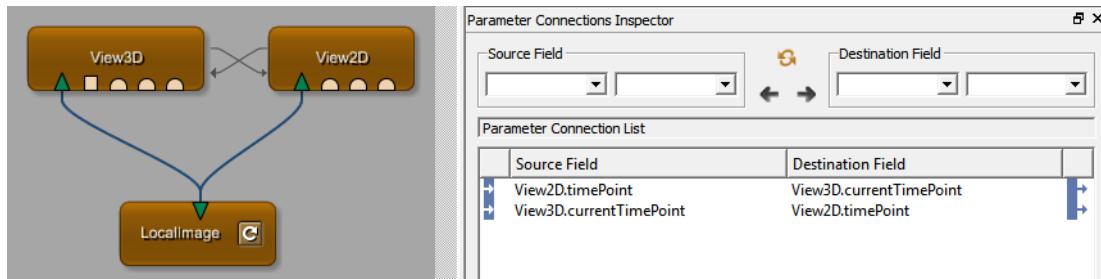
- by using the drop-down lists on top of the **Parameter Connections Inspector**. Select the modules and fields to connect and click the respective arrow for creating the connection. By clicking the circle button, source and destination can be swapped.
- by dragging fields (parameter labels) from one automatic panel to the other. This works also for buttons.
- by dragging fields from one panel or settings window to the other. This does usually not work for buttons.



Tip

If the button trigger field is created with “field”, the label can be connected by dragging.

A minimalist bidirectional parameter connection might connect a `View2D` and a `View3D` module so that the time points would be synchronized:

Figure 16.2. Parameter Connection Example — View2D and view3D**Note**

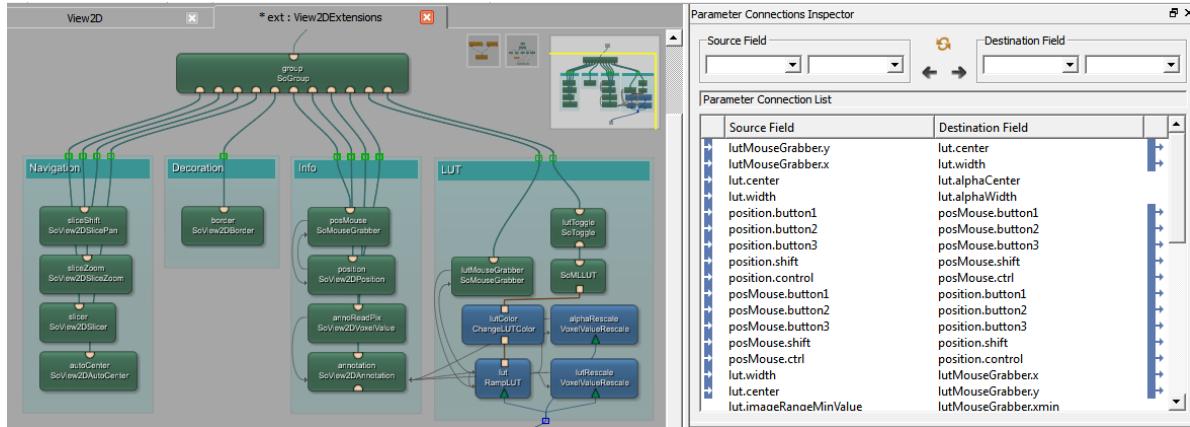
Fields cannot be connected if they are encapsulated in macros or in already defined panels. (Such fields would only be reachable by scripting.) In this case, that means that the cine settings of View2D cannot be connected to the movie settings of View3D.

Each source/destination field pair in the list shows an existing connection. If the fields are also the destination or source of another connection, additional arrow symbols are displayed:

Table 16.1. Connections Symbols

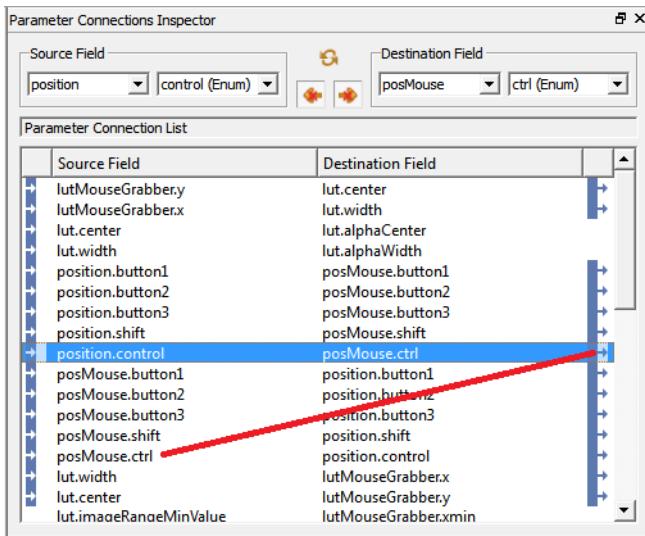
Symbol	Direction
↑	Source field
↓	Destination field
↔	Source and destination field

A more realistic example from the internal network of `View2DExtensions` shows a complex picture with many forwarded parameters.

Figure 16.3. Parameter Connection Example — View2DExtensions

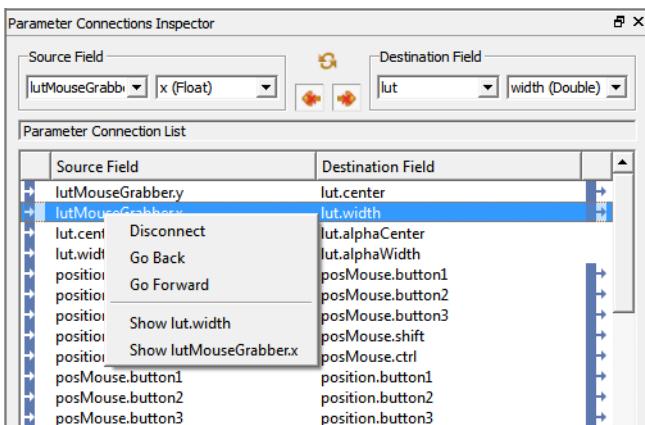
To navigate from destination and source and vice versa, double-click on the respective arrows.

Example: In [Figure 16.4, “Parameter Connection Example — Navigating between Fields”](#), `lut.grayWidth` is the source of a parameter connection to the destination `lut.alphaWidth`. `lut.alphaWidth` itself is the source for another connection. A double-click on the arrow symbol will highlight the list entry where `lut.alphaWidth` is the source field.

Figure 16.4. Parameter Connection Example — Navigating between Fields

16.2. Parameter Connections Inspector Context Menu

In the **Parameter Connections Inspector**, a context menu is available:

Figure 16.5. Parameter Connections Inspector Context Menu

Options:

- **Disconnect**: Disconnects the parameter connection.
- **Go Back** (depending on connection): Selects the parameter connection in which the current source field is the destination field.
- **Go Forward** (depending on connection): Selects the parameter connection in which the current destination field is the source field.
- **Show <field>**: Highlights the associated module in the network. If other Views are open, like the **Module Inspector** or the **Module List**, also highlights the according field or module there.

For the context menu of parameter connections in the network, see [Section 3.10.2.1, “Context Menu of Parameter Connections”](#).

Chapter 17. Profiling

The **Profiling** view generally measures time and memory consumption of modules in a network.

17.1. Introduction to Profiling

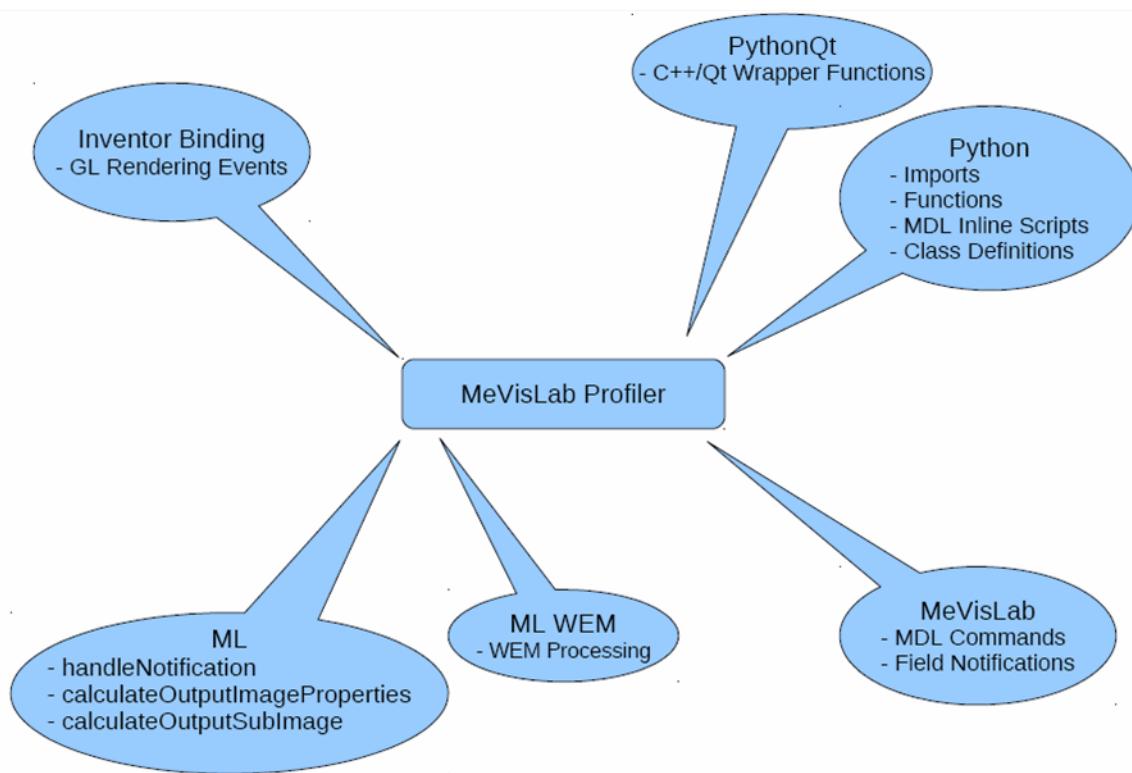
Profiling is dynamic program analysis (as opposed to static code analysis) and is used to identify slow functions, frequently called functions, and memory usage during runtime. Outside of MeVisLab, a number of profilers exists: gprof, GlowCode, Valgrind, DevPartner/BoundsChecker.

The advantages of profiling inside of MeVisLab are:

- Network performance can be analyzed by profiling on the C++ and Python level, with an inherent awareness of MeVisLab entities like modules, PagedImages, etc.
- No code recompilation is required.
- No additional programs are necessary, which makes profiling possibly faster.

What can be profiled?

Figure 17.1. Functions to be Profiled



- All ML modules offer profiling as the profiling is implemented in the base class `ml::Module`.
- WEM and CSO Modules also support the profiling of time consumption. However, in general the profiling of the memory consumption is not supported as this requires the memory being managed by the internal memory manager of the ML. So the memory managed by these modules is either not profiled at all or only the portions of the module that use ML methods are profiled.
- Python functions, scripts definitions, and Python Qt wrappers.

- Inventor Bindings
- MDL commands and field notifications



Note

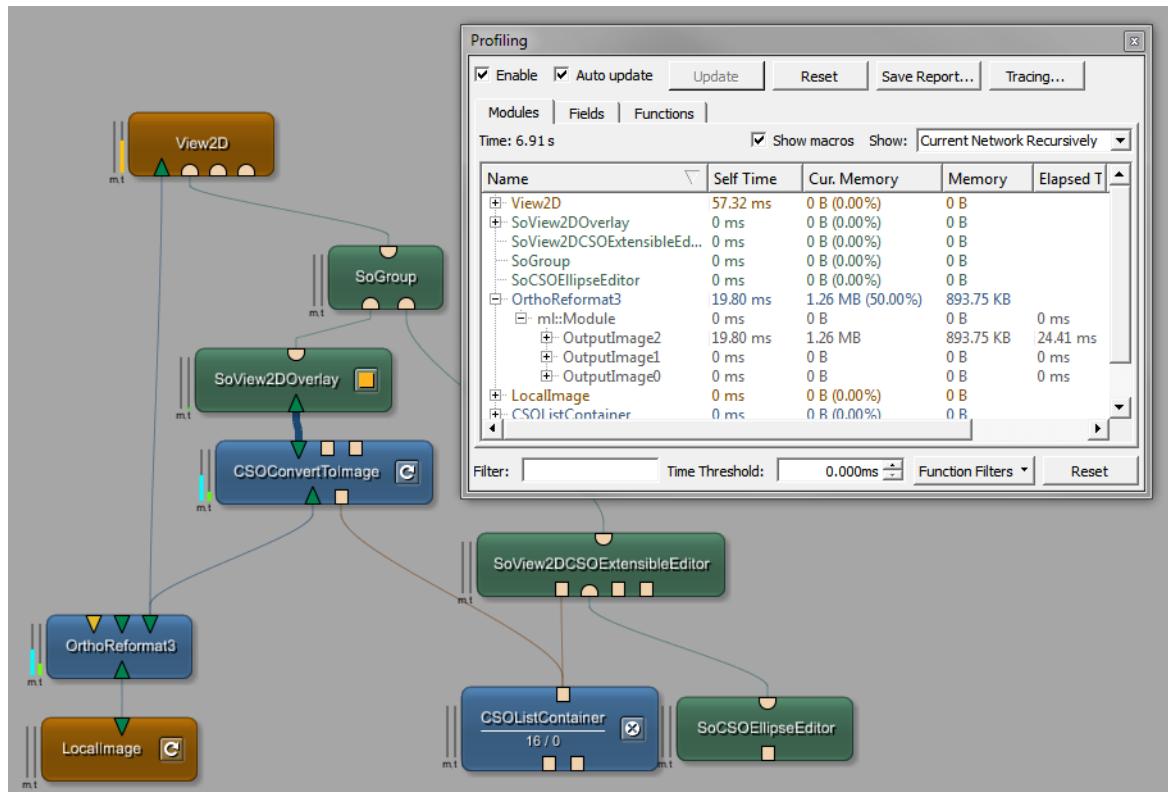
The **Profiling** view marks processes that use multi-threading and shows only their accumulated time in this view. If you need detailed profiling information about each thread, use the view described in [Chapter 9, ML Parallel Processing Profiler View](#).

17.2. Using Profiling

In the **Profiling** View, the modules of networks to be profiled are listed.

- Entries have the same color as corresponding modules (brown for macros, blue for ML modules and green for Inventor modules).
- Entries are set in italics if the module has been deleted (not just removed).

Figure 17.2. Profiling



The consumption is also displayed in the network: Two vertical bars to the left of a module indicate the percentage of memory (m) consumption and time (t) consumption of that module in the current network.

- The memory bar's color ranges from green over teal to magenta (for 0%..50%..100%).
- The CPU time bar's color ranges from green over yellow to red (for 0%..50%..100%).

That means that small green dots indicate low memory/time consumption while full magenta or red bars indicate high memory/time consumption. The colors are chosen so that memory and time are easy to distinguish from each other.

The **Profiling** View and the network are linked as follows:

- Clicking an entry on the **Modules** tab selects the module in the network (and vice versa).
- Double-clicking an entry on the **Modules** tab navigates to the module or, in case of a scripting function being counted, to the line of the scripting in MATE (this also works for modules inside macros).

Options:

- **Enable:** Enables the profiling and toggles the visualization of the relative time/memory consumption of the modules.
- **Auto update:** Enables the automatic display of profiling results every second.
- **Filters visible (Functions only):** Enables the display of the filter section for function filtering, see [Figure 17.9, “Functions with Filters Visible”](#).

Buttons:

- **Update:** (Only if no auto update is set) Updates the displayed results manually.
- **Reset:** Resets the list of modules and the profiling results. Memory/cache that is still in use will remain listed in the **Current Memory** column.
- **Save report:** Generates an HTML page with reports for Modules, Call Graph and Flat Profile (for functions), and Fields.

Figure 17.3. Profiling Report

Profiling Report: profilingReport

Di 16. Feb 14:47:44 2016

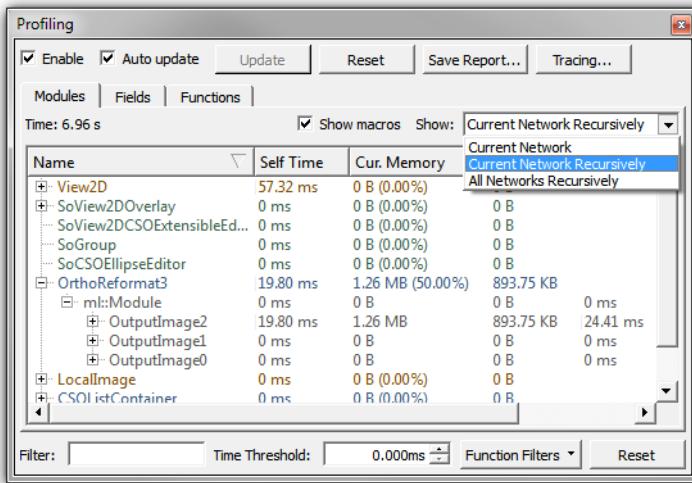
Modules	Call Graph	Flat Profile	Fields						
Name	Type	Elapsed Time	Self Time	Time In Children	Min. Time	Max. Time	Cur. Memory	Memory	Count
+View2D	View2D (Macro Module)	57.32 ms			0 B (0.00%)		0 B		
+SoView2DOOverlay	SoView2DOOverlay (Inventor Module)	0 ms			0 B (0.00%)		0 B		
SoView2DCSOExtensibleEditor	SoView2DCSOExtensibleEditor (Inventor Module)	0 ms			0 B (0.00%)		0 B		
SoGroup	SoGroup (Inventor Module)	0 ms			0 B (0.00%)		0 B		
SoCSOEllipseEditor	SoCSOEllipseEditor (Inventor Module)	0 ms			0 B (0.00%)		0 B		
+OrthoReformat3	OrthoReformat3 (ML Module)	19.80 ms			1.26 MB (50.00%)		893.75 KB		
+LocalImage	LocalImage (Macro Module)	0 ms			0 B (0.00%)		0 B		
+CSOListContainer	CSOListContainer (ML Module)	0 ms			0 B (0.00%)		0 B		
+CSOConvertToImage	CSOConvertToImage (ML Module)	15.64 ms			1.26 MB (50.00%)		1.26 MB		

17.2.1. Modules

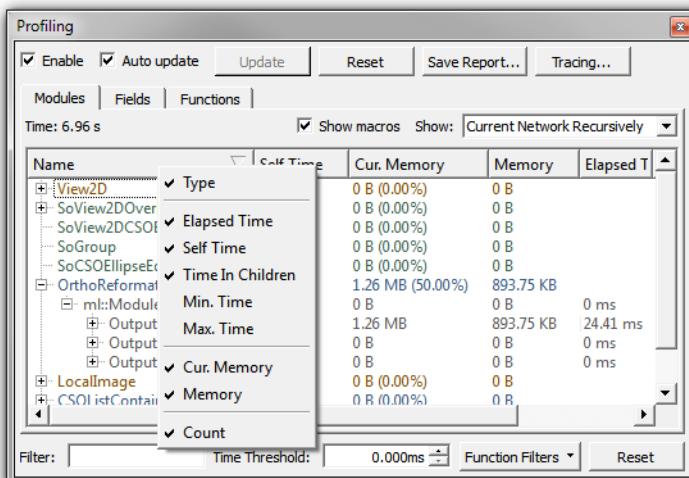
On the **Modules** tab, the modules and their profiling information are listed. On mouse-over over the headings, a context-sensitive tool tip is displayed for each.

Options:

- **Show macros:** If enabled, shows information of macros in the network and of modules inside that macro as a tree. If disabled, the information is flat and without the macros.
- **Show:** Defines which networks and depths should be profiled:

Figure 17.4. Profiling Modules

Right-clicking the headings opens the context menu in which the columns to be displayed can be configured.

Figure 17.5. Profiling — Heading Configuration

- Type:** Shows the type of the module.
- Elapsed Time:** Shows the total time spent in the profiled routines (the sum of self-time and time in children).
- Self Time:** Shows the time spent only in routines of the module.
- Time In Children:** Shows the time spent in routines called by the module.
- Min. Time:** Shows the minimum measured total time.
- Max. Time:** Shows the maximum measured total time.
- Cur. Memory:** Shows the memory currently allocated by the module.
- Memory:** Shows the total accumulated memory allocated by the module to ML pages during profiling.

- **Count:** Shows a counter of method calls. Expand a module's node to see the details; for example, the calls to `calculateOutputImageProperties` and to `calculateOutputSubImage` are counted, Page Cache hits and misses are counted, and calls to scripting methods are counted. On mouse-over, details are displayed.

17.2.2. Fields

On the **Fields** tab, all fields that have been touched at least once are listed here.

- **Name** Shows the field name.
- **Type** : Shows the field type.
- **#:** Shows the number of field triggers (notifications).

Figure 17.6. Profiling Fields

Name	Type	#
View2D.startSlice	Integer	331
View2D.plane	Plane	331
View2D.lutWidth	Double	67
View2D.lutCenter	Double	66
CSOConvertToImage.progress	Float	19
CSOConvertToImage.isPlaying	Bool	2
SoView2DOverlay.image	Image	1
CSOConvertToImage.output0	Image	1
CSOConvertToImage.outputXMarkerList	MLBase	1
CSOConvertToImage.apply	Trigger	1
CSOConvertToImage.done	Trigger	1
CSOConvertToImage.startVoxelBoundingBox	Vector3	1
CSOConvertToImage.endVoxelBoundingBox	Vector3	1

17.2.3. Functions

On the **Functions** tab, all functions that have been called at least once are listed here.

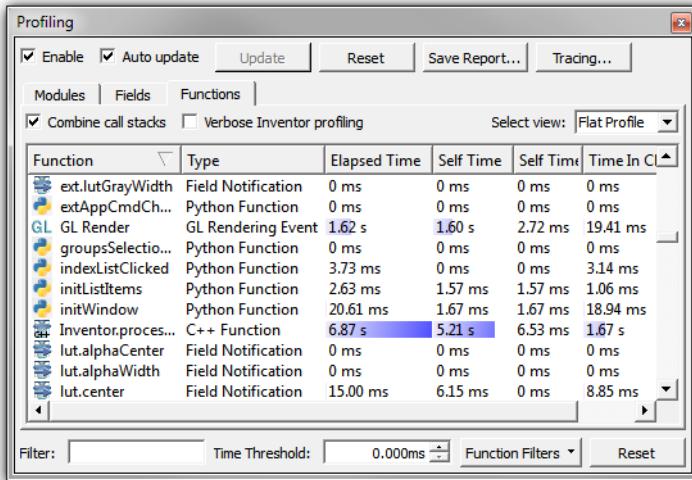
Right-clicking the headings opens the context menu in which the columns to be displayed can be configured.

- **Type:** Shows the function type. The possible types are listed beneath the **Function filters** button.
- **Elapsed Time:** Shows the total time spent in the profiled functions (the sum of self-time and time in children).
- **Elapsed Time per call:** Shows the time spent per call.
- **Self Time:** Shows the total time spent only in functions of the module.
- **Self Time Per Call:** Shows the time spent in functions of the module per call.
- **Time In Children:** Shows the time spent in routines called by the module.
- **Calls:** Shows the total number of calls for this function (only in Flat Profile view).

Two display options are available from **Select view:**

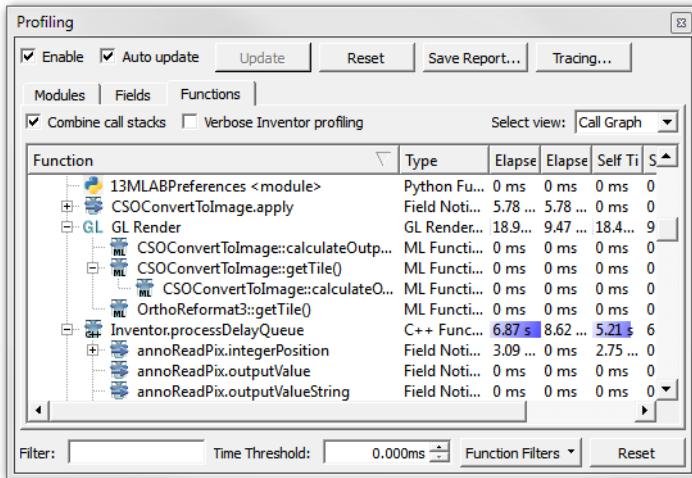
- **Flat Profile:** Shows the functions in a flat list. No hierarchy/dependency is visible. Calls of the same function are automatically bundled and summed up.

Figure 17.7. Profiling Functions as Flat Profile



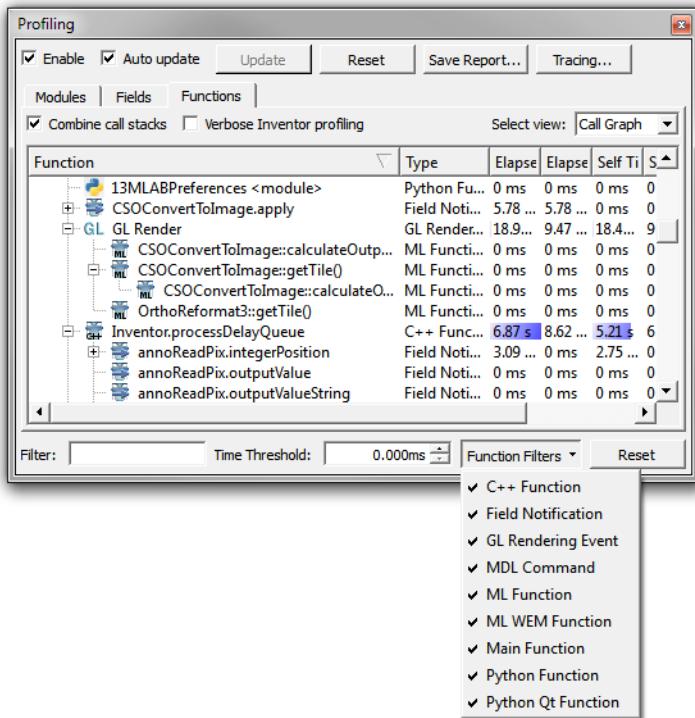
- **Call Graph:** Shows the functions in the hierarchy they were called in. In case of types that should be not displayed due to filtering, these types may still appear if the functions to be displayed are below them in the hierarchy.

Figure 17.8. Profiling Functions as Call Graph



Double-clicking a function navigates to the module the function is called in, even if it is in the network of a macro module.

The option **Reduce call graph (resets profiling)** combines all calls of the same function to one entry in the call graph list.

Figure 17.9. Functions with Filters Visible

When **Filters visible** is selected, the functions filter options are displayed.

Options:

- **Filter:** A text field in which the text to be filtered for can be entered.
- **Time Threshold:** Sets a value below which the row is filtered. This way, more time-consuming functions become more visible.

Buttons:

- **Function Filters:** Opens a list of all function types, to filter which ones should be displayed. Checked types are displayed.
- **Reset:** Deletes entries in **Filter** and **Time threshold** and resets the Function filters so that all types are listed again.

The function types have different icons in the list:

Table 17.1. Function Type Icons

Type	Icon
ML call (typically ML image)	
ML WEM call	
Field call (Field notification)	
Python call	
Python Qt call	
Open GL call	

Some special functions:

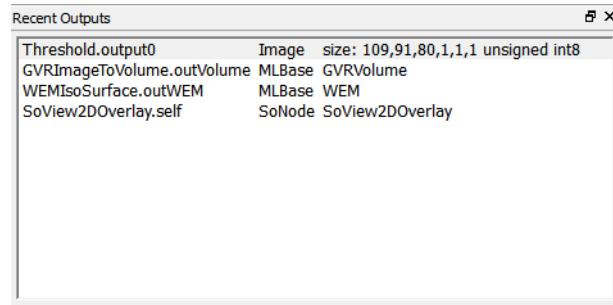
- **main**: pseudo function that is active when profiling is enabled.
- **__tmpScriptHandler**: generated Python functions for MDL inline code (“py:some python code”).
- **ScriptFile <inline script definition>**: definition of inline script functions.
- **PyModuleName <module>**: executed code on Python module level, which is not inside any function; typically called when importing Python modules.
- **<python qt wrapper>**: PythonQt wrappers of Qt functions.

Chapter 18. Recent Outputs

The **Recent Outputs** view shows a list of the recently selected input or output connectors. If the module with the listed connector is removed from the network, its connector is also removed from this list. The last clicked recent connector is sorted to the top of the list.

Each list entry features the context menu of the connector. A list entry in the view can be selected; the content of the selected item is shown in the [Chapter 15, Output Inspector](#) if that view is open. Selecting an item does not make it the most recent, so that the list is not re-sorted on selecting an entry.

Figure 18.1. Recent Outputs



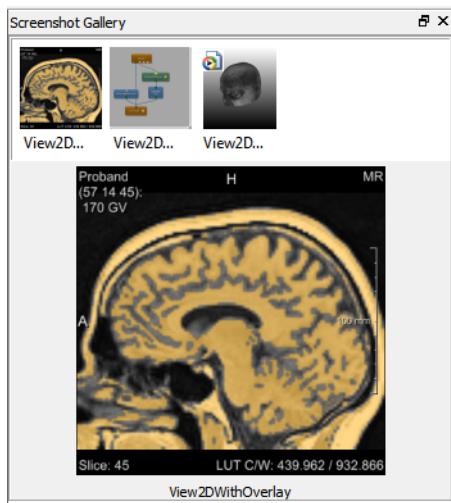
Chapter 19. Screenshot Gallery

19.1. Screenshot Gallery

The **Screenshot Gallery** maintains the screenshots and movies made with MeVisLab's viewers. The screenshot gallery offers a preview of all screenshots and movies, as well as the possibility to show an enlarged version, or to copy, rename, or delete a screenshot or movie.

For showing movies, additional video software may be necessary.

Figure 19.1. Screenshot Gallery



To capture a screenshot, select any viewer and press **F11**. A thumbnail of the screenshot is added to the gallery.

For each screenshot, two files are saved locally in .png format:

- an image file in

```
$USERPATH\screenshots\screenshot_<ModuleName>_<number>.png
```

- a thumbnail file in

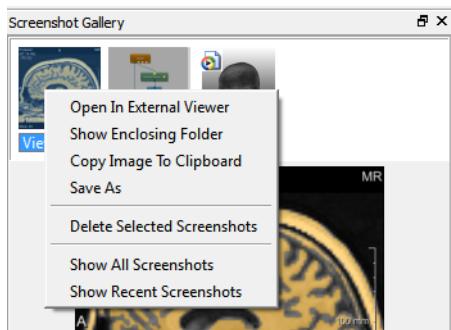
```
$USERPATH\screenshots\screenshot_<ModuleName>_<number>.png.thumb.png.
```

The path for screenshots can be changed in the Preferences, see [Section 4.3.5, “Preferences — Paths”](#).

To view a larger version of the screenshot, click it. The screenshot is displayed below the gallery. Its size depends on the available space.

19.2. Screenshot Gallery Context Menu

In the **Screenshot Gallery**, a context menu is available. To open it, right-click the thumbnail.

Figure 19.2. Screenshot Gallery Context Menu

Open In External Viewer: Open the selected screenshot in the default viewer for .png format or the selected movie in the default player for the video format (the viewer can be set in the Preferences, see [Section 4.3.4, “Preferences — Supportive Programs”](#)). Alternatively, double-click the thumbnail.

Copy To Clipboard: Copy the screenshot or movie to the clipboard.

Save As: Save the screenshot or movie under another name and path.

Delete Selected Screenshots: Delete the selected screenshots and movies from hard drive. To select more than one screenshot, use the platform's standard features. (For example, on Windows use **SHIFT+click** for selecting continuous and **CTRL+click** for selecting multiple screenshots.)

Show All Screenshots: Show all screenshots and movies that are saved in the mevislabscreenshots folder.

Show Recent Screenshots: Show all screenshots and movies of the current day.

19.3. Movies in the Screenshot Gallery

To add movies, a module that supports a movie output needs to be available. For 2D, movies can be recorded for moving through slices (for example in the CineMode of the `View2D` module) by combining this with a Viewer.

For a 3D example, open the `SyncroView2DExample` network, add the `View3D` module and connect it with an `OrthoReformat3` connector. Open the panel of the `View3D` module and click the Advanced tab to find the movie recording settings. Record a movie and then click **Create Movie** so that the output is generated.

Movies are saved in the same folder as screenshots, as `movie_<name>_<number>. <videoformat>` and `screenshot_<name>_<number>. <videoformat>. thumb.png`.

The final result will be displayed as a thumbnail in the **Screenshot Gallery** with a video symbol in the top left corner. Click it once for a small preview in the gallery or double-click it to open it in a video viewer (for example Windows Media or Quicktime player).

Chapter 20. Scripting Console

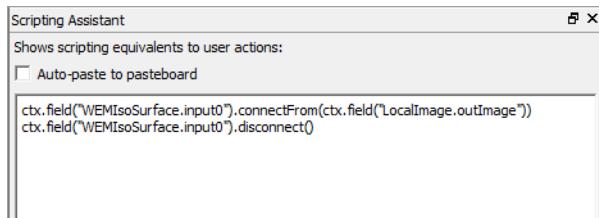
General scripting console for testing Python without any meaningful network or module context.

If you want to test scripting in the context of a certain module, either use the [Section 4.7.1, “Show Scripting Console”](#) or open a scripting console from the module’s context menu ([Section 3.9.1, “Module Context Menu”](#)).

Chapter 21. Scripting Assistant

The Scripting Assistant is a useful tool which translates actions (like (dis-)connecting modules or parameter fields, setting parameter values) into the corresponding scripting commands.

Figure 21.1. Scripting Editor

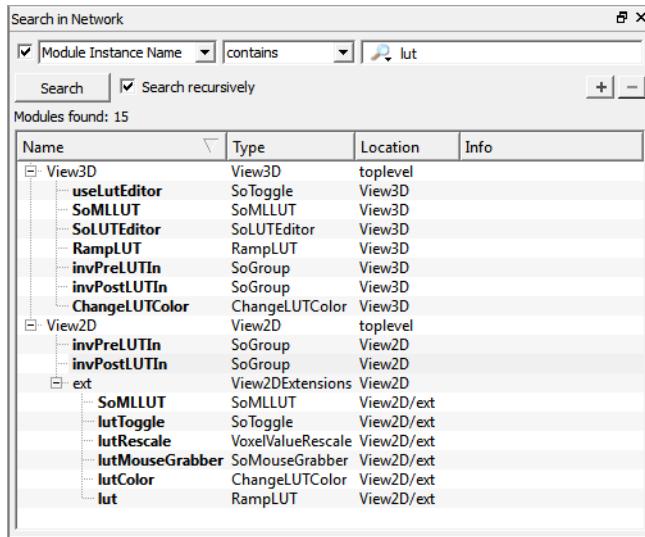


Auto-paste to clipboard: If selected, the Python code resulting from an interaction in the network is automatically pasted to the clipboard, ready to be used in other editors.

Chapter 22. Search in Network

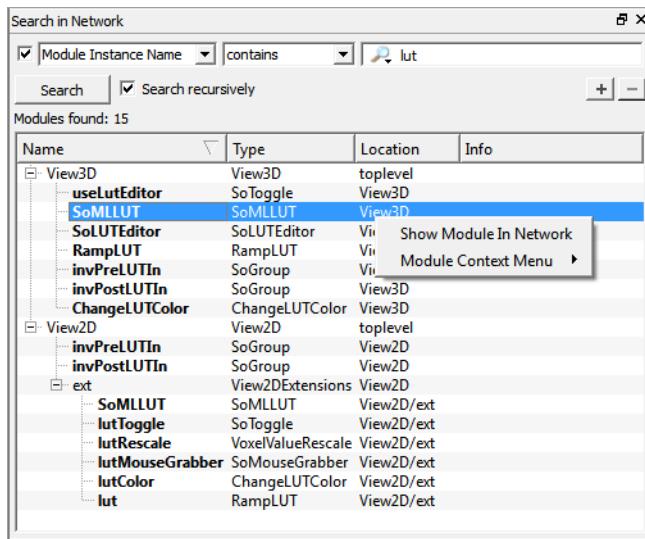
Searches for some named item in the current network. Optionally, the search is applied recursively so that it also searches in macro modules.

Figure 22.1. Scripting Editor



A listed item has a context menu. Its first entry is an option to show the item in the network. If chosen, the network is opened, the module is highlighted, and the highlighted module is centered. Alternatively, a listed item can just be double-clicked to highlight and show it.

Figure 22.2. Scripting Editor

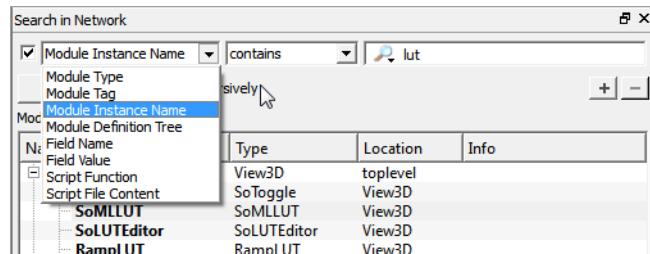


This search can search for different kinds of items. For some of the options, additional choices are available.

- **Module Type:** searches in the modules' type strings
- **Module Tag:** searches in the modules' meta tags
- **Module Instance Name:** searches in the modules' instance name
- **Module Definition Tree:** searches in the modules' definition tree for a tag with the given value

- **Field Name:** searches in the fields' names
- **Field Value:** searches in the fields' values
- **Script Function:** searches in the (Python) scripting functions
- **Script File Content:** searches in the module's GUI definition files

Figure 22.3. Scripting Editor



Chapter 23. Search in Documentation

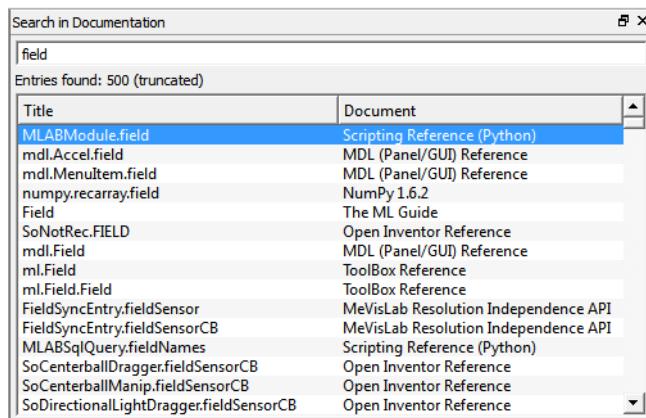
With the **Search in Documentation** View, the online documentation of MeVisLab can be searched (including all Doxygen references, all DocBook books, the OpenInventor reference, Python, and NumPy). The search starts immediately with each entered key (incremental search).



Note

As the index is read upon calling the search for the first time, the first search might take longer than expected. Further searches are fast.

Figure 23.1. Search in Documentation



Click the entry to open the linked documentation in a browser window.

The search works case-insensitive. However, correct spelling is preferred so different best match results will be received for “field” and “Field”.

The search is index-based. When using a local repository, the documentation needs to be built locally to generate the index databases.



Note

Selections in the **Search in Documentation** View are persistent and set as default the next time the View is used.

See the following example screenshots for some possible search terms:

Figure 23.2. Search in Documentation — ML Example

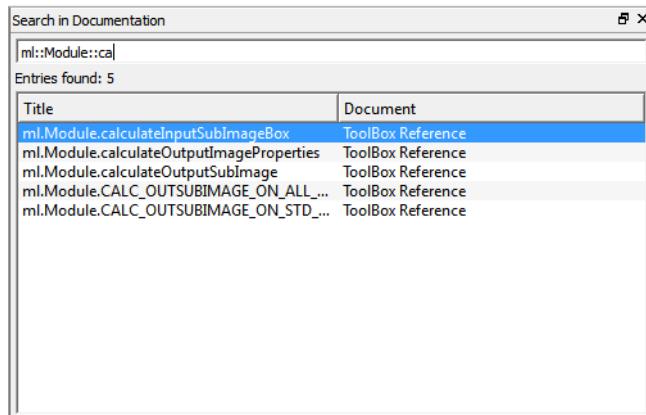
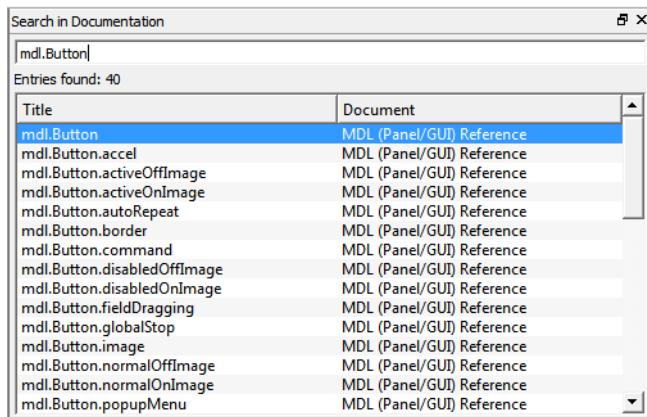
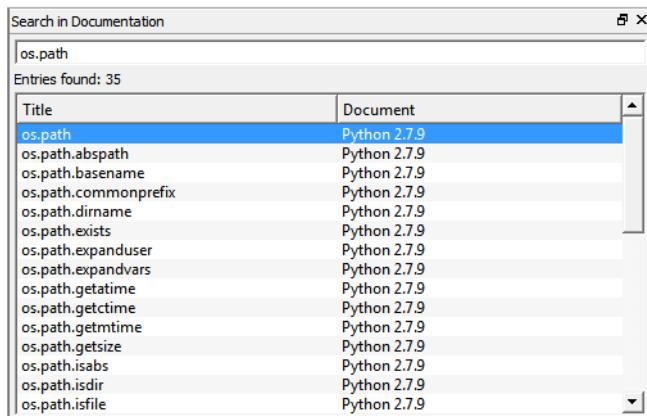
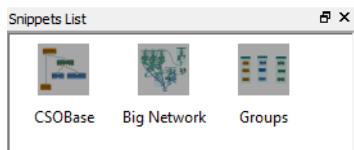


Figure 23.3. Search in Documentation — MDL Example**Figure 23.4. Search in Documentation — Python Example**

Chapter 24. Snippets List

The **Snippets List** View allows reusing often used modules and network snippets. Unlike the normal copy and paste of selections, the snippets are saved and available in future sessions.

Figure 24.1. Snippets List



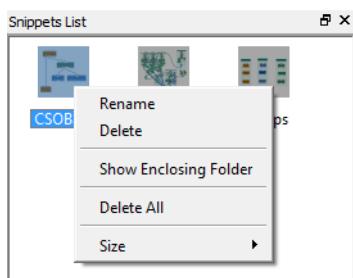
To add modules/networks from the workspace to the snippet list, select them, right-click and choose **Add Selection To Snippets List** from the context menu. Enter a snippet name and click OK to save the snippet.

To add snippets to a network, either double-click the thumbnail to insert the snippet in the middle of the workspace, or drag the thumbnail from the snippets list to the designated position.

The context menu offers the following options:

- **Rename:** Opens a dialog to rename the snippet.
- **Delete:** Deletes the snippet (with confirmation).
- **Open Enclosing Folder:** Opens the folder where the snippets are stored.
- **Delete All:** Deletes all snippets (with confirmation).
- **Size:** Sets the thumbnail size for all snippet thumbnails (small, medium, or large).

Figure 24.2. Snippets List — Context Menu



Multiple snippets can be de-/selected by selecting snippets while holding SHIFT

All selected snippets can be deselected by pressing ESC.

The snippets can be re-arranged by dragging them. Note that a dropped snippet will either be inserted as the first or as the last element; snippets cannot be inserted by dropping between other snippets.

Chapter 25. Project Wizard

25.1. Project Wizard Introduction

With the project wizard, packages and modules can be created and added to the MeVisLab packages and module database.



Note

As user packages are necessary to save new modules, all options except for **New Package** are disabled when first using the Project Wizard.

Figure 25.1. Project Wizard (no user packages available)

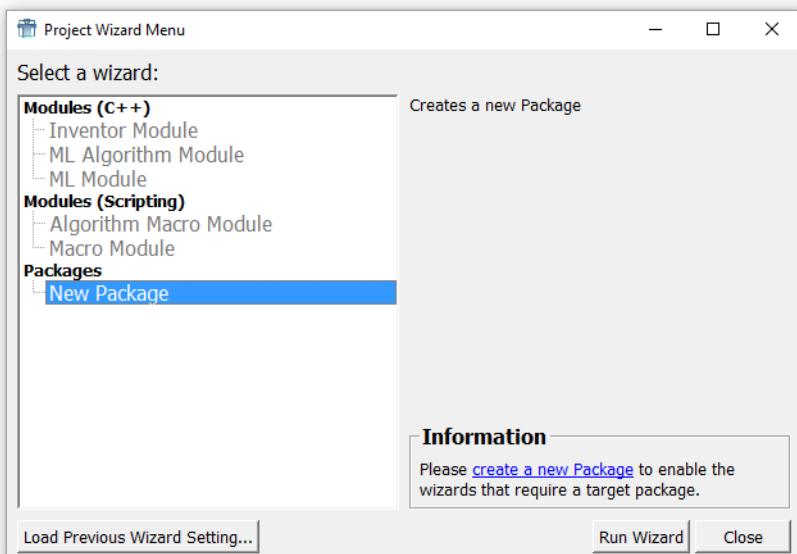
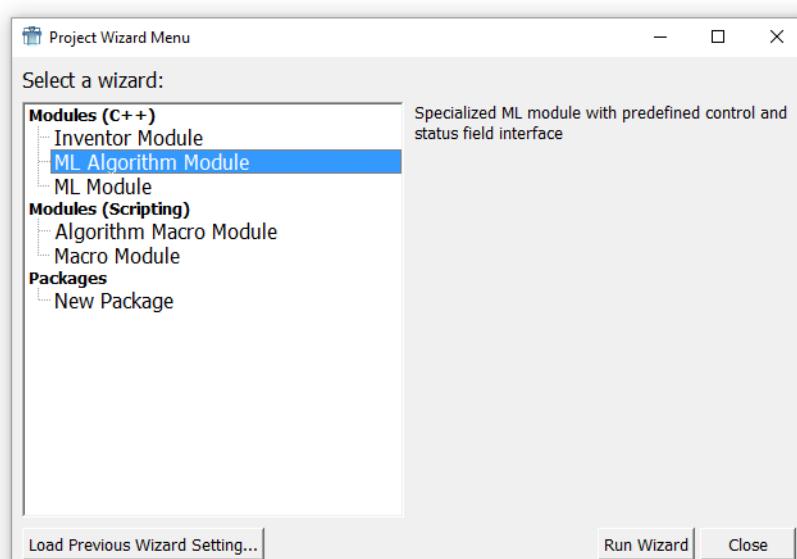


Figure 25.2. Project Wizard (with user packages available)



The following can be created:

- Programmed Modules: Inventor, ML modules, and Algorithm module.
- Modules (Scripting): macro modules and algorithm macro modules for bundling modules and script files
- Packages: for organizing modules in a package structure.

25.2. Modules (C++) Wizard



Tip

For details on using the C++ module creation wizard, see the Getting Started, chapter "Developing ML Modules".

The Wizards for programmed modules of the type Inventor, ML, and WEM module start rather similar, with a first dialog on which the name, package and other descriptors are added.

The Wizard leads through the creation process. Move between the dialogs with **Back** and **Next**. The settings can be saved at any point as .wiz file via **Save Setting**.

Please refer to the Getting Started document for a complete list of the files that are generated by the wizard.

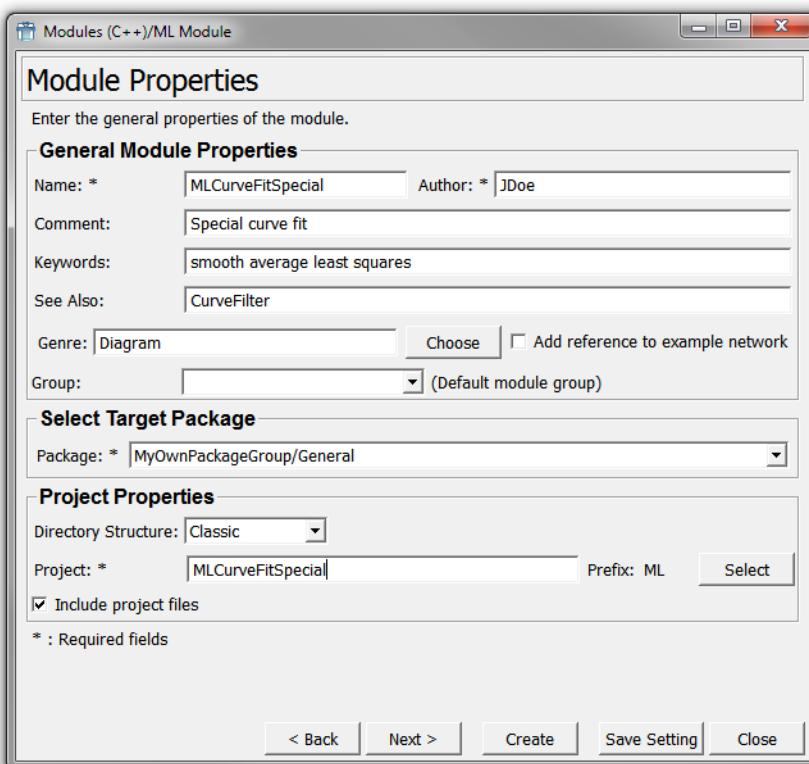


Note

After module creation, the module database has to be reloaded before the new module can be used in a network.

25.2.1. First C++ Module Wizard Dialog

Figure 25.3. First C++ Module Wizard Dialog — ML Module Example



Name

Enter the <ModuleName> here. It has to be a unique name within the MeVisLab module database (including the SDK module database).

Author, Comment, Keywords, See Also, Genre:

Enter descriptors to classify the module within the MeVisLab module database. Author, comment and genre are mandatory entries. See the descriptions of existing modules for inspiration of what to enter here. Errors in the descriptions will be displayed in the Debug Output upon loading the module database.

Add reference to example network

If selected, creates an empty example network <ModuleName>Example.mlab which may be edited later (optional). It is recommended that each module should be completed by an example network to explain its function and usage in an exemplary application.

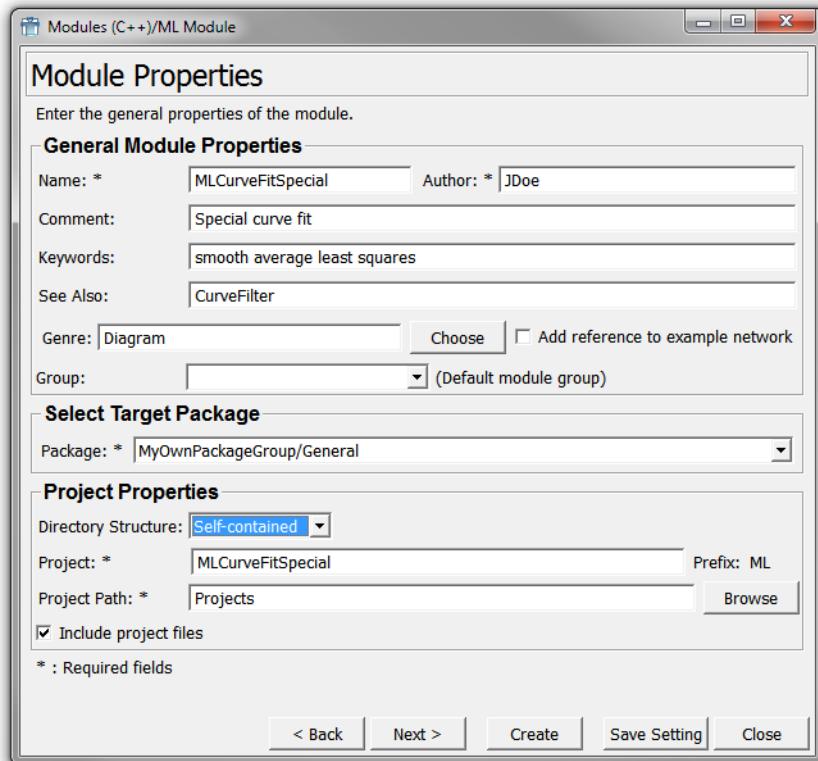
Select Target Package

Sets the Package for the module project. Select one from the list. For more information on Packages, see [Section 25.5, "Packages"](#).

Project Properties**Directory Structure**

- Classic:** the project files are separated into Sources and Modules folders
- Self-contained:** all project files are located in one self-contained folder that makes it easy to move and exchange the whole folder

Figure 25.4. Create an ML Module in a Self-contained Folder



Select

Allows adding the project to an already existing ML project within the selected Target Package. The selected project name is inserted in the field **Project**.

If working without a license file, the project prefix receives a leading underscore, for example “_ML”.

Include project files

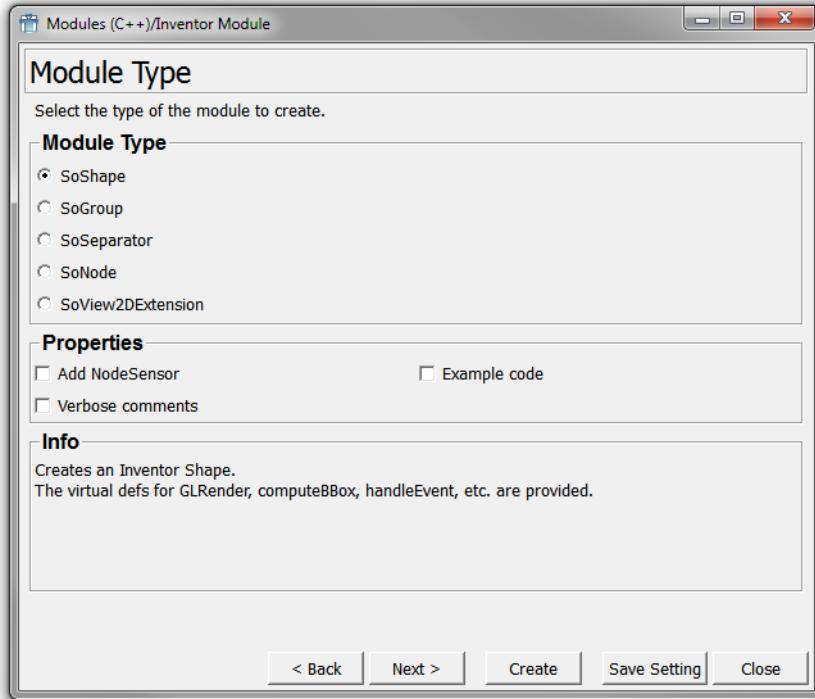
If selected, the following C++ files are also created (in addition to the files listed above): CMakeLists.txt, <ModuleName>System.h, <ModuleName>Init.h, <ModuleName>Init.cpp.

Click **Next** for the next screen, which is specific for each module type available in the Wizard.

25.2.2. Inventor Module

For the first dialog, see [Section 25.2.1, “First C++ Module Wizard Dialog”](#).

Figure 25.5. Inventor Type



The Open Inventor module has to be assigned a module type. Select a type to see additional comments in the Info area.

- **SoShape:** Creates an Inventor shape, representing a geometric object, for example cones, spheres, cubes and alike. Functions for GLRender, handleEvent, etc. are provided.
- **SoGroup:** Creates an Inventor group, which offers more than one input.
- **SoSeparator:** Creates an Inventor separator. In addition to a SoGroup, the OpenGL state is pushed and popped.
- **SoNode:** Creates a basic Inventor node.
- **SoView2DExtension:** Creates a View2D extension which renders in Open GL on a 2D viewer and processes user interaction (mouse, keyboard).

- **Add Node Sensor:** Adds a `SoNodeSensor` that will trigger on any field changes and thus allows for user interaction. The necessary code is provided to fill in the gaps.
- **Verbose comments:** Adds verbose comments that you might want to add when first trying out this Wizard.
- **Example code:** Adds additional example source code.
- **Has group inputs:** (not for all Open Inventor module variants): Sets whether the MeVisLab GUI allows multiple `SoNode` inputs.

Click **Next** for the next screen, see [Section 25.4, “Module Field Interface”](#),

25.2.3. ML Module

ML modules generate compilable C++ code and platform-dependent project files to implement a new image processing template module in MeVisLab. Define the number of image inputs and outputs of the module as well as module parameters (fields). After compilation and reloading of the module database via **Extras → Reload Module Database (Clear Cache)**, you can create the new module on your network document. If you created the module in an existing Project (DLL) you may have to restart MeVisLab (use **File → Restart With Current Networks**).



Note

The properties available in this dialog are also described in the ML Guide, chapter “Deriving Your Own Module From Module”. The Project Wizard simply makes it easier to implement these methods.

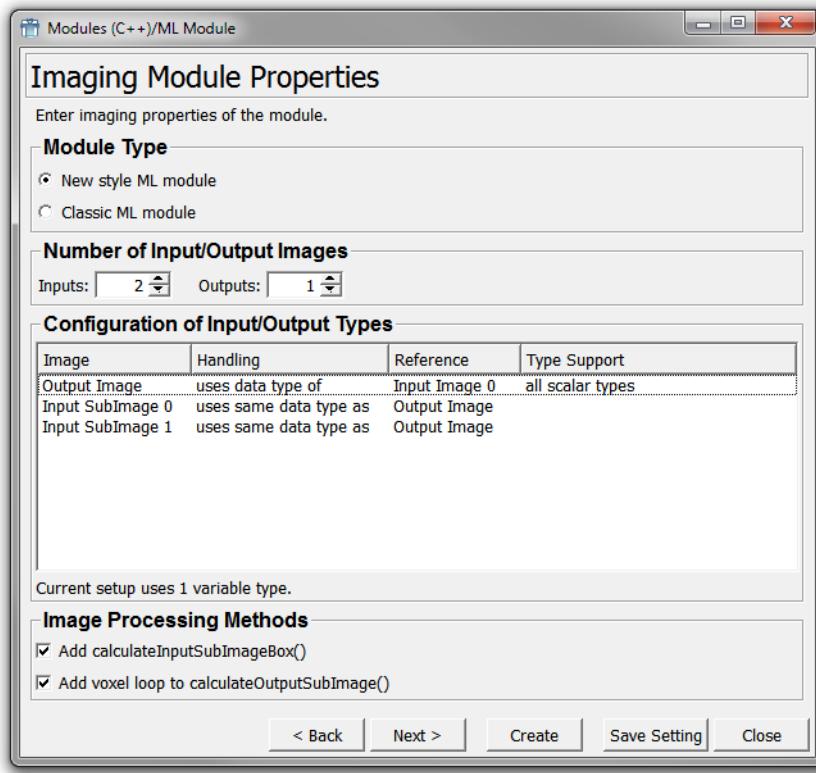
As of MeVisLab version 2.1, two implementation styles of ML modules are available, new and classic. The difference is that in the classic ML style, the standard assumption was that input and output image were of the same data type, and changes of the data type had to be programmed manually. In the code, this was done via `Template` that had the data type as parameter, and during compilation, the function `ML_CalculateOutputSubImage` was compiled as routine for all data types.

In contrast to this, the new style ML module does not use the `Template` mechanism but uses handler classes for the input/output routine. The wizard supports the handling configuration already in the GUI.

For the first dialog, see [Section 25.2.1, “First C++ Module Wizard Dialog”](#).

25.2.3.1. ML Module (New Style)

Figure 25.6. Imaging Module Properties (New Style)



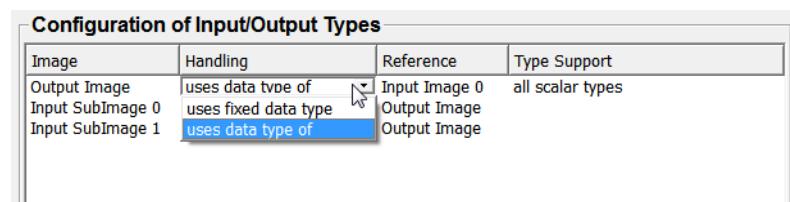
Number of Input/Output Images

Inputs, Outputs: Sets the number of image processing inputs and outputs.

The handling of the datatypes of output and input image is done in via the drop-down lists in the window.

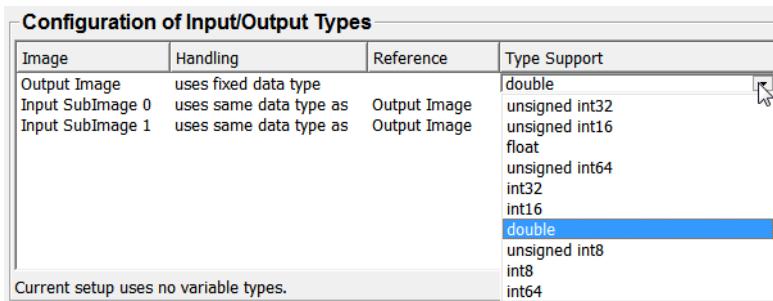
In the following figures, the number of **Inputs** is set to two so that the extended **Handling** and **Reference** options are better visible.

Figure 25.7. New Style ML Module

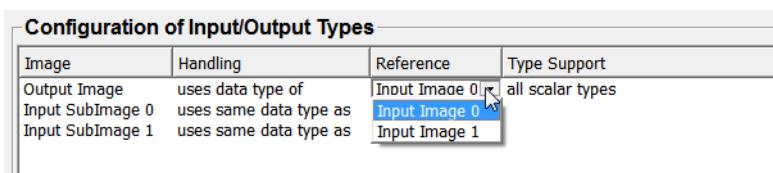
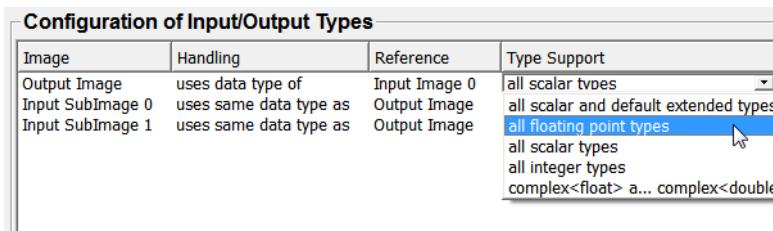


Two options are available for the handling of the output image:

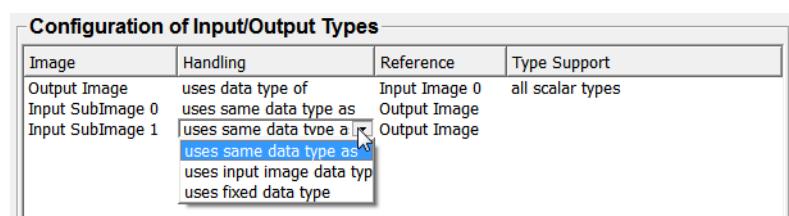
- **uses fixed datatype:** Sets the output datatype to a fixed format that has to be entered in the column **Type Support**, see [Figure 25.8, "New Style ML Module — Uses Fixed Datatype"](#).

Figure 25.8. New Style ML Module — Uses Fixed Datatype

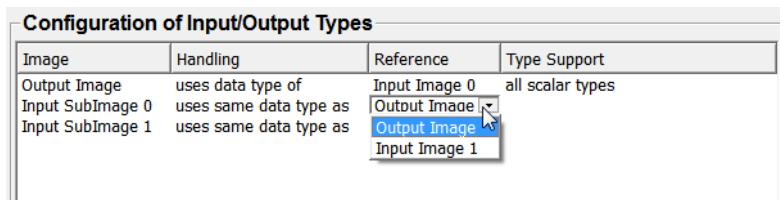
- **uses datatype of:** Sets the output datatype to the same format as the selected input image. In addition, the supported types can be limited to certain types in the column **Type Support**.

Figure 25.9. New Style ML Module — Uses Datatype Of Input Image**Figure 25.10. New Style ML Module — Entering The Supported Types**

Three options are available for the handling of the input image:

Figure 25.11. New Style ML Module — Configuring The Input Handling

- **uses fixed datatype:** Sets the input image datatype to a fixed format that has to be entered in the column **Type Support**, identical to the selection for the output image, see [Figure 25.8, “New Style ML Module — Uses Fixed Datatype”](#).
- **uses input image datatype:** Keeps the input image datatype. In addition, the supported types can be limited to certain types in the column **Type Support**.
- **uses same datatype as:** Sets the input image datatype to the same format as the selected image, see [Figure 25.9, “New Style ML Module — Uses Datatype Of Input Image”](#).

Figure 25.12. New Style ML Module — Uses The Same Datatype As

Below the configuration of the input/output types, two image processing options are available:

Image Processing Methods

Add calculateInputSubImageBox(): If enabled, the method `calculateInputSubImageBox()` is added. It is required if the calculation of an output page requires another (smaller or larger) image region from the input than the one of the output page.

Add voxel loop to calculateOutputSubImage(): If enabled, an example loop is implemented which reads all voxels from input 0 (if available) and copies them to the output page. It simplifies loop-based implementations of `calculateOutputSubImage()`.

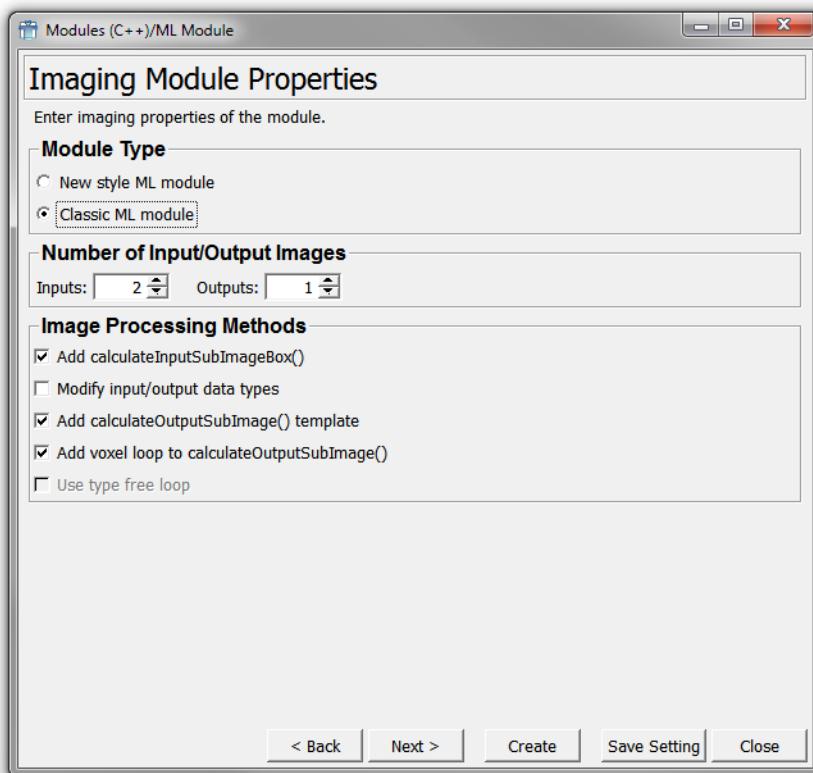
Click **Next**.

25.2.3.2. ML Module (Classic Style)



Note

For new modules, it is recommended to use the new style. The classic style implementation is offered as legacy option. For an explanation of new style versus classic style, see [Section 25.2.3, “ML Module”](#).

Figure 25.13. Imaging Module Properties (Classic Style)

Number of Input/Output Images

Inputs, Outputs: Sets the number of image processing inputs and outputs.

Image Processing Methods

Add calculateInputSubImageBox(): If enabled, the method `calculateInputSubImageBox()` is added. It is required if the calculation of an output page requires another (smaller or larger) image region from the input than the one of the output page.

Modify input/output data types: If enabled, adds demo code for the modification of input/output image data types in classic style. (For new style ML module, the modification of data types is supported in the methods and the GUI, see [Section 25.2.3.1, “ML Module \(New Style\)”](#).)

Add calculateOutputSubImage() template: If enabled, a template function is added for the implementation of the virtual method `calculateOutputSubImage()`. This is the typical way to implement the algorithm independent of the voxel type.

Add voxel loop to calculateOutputSubImage(): If enabled, an example loop is implemented which reads all voxels from input 0 (if available) and copies them to the output page. It simplifies loop based implementations of `calculateOutputSubImage()`.

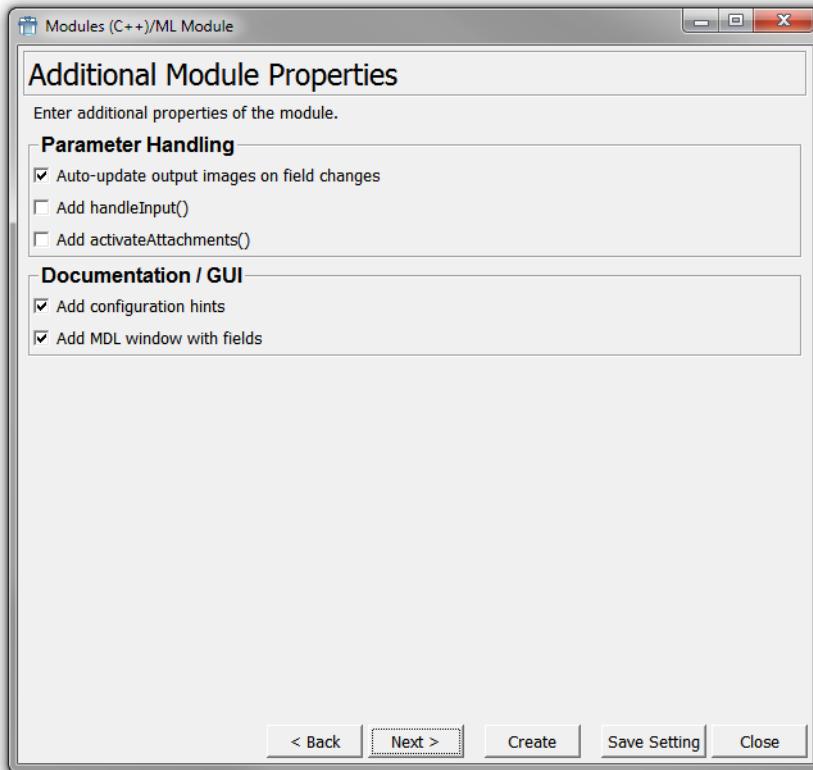
Use type free loop: (Only active for certain selections) If enabled, an example loop is implemented which reads all voxels from input 0 (if available) and copies them to the output page. It is implemented without using any voxel types, but only copying the image memory voxel by voxel. Useful if the algorithm is type-independent or specific for a known voxel type.

Click **Next**.

25.2.3.3. Additional ML Module Properties

For both new and classic style ML modules, the same additional options are available:

Figure 25.14. Additional ML Module Properties



Parameter Handling

Auto-update output images on field changes: If enabled, code is added to the `handleNotification()` method that causes a touching of the output image field(s) on changes of any input or parameter field.

Add handleInput(): If enabled, the method `handleInput()` is added. It is required if the module shall be able to operate with optionally disconnected or invalid image inputs.

Add activateAttachments(): If enabled, the method `void activateAttachments()` is added. It is called after module clones or reloads and is needed to update the internal module state/members after reload or clone operations if it depends on field values.

Documentation / GUI

Add more detailed comments: If enabled, more comments to the generated code, for example possible parameters, their effect or details about the methods or functions to be implemented are added to the code.

Add configuration hints: If enabled, hints for inplace, bypass, voxel type and multi-threading support are added to the code.

Add MDL window with fields: If enabled, a window section with all fields is added to the MDL definition file of the module.

Click **Next** for the next screen, if you want to add fields, see [Section 25.4, “Module Field Interface”](#). Otherwise, click **Create** to create the new ML module.

25.2.3.4. ML Module — Created Files

When creating the new ML module, A number of files are generated, some of which have the same purpose in another surrounding/operating system:

In the path: <Package>/Modules/<ModuleType>/<ModuleName>

- <ModuleName>.def: MeVisLab definition file

In the path: Modules/<ModuleType>/<ModuleName>/networks

- testExample.mlab: Example network (template)

In the path: <Package>/Sources/<ModuleName>

- <ModuleName>.vcxproj: Visual C++ project file
- <ModuleName>System.h:
- <ModuleName>Init.h: C++ header file
- <ModuleName>Init.cpp: C++ file
- <ModuleName>.bat: Windows batch file for creating the module project for the development environment (for example VC10).
- <ModuleName>.sh: Shell script for Linux/Mac OS X (equivalent to batch file).
- CMakeLists.txt: MeVisLab project file, see CMake documentation.

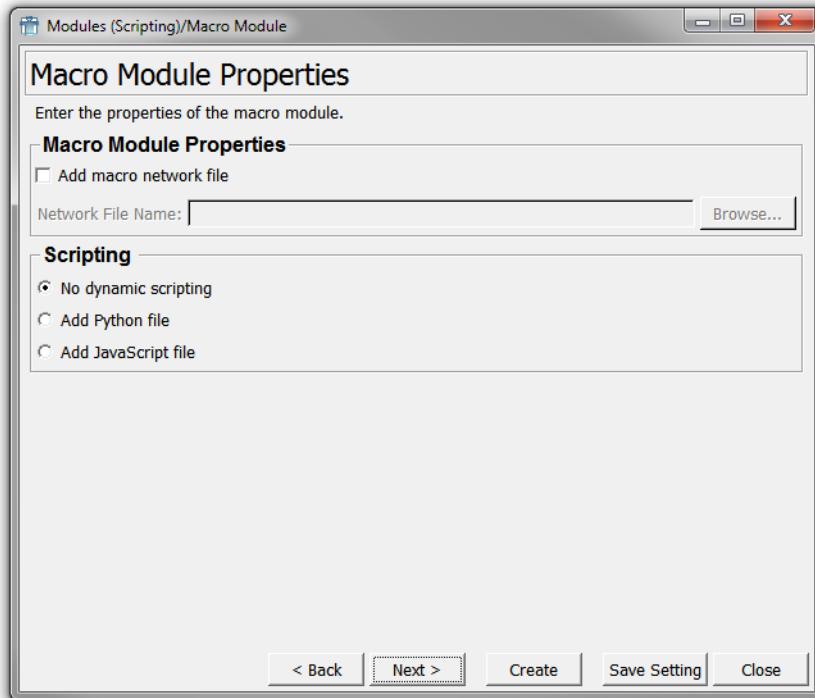
25.3. Modules (Scripting) Wizard

This wizard creates functioning macro modules by generating the template files <ModuleName>.def and <ModuleName>.script (optional: <ModuleName>.mlab and <ModuleName>.py).

For the first dialog, see [Section 25.2.1, “First C++ Module Wizard Dialog”](#).

In the second dialog, the module properties can be set and a local macro can be chosen as starter for the macro module.

Figure 25.15. Project Wizard



Add Macro Network File

Enable if the macro module shall encapsulate a module network. Either generates an empty `<ModuleName>.mlab` document or uses an existing one.

Add Python

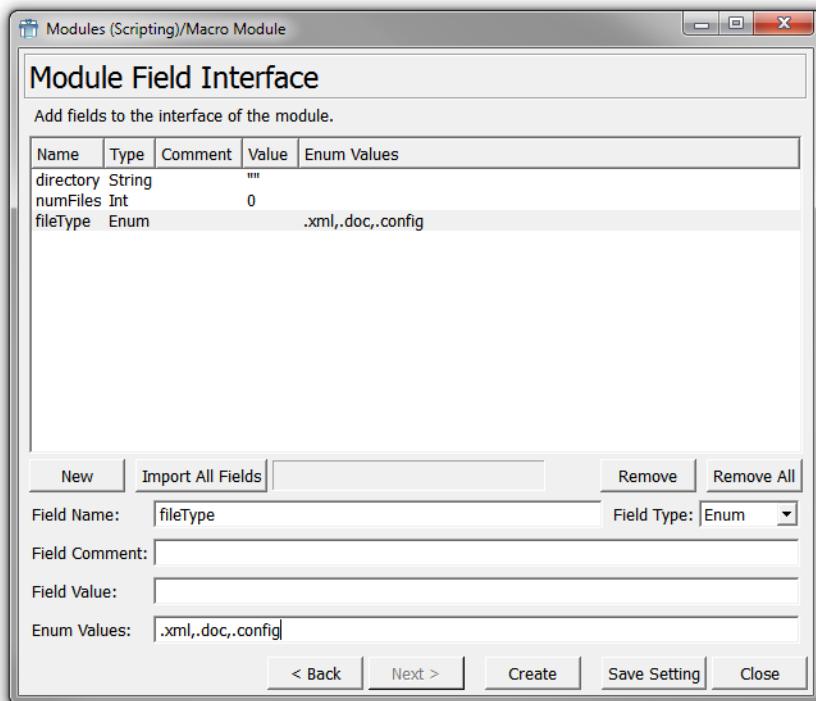
Creates an empty file (`<ModuleName>.py`) which is included in the module's definition file `<ModuleName>.script`. Used to define script commands embedded in the MDL script code to implement dynamic user interfaces.

Click **Next** for the next screen, if you want to add fields, see [Section 25.4, “Module Field Interface”](#), Otherwise, click **Create** to create the new macro module.

After module creation, the module database has to be reloaded before the new module can be used in a network.

25.4. Module Field Interface

Add fields to the interface of the module.

Figure 25.16. Module Field Interfaces**New**

Adds a new field. Click first and then edit the field parameters.

Import All Fields

Imports all fields of all modules of the internal network.

Remove

Removes the selected field.

Remove all

Removes all fields.

For each field, the following may be entered, depending on the field type:

Field Name

Gives the field name. Has to be unique in the module.

Field Type

Gives the field type. Available types are String, Enum, Bool, Int, Float, Double, Progress, Notify, Base, and SoNode.

Field Comment

Adds a comment to the field, useful for the generated code.

Field Value

Gives the field value.

Enum Values (for enum field only)

Gives the enumerator values. Separate by commas.

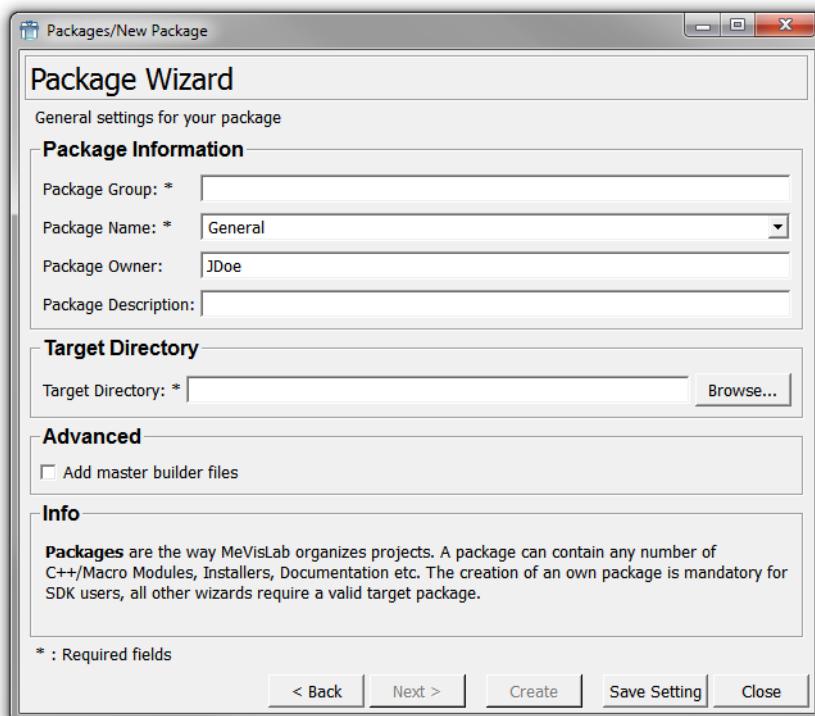
This is the last screen of the Wizards. Click **Create** to create the module.

25.5. Packages

Packages are the way MeVisLab organizes projects. A package can contain any number of C++/Macro Modules, Installers, Documentation, etc. The creation of an own package is mandatory for SDK users, all other wizards require a valid target package.

With the Package Wizard, new packages can be created. For detailed information on the package structure, see the Package Structure documentation.

Figure 25.17. Package Wizard

**Package Group**

Sets the package group in which the package is saved. Enter a name, for example your company or site name.

Package Name

Sets the package name. Select a typical user package name from the list or enter a new package name.

Package Owner

Sets the package owner. Meta description.

Package Description

Sets the package description.

Target Directory

Sets a target directory.

The information entered in the dialog is saved in the `Packages.def` file. The new package is added to the User Package Path, including all subdirectories and files (see the Package Structure documentation). After this, a reload of the MeVisLab module database is necessary to use the new package.

25.6. Example .Wiz File (Inventor Module), indented for a better readability

```
// MDL v1 utf8
wizard = InventorModuleWizard
fields {
    instanceName      = wizard
    moduleName        = SoTest

    author            = JDoe
    genre             = SoGenre
    comment           = ""
    keywords          = sogroup
    seeAlso           = ""
    exampleNetwork    = TRUE

    project           = Sotest
    projectPrefix     = So
    includeProjectFiles = FALSE
    stepTitle         = "Module Field Interface"
    stepInfo          = "Add fields to the interface of the module."
    packageIdentifier = MyPackageGroup/Internal
    storedFieldList   = "fieldName@fieldType@fieldComment@ [...]"
    fieldSelected     = TRUE
    nodeSensor        = FALSE
    verboseComments   = FALSE
    exampleCode       = FALSE
    hasGroupInputs    = FALSE
    type              = SoShape
    typeExtra         = Shape
    fields             = Test
}
```

Chapter 26. MATE

26.1. What is MATE?

MATE is the internal text editor for MeVisLab. MATE is an acronym that stands for **MeVisLab Advanced Text Editor**.

MATE supports the programming languages MDL, Python, and JavaScript, offering auto completion (with a list of suggestions), context-sensitive specific help, syntax highlighting, and indentation.

It also supports HTML, CSS, and XML with simple syntax highlighting.

Additionally, MATE has a built-in debugger for Python scripting, a GUI editor for module panels, and functions as an editor for help files for modules (.mhelp).

For .mhelp, HTML, CSS, and XML, the editor checks the spelling of written text using hunspell.

Besides all this built-in functionality, MATE offers a scripting API to configure MATE on startup by user written Python scripting, as well as adding new features to MATE's GUI by writing Python user scripts, similar to [Section 4.8, “User Scripts”](#)

MATE also offers direct access to a module's panel and automatic panel, and related files via the **Module** menu.

For quick scripting, a scripting console is available.

The following file types are supported:

- MDL files, namely
 - .dat (MeVisLab license files written in MDL)
 - .def (MeVisLab module definition)
 - .mhelp (MeVisLab module help file, see [Section 26.9, “Module Help Editor”](#))
 - .mlab (MeVisLab network)
 - .prefs (MeVisLab preferences file, for packages or other purposes)
 - .script (MeVisLab MDL script)
- Other MeVisLab specific files
 - .mlinstall, .mli (Installer specification used by the ADK add-on)
 - .py (Python)
 - .js (JavaScript)
 - .html (HTML)
 - .css (Cascading Style Sheets)
 - .xml (Extensible Markup Language)
 - .txt (Generic text files)

Open MATE via **File → Show Integrated Text Editor** (to start it without files) or **File → Open File in Integrated Text Editor** (to start it with a file dialog for selecting a file).

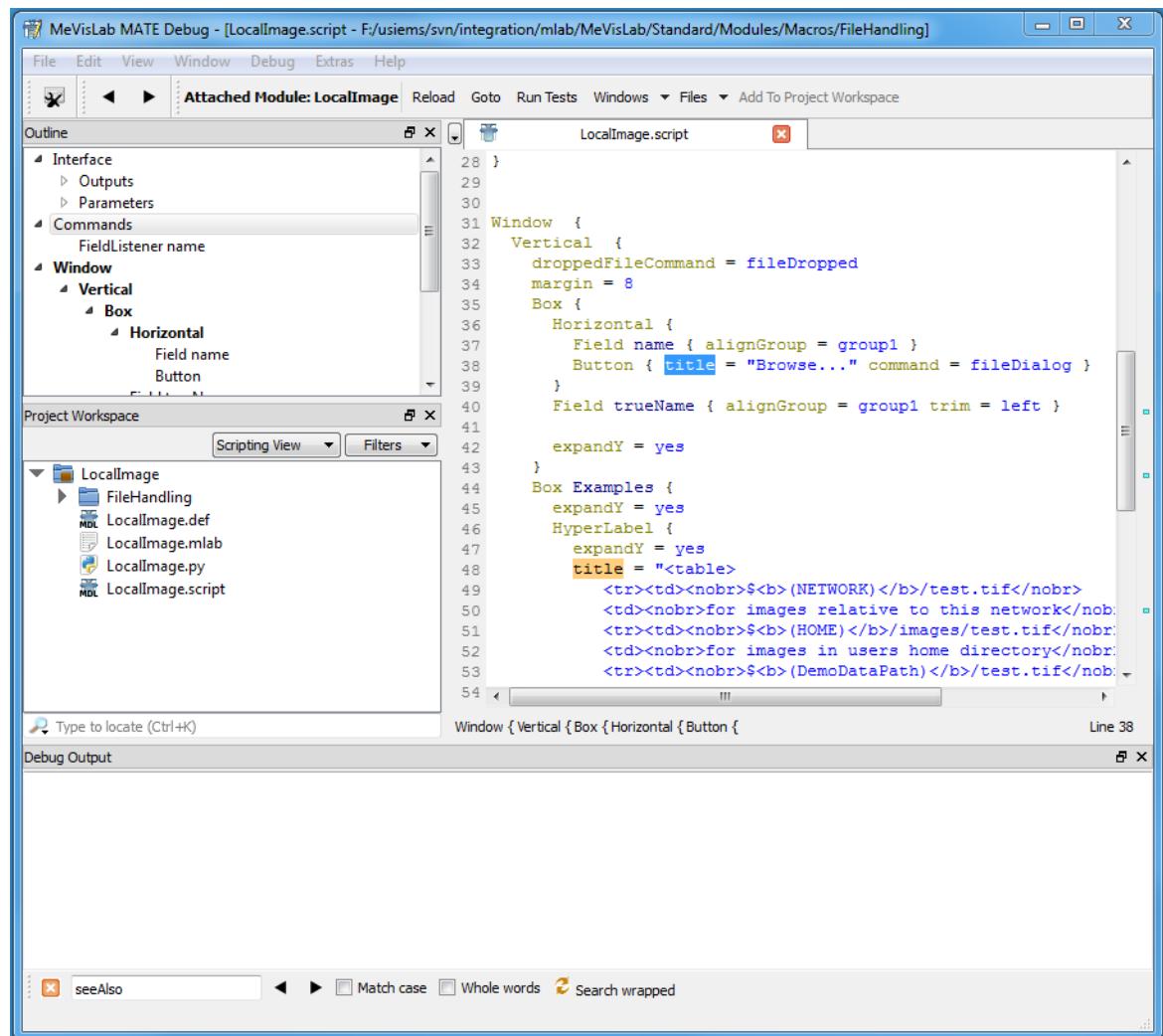
MATE is also used as editor for source code by default (can be changed in the Preferences, see [Section 4.3.4, “Preferences — Supportive Programs”](#)). For example

- for **Related Files** in the context menu of a module, when selecting one of the possibly available `.def`, `.script`, or `.py` files.
- when clicking a link to a license `.dat` file or a module `.def` file in the **Debug Output** of MeVisLab.

MATE runs in a process separate from MeVisLab. This allows using it for Python debugging, see [Section 26.8, “Python Debugger”](#).

26.2. Text Editor User Interface

Figure 26.1. User Interface



The user interface offers the following areas:

- the menu bar
- the **Attached Module** menu (can be switched off in the **Views** submenu)

- the **Outline** area with a list of related modules (can be switched off in the **Views** submenu)
- the **Edit** area, with tabs for open files
- the **Project Workspace** area, with a list of projects which are collections of files belonging together (can be switched off in the **Views** submenu)
- the **Debug Output** area, with the same information as in MeVisLab (see [Chapter 8, Debug Output](#), can be switched off in the **Views** submenu)
- the Find menu for incremental and normal search (can be switched off in the **Views** submenu)

As of MeVisLab 2.2, another two specialized MATE user interface configurations are available, see [Section 26.9, "Module Help Editor"](#) and [Section 26.8, "Python Debugger"](#).

As of MeVisLab 2.3, MATE also offers functionality of a session manager and a GUI editor, see [Section 26.10, "Session Management"](#) and [Section 26.12, "GUI Editor"](#).

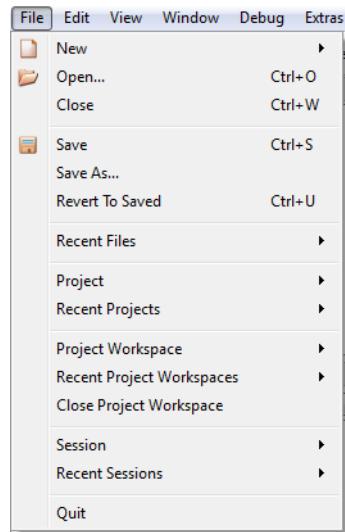
As of MeVisLab 2.7, MATE offers a project workspace, which is superior to the session manager of previous versions, see [Section 26.11, "Project Workspaces"](#). There is also a rudimentary integration with the Pylint Python code checking tool, see [Section 26.14, "Pylint Integration"](#).

26.3. Menu Bar

In the menu bar, the following entries are offered.

In the **File** menu, **New**, **Open**, **Close**, **Save**, **Save As**, **Revert To Saved**, **Recent Files**, **Project**, **Recent Projects**, **Project Workspace**, **Recent Project Workspaces**, **Close Project Workspace**, **Session**, **Recent Session**, and **Quit** are available.

Figure 26.2. MATE File Menu

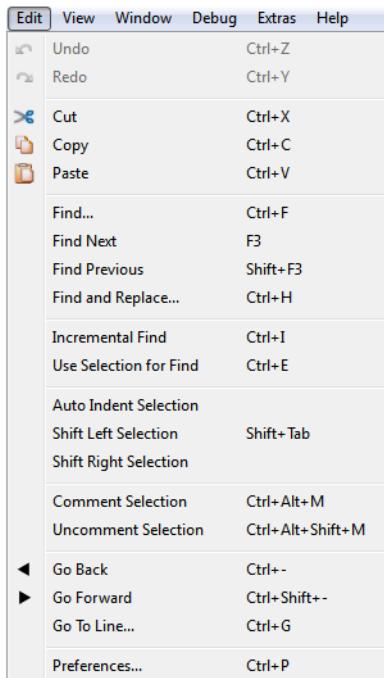


In the **Edit** menu, **Undo**, **Redo**, **Cut**, **Copy**, **Paste** are available. In addition to these standards, a number of options for searching and code formatting are available. In the search, regular expressions may be used.

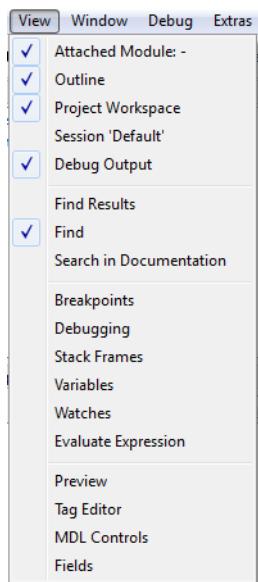
- **Find:** Opens a **Find** dialog. Enter search term.
- **Find Next:** Finds next entry, search direction down.

- **Find Previous:** Finds next entry, search direction up.
- **Find and Replace:** Opens a **Find and Replace** dialog. Enter old and new term.
- **Find Incremental:** Opens an independent search bar at the bottom of the MATE screen and starts searching immediately while the term is entered.
- **Use Selection for Find:** Uses only the currently selected text portion as input for the Find dialog.
- **Auto Indent Selection:** Arranges the selection according to indentation per level. Alternatively, **TAB** may be used.
- **Shift Left Selection:** Shifts the selection to the left.
- **Shift Right Selection:** Shifts the selection to the right.
- **Comment Selection:** Comments the selection (available for Python, MDL, and JavaScript).
- **Uncomment Selection:** Uncomments the selection (available for Python, MDL, and JavaScript).
- **Go Back:** Jumps to the last cursor position.
- **Go Forward:** Jumps to the next cursor position.
- **Go To Line...:** Opens a dialog to enter a line number in the currently active document to set the cursor to.
- **Preferences:** Opens the preferences for the MATE editor, see [Section 26.7, “Preferences”](#).

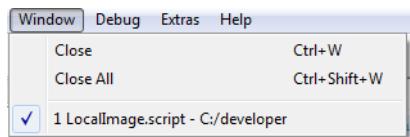
Figure 26.3. MATE Edit Menu



In the **View** menu, select the user interface areas for display: **Attached Module** toolbar, **Outline** area, **Project Workspace** area, **Session** area, **Debug Output** area, the search functions **Search in Documentation** area, **Find Results** area, **Find** toolbar, the Python debugging options **Stack Frames** area, **Variables** area, **Watches** area, **Evaluate Expression** area, **Breakpoints** area, **Debugging** toolbar, and the GUI editor options **Preview** area, **Tag Editor** area, **MDL Controls** area, and the **Fields** area.

Figure 26.4. MATE View Menu

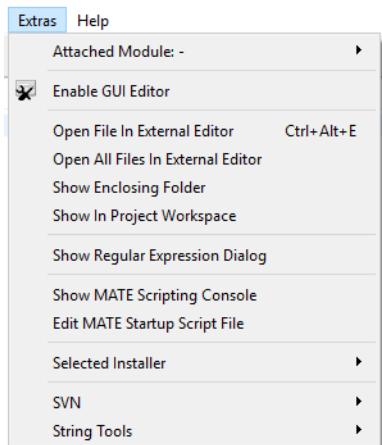
In the **Window** menu, the files in the **Edit** area can be closed and selected: **Close**, **Close All** (you will be asked if you want to save changes).

Figure 26.5. MATE Window Menu

In the **Debug** menu, debug options are available, see [Section 26.8, "Python Debugger"](#).

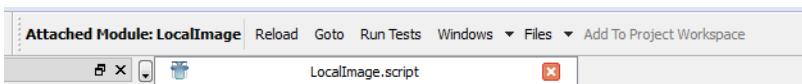
In the **Extras** menu, the following options are available:

- **Attached Module:** see [Section 26.4, "Module Menu"](#).
- **Enable GUI Editor:** see [Section 26.12, "GUI Editor"](#).
- **Open File In External Editor:** Opens file in the default editor (may be set in [Section 4.3.4, "Preferences — Supportive Programs"](#)).
- **Open All Files In External Editor:** Opens all files in the default editor (may be set in [Section 4.3.4, "Preferences — Supportive Programs"](#)).
- **Show Enclosing Folder:** Opens the folder where the currently active document is in.
- **Show Regular Expression Dialog:** Shows a dialog where the user can test regular expressions with own sample texts.
- **Show MATE Scripting Console:** Opens the MATE scripting console, see [Section 26.13, "Scripting"](#).
- **Edit MATE Startup Scripting File:** Opens the MATE startup scripting file for editing, see [Section 26.13, "Scripting"](#).
- **SVN:** Contains a submenu with subversion commands that are applied to the currently open file. This needs a svn command line executable available from the PATH environment variable.
- **String Tools:** Contains a submenu with assorted string manipulation tools. The scope of these operations usually is the selected text.

Figure 26.6. MATE Extras Menu

26.4. Module Menu

MATE communicates with a running MeVisLab instance to get information about a currently edited module, which is used for tasks like auto-completion. For this the MeVisLab instance might create the required module invisible to the user. The module menu allows for a direct handling of the attached module. If no attached module is available yet, it has to be created (loaded). For this, select the module in the Outline area and then click **Create**.

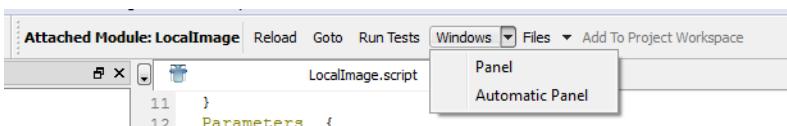
Figure 26.7. MATE Module Menu — Without Attached Module**Figure 26.8. MATE Module Menu - With Attached Module**

Once a module is attached, more options are available.

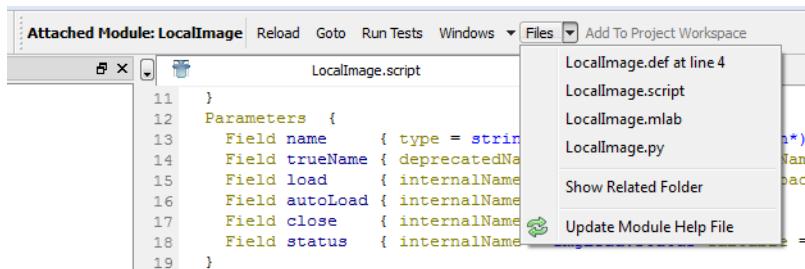
Click **Reload** to reload the module information. This is useful if the module files were changed (for example `DLL`, `.script`, or `.def` file).

Click **Goto** to select and center the associated module in MeVisLab. This is only available if the file has been opened from the context menu of an existing module in MeVisLab.

Run Tests will run the associated tests of an associated module in MeVisLab. If the currently edited file belongs to a test case instead this test case will be executed.

Figure 26.9. MATE Module Menu — Windows Submenu

The **Windows** submenu has the same options and effects as the module context menu entry **Show Windows** in the MeVisLab user interface. The panels or viewers will be opened in MeVisLab.

Figure 26.10. MATE Module Menu — Files Submenu

In the **Files** submenu, all source files related to the attached module are listed. Select one to open it in the editor.

Note



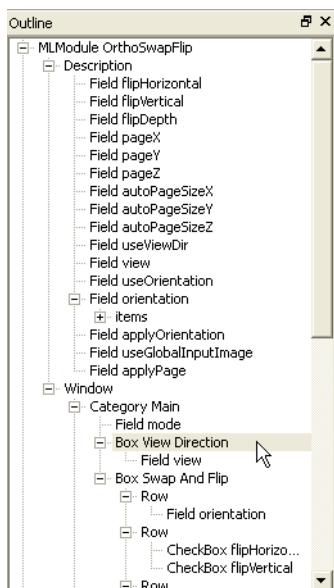
Not all related files may be available as source, for example compiled modules delivered with the SDK.

Add To Project Workspace adds the files of the module to the Project Workspace as an automatic project if it isn't already contained there.

26.5. Outline Area

The **Outline** area offers an overview over the content of the currently selected file, for example if the `MLCoordUtils1.def` file is opened, all modules defined there will be listed.

The nested elements of the module definition can be opened by clicking the plus sign. Double-click the entries to go to the code line in which this entry is defined.

Figure 26.11. Outline Area

26.6. Edit Area

In the **Edit** area, the source code is displayed. For each open file, a tab with the file name is displayed. For the supported languages MDL, Python, HTML, CSS, XML, and JavaScript, syntax highlighting is available.

For the text portions of .mhelp, CSS, HTML, and XML files, MATE performs a spell checking based on hunspell. If a word is unknown to the spell checker, it is underlined in red; the context menu of this word offers to replace the unknown word with a selectable known variant or to add the word to the user's dictionary.

The spell checker is based on hunspell and uses three dictionaries: the standard American English dictionary, a MeVisLab-specific dictionary, and the user dictionary where new words are stored to.

At the bottom of the area, the line in which the cursor currently stands is displayed. In addition, the cursor position in the source code in terms of level/depth is displayed, if applicable.

Figure 26.12. MATE Edit Area

```

16 Field useOrientation { deprecatedName = UseOrientation } ^
17 Field orientation {
18     deprecatedName = Orientation
19     items {
20         item XYZ { title = "XYZ" }
21         item YZX { title = "YZX" }
22         item ZXY { title = "ZXY" }
23         item XZY { title = "XZY" }
24         item ZYX { title = "ZYX" }
25         item YXZ { title = "YXZ" }
26     }
27 }
28 Field applyOrientation { deprecatedName = ApplyOrientation }
29 Field useGlobalInputImage {
30     deprecatedName = UseCache
31     deprecatedName| = useCache
32 }
33 Field applyPage { removed = yes }
34 }
35
36
37 Window {
38     Category Main {
39 <

```

Description { Field useGlobalInputImage { Line 31 }

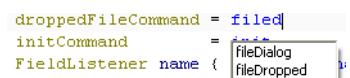
Figure 26.13. MATE Edit Area — Code Completion for Keywords



Code completion is available for the supported languages MDL, and Python. Select the option with the cursor keys and press **ENTER**.

In Python scripting files, the auto-completion offers information about functions of included libraries and about local variable names.

Figure 26.14. MATE Edit Area — Code Completion for Commands Defined in MDL



For the supported languages, context-specific help is available. Either press **F1** (Windows/Linux) or right-click to open the context menus and select a help to be displayed in the default browser (may be set in [Section 4.3.4, “Preferences — Supportive Programs”](#)).

Figure 26.15. MATE Edit Area — Context Menu


```

Commands {
    source           = "${LOCAL}/LocalImage.js"
    droppedFileCommand = fileDropped
    initCommand     = init
    FieldListener[er name / command = nameChanged]
}
}

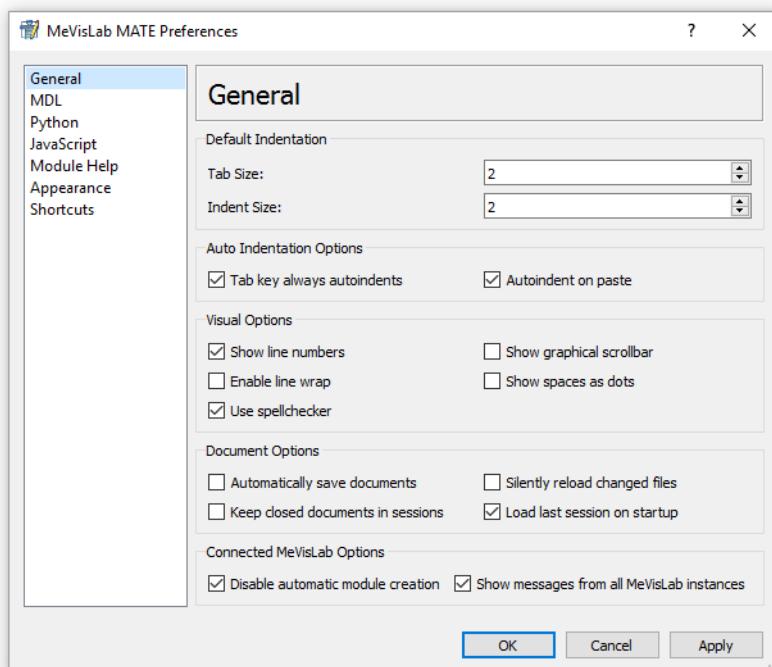
```

**Tip**

If you double-click a word in the editor, the word is being selected and all other occurrences of that word are highlighted in the text (except they are keywords). To remove the highlighting, just hit ESC.

26.7. Preferences

Under **Edit → Preferences**, the preferences for the user interface concerning indentation, fonts, and syntax highlighting can be edited.

Figure 26.16. MATE Preferences*Indentation*

Sets **Tab Size** (for TAB) and **Indentation** (for automatic indents).

Font

Sets **Font** (fixed-width fonts only) and **Font Size**.

Syntax highlighting

Sets **Formats**, their color and the font type **Bold** (optional). **Reset To Defaults** will reset all formats to their default settings.

Extras

Select **Show line numbers** to show line numbers in the editor.

Select **Show graphical scrollbar** to show a scroll area with an approximated text display in the edit area instead of the usual vertical scrollbar.

Select **Show spaces as dots** to display spaces in the edit files with dots.

Select **Tab key always autoindents** to translate TAB into the indent set above.

Select **Autoindent on paste** to automatically indent a selection on pasting it.

Select **Use spellchecker** to enable spell checking in .mhelp, CSS, HTML, and XML files.

Select **Show messages from all MeVisLab instances** to show the messages written to the debug output console of all running MeVisLab instances to the debug output console of MATE. If not selected, only messages from non-application, non-background instances will be shown.

Select **Disable automatic module creation** to disable the creation of a module when starting to edit module files (.def, .script, .py).

Select **Load last session on startup** to load the last active session and its files on starting MATE. The session files are being loaded lazily.

Select **Enable line wrap** to enable wrapping of lines too long to display. This option can be set individually for other document types.

Select **Keep closed documents in sessions** to keep documents on closing in the currently active session. Otherwise, a document will be removed from the session on closing the document.

Under the links **MDL**, **Python**, **JavaScript**, and **Module Help**, the tab, indent sizes, and line wrap behavior for each language can be set. This will override the general settings.

Select **Use internal HTML preview** in the **Module Help** preferences to tell MATE to open generated module help documents in an interal HTML view instead of the system web browser.

26.8. Python Debugger

MATE comes with an integrated Python debugger. Its symbols and basic functions resemble those of Visual Studio. Depending on the operating system, the available key commands are taken from Visual Studio (Windows/Linux) and XCode (Mac OS X).

Debugging can be used for .py Python files and for .script files. In case of the latter, only lines with real Python code (i.e. starting with "py: or "*py:) will work as breakpoints.

There are two main ways to open code in the MATE-integrated debugger:

- Click the bug button  in MeVisLab to start MATE in a separate process with the debugging enabled, then open a file. The button is part of the **Script Debugging** toolbar, see [Figure 4.30, “View — Toolbars Submenu”](#) for enabling and disabling the toolbar.
- Open a Python or Script file in MATE, then select **Debug+Enable Debugging**. (If the **Debug** menu is disabled, MATE is not set up to start as separate process. In this case, change the Preferences setting, see [Section 4.3.4, “Preferences — Supportive Programs”](#).)

When debugging is enabled, the views **Debug Output**, **Breakpoints**, **Stack Frames**, **Variables**, and the **Debugging** toolbar are switched on in standard configuration.



Note

Configuration changes made by the user will be saved separately for MATE without debugging and for MATE with debugging enabled. This way, two basic configurations are available.

Figure 26.17. MATE with Python Debugger

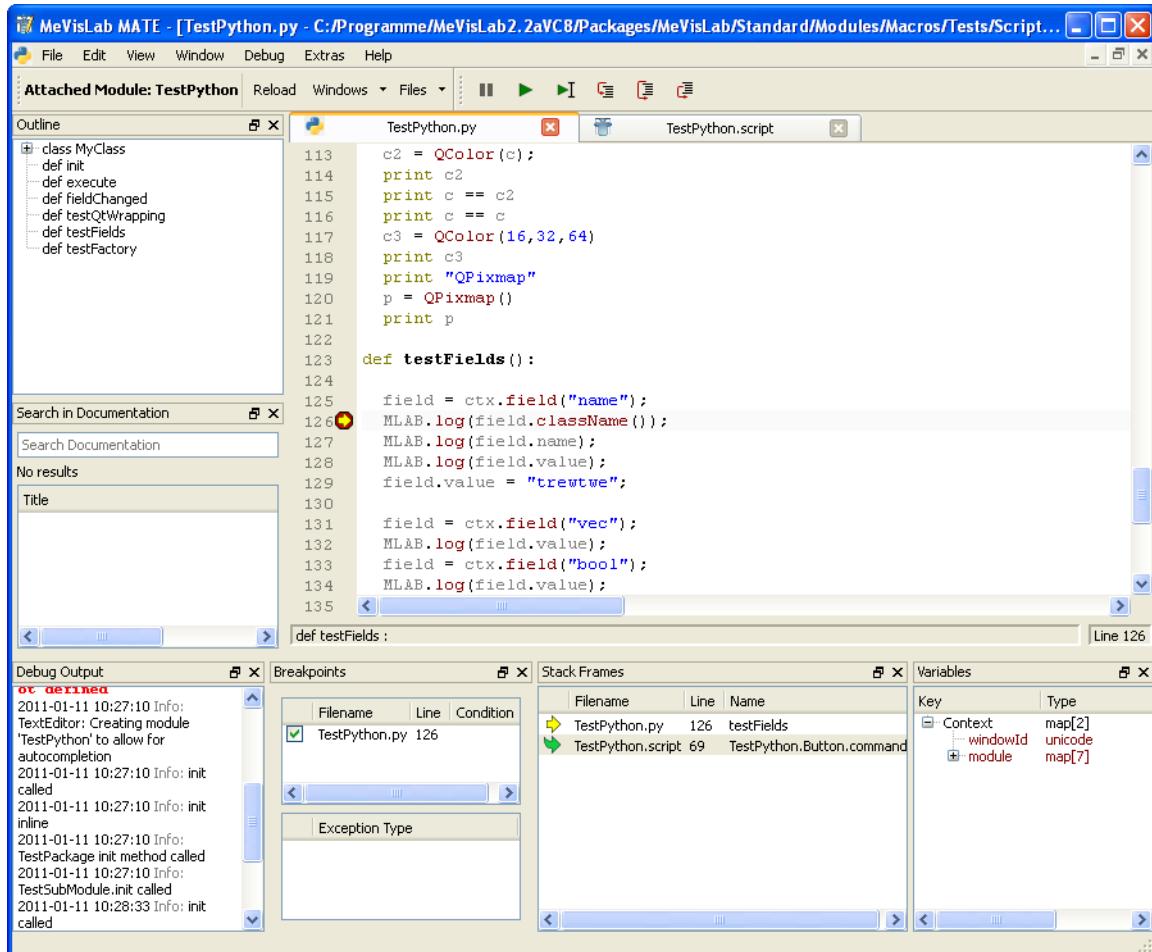
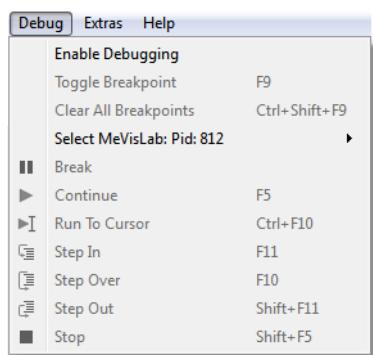


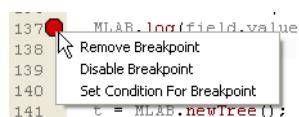
Figure 26.18. MATE Debug Menu



Once the debugging is enabled in principle, the actual process needs to get started. For a first test, the module `TestPython` can be used.

1. In a line with executable code, click between line number and code to set a breakpoint. A red dot will be displayed at that position.

Breakpoints offer context menus.



Breakpoints can be set to conditional; this condition can also be removed. Conditions have to be entered in Python syntax, for example: `a==12`.



2. In a next step, do something on the module's panel in MeVisLab, e.g., click a button.



Note

It is important to remember that the Python scripts in MeVisLab are part of a module or network, and not stand-alone. Therefore, you cannot simply open a script in the MATE debugger and debug it at runtime. What looks like a Start button is actually a Continue button.



Note

During the debugging, MeVisLab is unresponsive! Finish the debugging by letting it run its course before trying to interact with MeVisLab.



Note

The debugger needs an up-to-date `.pyc` file for associating the run-time state with the lines in the `.py` file. The debugger will not work (correctly) if you have multiple `.pyc` copies of one Python file on your system!

The script stops at the breakpoint, showing a yellow arrow. The **Debug Output** is the same as in MeVisLab. In the **Breakpoints** view a list of all breakpoints can be found, and a context menu is available for editing them. The **Stack Frames** view shows where the script currently is. In the **Variables** view, existing variables can be examined.

3. Click the Continue button ► to run to the next breakpoint, or other buttons for other actions.

Table 26.1. Buttons for Debugging

Button	Description	Explanation
■	Break	Pauses the running function.
■	Stop	Stops and finishes the running function.
►	Continue	Runs to the next breakpoint or the end of the routine.
►I	Run to cursor	Runs to line where the cursor is placed.
□	Step In	Step through the code one statement at a time. If the statement executes another function, the debugger will step into that function.
□	Step Over	Step Over is the same as Step In, except that when it reaches a call for another function, it will not step into it. The function will run, and then go to the next statement in the current function.
□	Step Out	If Step In has been used, Step Out can be used to run the currently called function and return to the function it was called from.

Table 26.2. Icons for Debugging

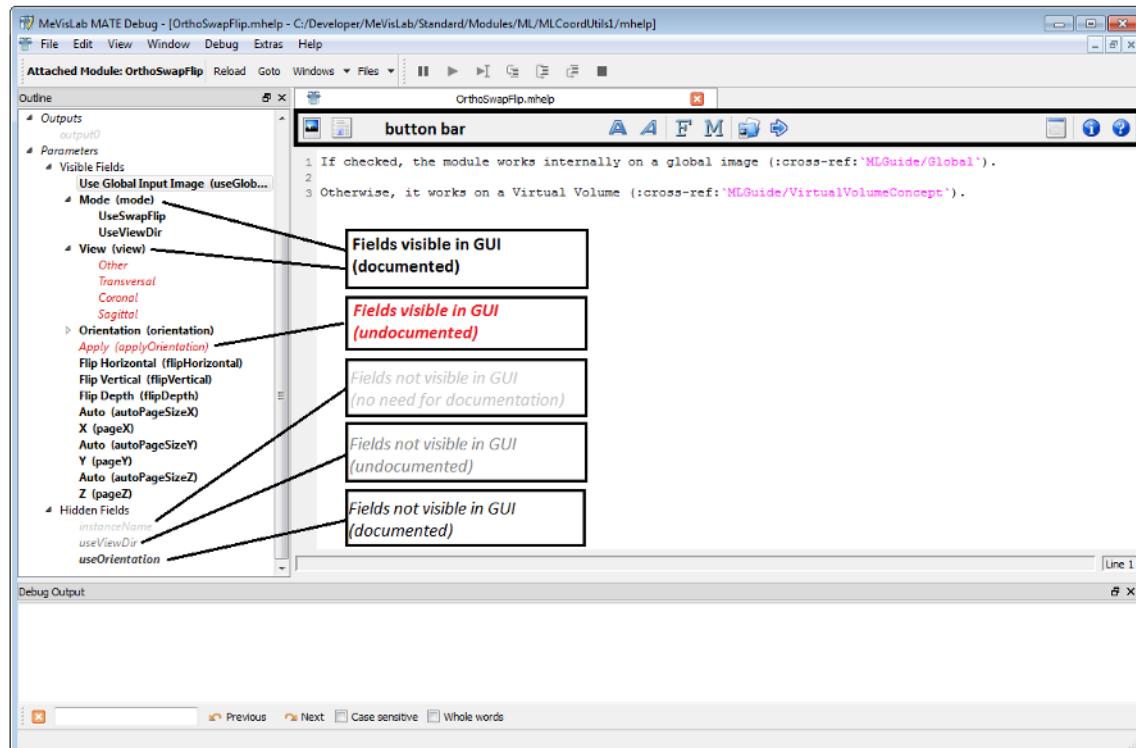
Icon	Description	Explanation
●	Breakpoint	Static breakpoint set by user.
if	Conditional breakpoint	Breakpoint depending on a statement to be true. The condition (in Python syntax) can be set via the context menu of a breakpoint.
▶	Top stack location	Last executed statement.
◀	Current stack location	Displayed via double-clicking the stacks in the Stack Frames view.

26.9. Module Help Editor

A module's help is created on base of *.mhelp files. When these are edited in MATE, a number of additional, help-specific features are available: a customized GUI with special outline appearance and toolbar, syntax highlighting, and auto-completion. The goal is to keep the actual documentation generation process far from the user; for details on the workflow behind the scenes, see [Section 26.9.3, "How it Works"](#).

The free text areas use the reStructuredText (reST) markup, see [Section 26.9.2, "Formatting"](#). Aside of the usual formatting, two expressions are important. "Roles" refer to extensions for inline markup in reST, in our case for field and module references. "Directives" refer to extensions that add support for new constructs, in our case links to images, screenshots, and crossreferences.

To open MATE as Help Editor, either open an *.mhelp file or right-click a module and select **Edit Help**.

Figure 26.19. MATE for Module Help

When editing the help file of a module, all important information of the module down to the field specifications are extracted automatically. The basic module information is therefore always available

in the module help. Additional documentation should be added by the user, especially into the areas Purpose, Usage, Details, Interaction, and Tips.

The directives for screenshots (`.. screenshot:: [window]`) are already prepared, one entry for every window of the module. To remove the screenshot from the module help, just remove the text from the screenshot entries.

The help is aware of changes, this results in the following for field and enumeration items:

- Field is added: field is marked as new and needing documentation.
- Field is removed: documentation text is kept but marked as being removed. Previously written help text is stored in the `.mhelp` file but will not be visible in the resulting HTML help.
- Field is renamed: if the field name is `deprecated`, the text is copied. (Otherwise, the system sees the rename as an adding and removal of fields, so documentation text needs to be moved to the new field manually.)



Tip

With these change-tracking mechanisms, the TestCenter can be used to test for documented fields.

For fields and enumerations, the first paragraph of the help is used as tool tip in the panels. This should be kept in mind when entering information.

The outline formatting is as follows to show the state of the documented items:

- red italic: User-added documentation still missing.
- black bold: User-added documentation exists.
- light-gray italic: Element with “Needs documentation” unchecked.
- dark-gray italic: Fields not visible in the GUI.
- dark-gray italic bold: Fields not visible in the GUI, for which user-added documentation exists.

Formatting of the text can be done via the buttons in the middle of the toolbar (see [Table 26.3, “Help Toolbar Buttons”](#)), using the context menu (see [Figure 26.21, “Text Context Menu”](#)), and manually entering the markup in reST syntax (see [Section 26.9.2, “Formatting”](#)).

Generating the actual HTML help can be done with () or without () prior screenshot generation. The result is automatically displayed in the default browser.



Tip

On the first generation of the HTML help, MATE checks for screenshots being made no matter which of the two buttons above you press.

Table 26.3. Help Toolbar Buttons

Button	Description	Explanation
	Check screenshots and generate help	Opens a window first in which the screenshots to be created can be selected. For the selected screenshots, the images are created and added to the ... screenshot directives. If the module window is open with visible data, the windows will be captured as-is; this allows for adding illustrative information via the screenshots.
		Note Screenshots are not taken properly when using the Aero design of Windows 7. Please change to another design.
	Generate Help	Compiles the help project to create HTML output, without new screenshots.
	Bold	Formats selected text in bold (** * in reST syntax).
	Italic	Formats selected text in italic (* * in reST syntax).
	Field link	Enters field link in editor.
	Module link	Enters module link in editor.
	Image link	Opens a dialog to browse for images relative to the mhelp folder of the module's help.
	Insert cross reference	Opens a dialog in which references to other documents and sections in the MeVisLab help can be entered.
	Show reST	Opens the help source in reST format in a new tab. The reST format is the base for the HTML creation, so in case of help generation problems, check this to find the problematic lines.
	Show information	Toggles an information area on top of the editor area. It shows information about the selected element and module.
	Help	Toggles a help area on top of the editor area, showing general information on how to enter contents in the help format.

26.9.1. Context Menus

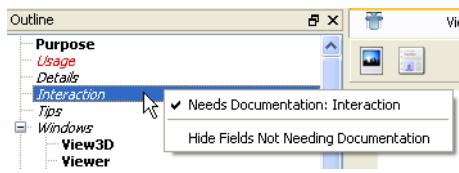
Two options for the outline display are available from the context menu:

- **Needs Documentation [element]:** If checked, the entry is in red italic. If not checked, the entry is in light-gray italic.
- **Hide Fields Not Needing Documentation:** Hides all fields for which “Needs Documentation” is unchecked.



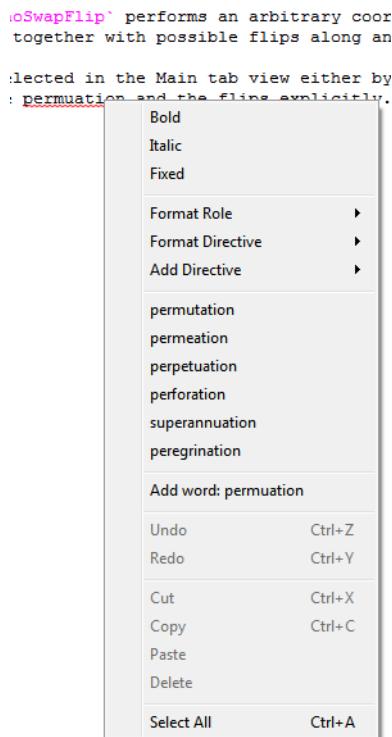
Tip

The context menu entry can be chosen by using the right mouse button. If you need to toggle the need for documentation for a lot of fields, just use the right mouse button to bring up the context menu and use the same right mouse button for selecting the option.

Figure 26.20. Outline Context Menu

The context menu of the text area offers shortcuts to the three basic formatting styles bold, italic, and fixed (see below). If more than one word is selected, the formatting will be applied to all of the selection. If the cursor is placed on a word, the formatting is only applied to this word. It also offers shortcuts to format words as roles (field, module, overview) and directives (screenshot, image, image in package, cross reference), and to add new directives (images and crosslinks).

If a word is unknown to the spellchecker, it will be underlined in red. The context menu (of that word) offers known variants as a correction or to add the unknown word to the user's dictionary.

Figure 26.21. Text Context Menu

26.9.2. Formatting

For the formatting, reStructured Text (reST) syntax is used. It has some similarity with MarkUp or Wiki syntax.

The editor supports the editing process by two means: syntax highlighting of reST elements and autocompletion.

Selected formatting options:

Table 26.4. Inline markup

Markup	Output												
text	<i>italic text</i>												
text	bold text												
~~text~~	fixed space text												
*	• bullet list												
#.	1. numbered list												
[number]. (e.g., 2.)	2. explicitly numbered list												
<pre> 1 * Level One 2 3 * first 4 * second 5 * third 6 7 * Level Two 8 9 #. first 10 #. second 11 #. third 12 13 * Level Three </pre>	<ul style="list-style-type: none"> • Level One <ul style="list-style-type: none"> ◦ first ◦ second ◦ third • Level Two <ul style="list-style-type: none"> 1. first 2. second 3. third • Level Three 												
<pre> 1 +-----+-----+-----+ 2 Column 1 Column 2 Column 3 3 +-----+-----+-----+ 4 a d g 5 b e h 6 c f i 7 +-----+-----+-----+ </pre>	<table border="1"> <thead> <tr> <th>Column 1</th><th>Column 2</th><th>Column 3</th></tr> </thead> <tbody> <tr> <td>a</td><td>d</td><td>g</td></tr> <tr> <td>b</td><td>e</td><td>h</td></tr> <tr> <td>c</td><td>f</td><td>i</td></tr> </tbody> </table>	Column 1	Column 2	Column 3	a	d	g	b	e	h	c	f	i
Column 1	Column 2	Column 3											
a	d	g											
b	e	h											
c	f	i											

Table 26.5. Directives

Directive	Effect
.. image:: relative/path/to/mhelpfile	inserts the referenced image
.. image-in-package:: packageidentifier.relative/path/to/image	inserts the referenced image from another package
.. screenshot:: windowname.PanelName.name or windowname.TabName.name)	<p>inserts the screenshot (if already created) or a link to the screenshot</p> <p>Example with autocompletion:</p> <pre> 1 .. screenshot:: Settings. </pre> 

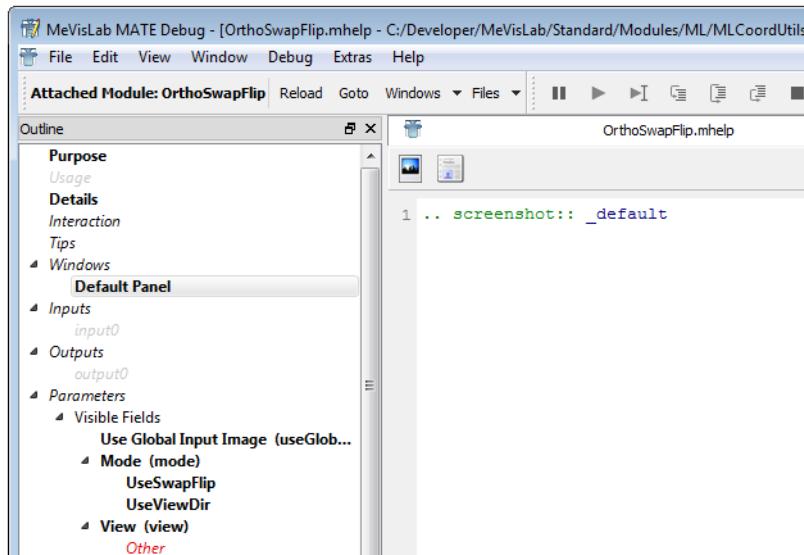
Table 26.6. Roles

Role	Effect
<code>:field: `(modulename.)fieldname`</code>	links to a field of a module
<code>:module: `modulename`</code>	links to a module Example with autocompletion: 
<code>:overview: `overviewname`</code>	links to an overview
<code>:cross-ref: `document/targetptr`</code>	links to targetptr in another document. Use "Insert Cross Reference" to insert this role (see Table 26.3, "Help Toolbar Buttons")
<code>:file-in-package: `packageIdentifier.relative/path/to/file`</code>	links to a file in a package. This is similar to the image-in-package directive
<code>:relative-link: `relative/path/to/file`</code>	links to a file relative to the mhelp file
<code>:sub: `superscript`\ text</code>	adds a superscript to the text
<code>:sub: `subscript`\ text</code>	adds a subscript to the text

For more information about the reST syntax, see [Sphinx reStructuredText Primer](#).

26.9.3. How it Works

.mhelp files are written in MDL and have an MDL tree structure. They are created from the module interface definition and the GUI definition. Upon every edit initiated from a module's context menu in MeVisLab, this MDL tree is created again with all module data in the module help being updated automatically. The texts that the user adds in the actual "Edit Help" step are merged with the updated data. If the module has changed in structure, e.g., elements, fields, or enumerators are moved or renamed, this will be handled as follows: deleted elements will be removed, added elements will be added, renamed elements will be renamed and a help text will be added to the element.

Figure 26.22. Automatically Documented Elements

Upon creation, the following information from module sources is extracted and added to the module help:

- Windows: For each window of the module, an outline entry with the respective screenshot directive is added.
- Inputs: Each input is added as outline entry. Available comments are added as documentation for the respective inputs.
- Fields: All fields used in the GUI windows are added as outline entry (red italic). After that, all fields not visible in the GUI (“hidden”) are listed (dark-gray italic). Available tool tips will be added as documentation for the respective fields.

The conversion of the `.mhelp` file to HTML happens in two major steps:

- The MDL file is converted to reStructured Text (see [reST](#)) by a core macro module .
- In a second step, the reST format is converted to HTML via Sphinx (see [Sphinx Python documentation generator](#)). To allow every MeVisLab user to create module help, Sphinx is delivered with and integrated in MeVisLab.

The help system is implemented in MeVisLab core libraries, this makes the automatic generation of tool tips from help texts (and from tool tips to help text upon first creation of a help file) possible.

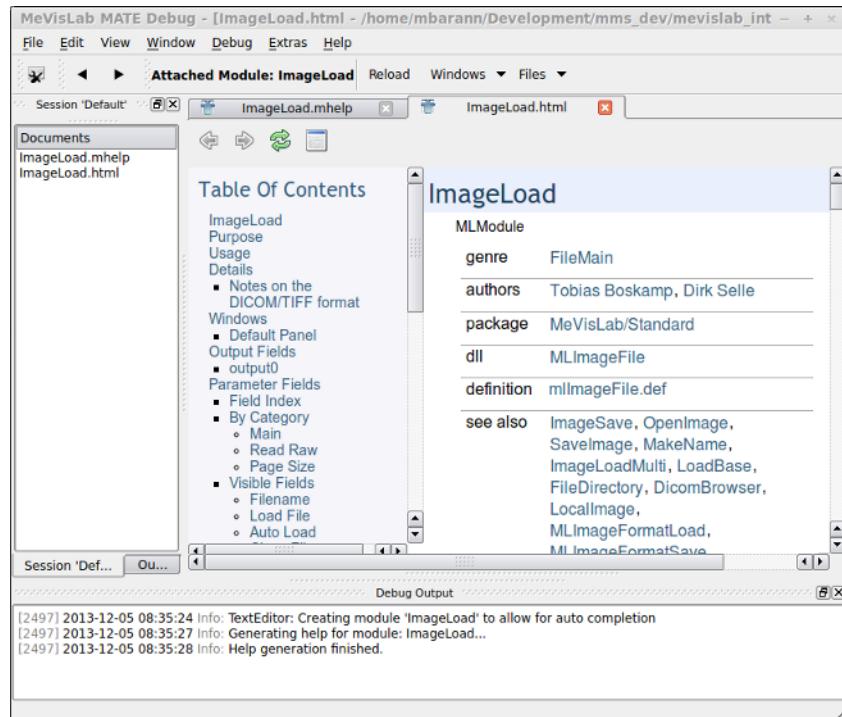
All module help source files are generated in a `mhelp` directory besides the module definition files to which they belong.

Resulting HTML files are put below the `Documentation` folder of the package.

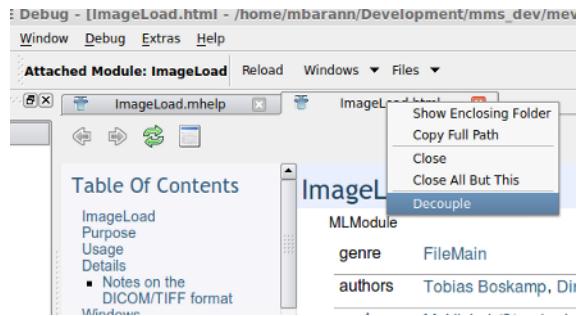
26.9.4. Internal HTML Preview

You can tell MATE to open generated module help documents in an internal HTML view instead of the system web browser ([Section 26.7, “Preferences”](#)). The HTML view is an additional tab in the workspace:

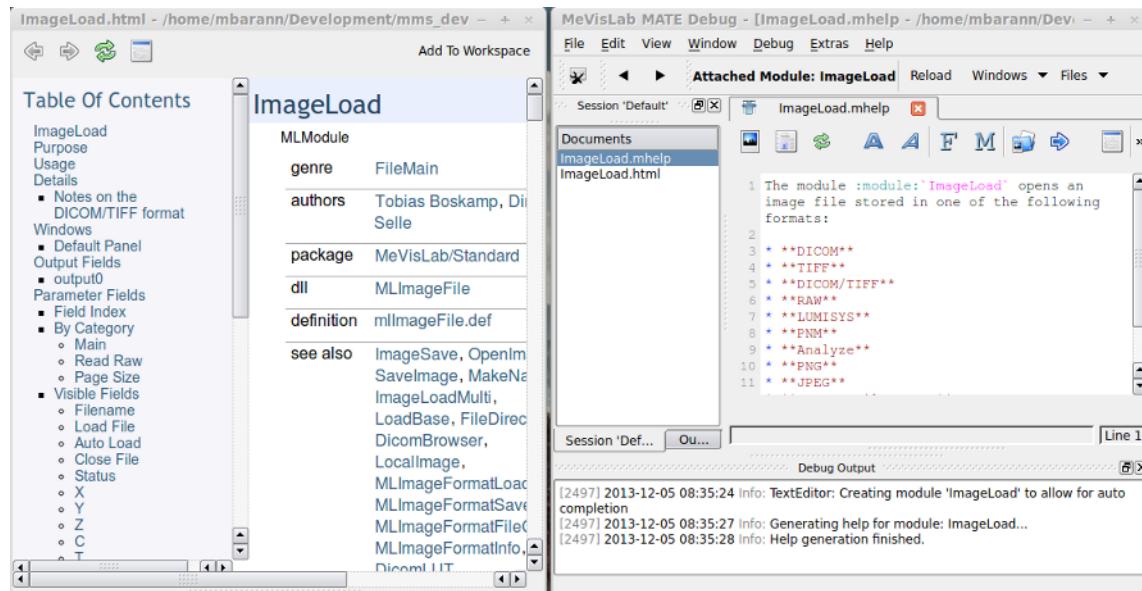
Figure 26.23. HTML View



You can view the module help editor and the HTML view at the same time, by decoupling the HTML view from the workspace as a separate window:

Figure 26.24. HTML View Decoupling

By pressing the **Add To Workspace** button, the HTML view is returned it to the workspace:

Figure 26.25. Decoupled HTML View

26.10. Session Management

MATE offers a session management. MATE can maintain multiple, named sessions and a session consists of a list of associated files.

One default session called **Default** always exists and cannot be removed. New sessions can be generated and named, and existing sessions can be cloned. This can be done either via the session view's context menu or via the **File** menu.

If a session is active, all opened and created documents will be stored in that session. In MATE's **Preferences** dialog, you can toggle to keep closed documents in the session.

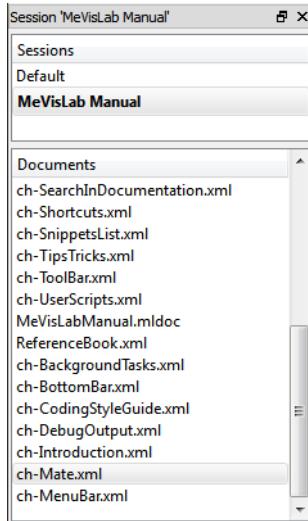
The last active session can be set in the **Preferences** to be loaded on starting MATE. Note that MATE loads the documents of a session lazily; only a tab with the according title is prepared and the actual document content is only loaded on opening that tab.

In the session and in the according documents window, the so-called extended selection is enabled. That means that multiple entries can be selected by either holding the **CTRL** key pressed for selecting successively or by holding the **SHIFT** key pressed for selecting a range of entries.

Pressing **DEL** or **BACKSPACE** removes the selected entries. Only the default session cannot be removed.

A single click selects a session or a document. A double click opens a session or a document.

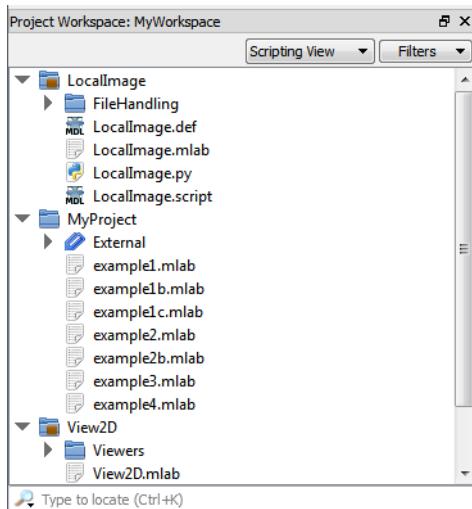
Both, the session and the documents window offer more options via the context menu.



26.11. Project Workspaces

MATE offers to organize files and directories in projects, which are referenced from a workspace. Only one workspace can be open at a time.

If no persistent workspace is open, MATE manages a default workspace, whose content and state is lost when MATE is closed. A persistent workspace can be created from the **File** menu by selecting **Project Workspace → New....**. Existing workspaces can be opened with **Project Workspace → Open...** and by selecting an entry from the **Recent Project Workspaces** menu. The default workspace can be restored by selecting **Close Project Workspace**. A workspace can be saved under a different filename from the **File** menu by selecting **Project Workspace → Save As....**. The default workspace can also be saved to change it into a persistent one.



26.11.1. Project Types

There are two different types of projects: **regular projects** and **module projects**.

26.11.1.1. Regular Projects

Projects are stored as files in the filesystem and provide a view of all the files and directories from their parent directory. Files and directories from other locations can be added to the project by right

clicking on the project in the workspace view and selecting **Add -> Existing File** or **Add -> Existing Directory...** from the context menu. These files and directories are listed under the paper clip symbol that is named **External**.

To create a project, go to the **File** menu and select **Project → New...**, the project will be added to the current workspace. Existing projects can be opened with **Project → Open...** and by selecting an entry from the **Recent Projects** menu.

Projects can be removed from the workspace by right clicking on them in the workspace view and selecting the **Remove Project** entry from the context menu.

26.11.1.2. Module Projects

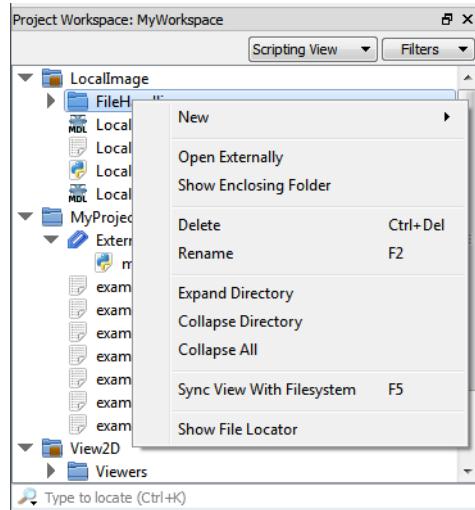
Module projects are automatically managed and contain all related files of the module and additionally the parent directory of these files. It is not possible to add files or directories to these projects, except by editing the related files and directories of the module.

If the default workspace is open, then module projects are automatically added to the workspace when a related file of a module is opened and MeVisLab is available to create the attached module. If a persistent workspace is open, then a module project is not automatically added. Instead, it can be done manually by clicking on **Add to Project Workspace** in the attached module menu.

Module projects can be removed from the workspace by right clicking on them in the workspace view and selecting the **Remove Module Project** entry from the context menu.

26.11.2. Context Menu

The project workspace context menu contains different actions.



The possible actions vary depending on the selected item:

- **Add → File...** and **Add → Directory...** (available on regular project's root item):
Adds an existing file or directory to the **External** section of the project.
- **Remove Project** or **Remove Module Project** (available on any project root item):
Removes the project from the workspace.
- **New → File...** and **New → Directory...** (available on directory entries):
Creates a new file or directory in the filesystem. New files are immediately opened in MATE.

- **Remove File From Project** and **Remove Directory From Project** (available on immediate children of the **External** section):

Removes the file or directory from the project.

- **Delete** (available on any file or directory object):

Deletes the file or directory in the filesystem.



Note

Deleting a file or directory also removes the project reference to it, if it was an external resource.

Deleting a file or directory does not remove the related file or related directory entry of the module. This must be done manually.

- **Rename** (available on any file or directory object):

Renames the file or directory in the filesystem and in the project.



Note

Renaming a file or directory does not adapt the related file or related directory entry of the module. This must be done manually.

- **Expand [...]** and **Collapse [...]** (available on any item with children):

Expands or collapses the item in the workspace view.

- **Collapse All** (available everywhere):

Collapses all items in the workspace view.

- **Sync View With Filesystem** (available everywhere):

Scans the filesystem for changes that are not yet reflected in the workspace view.

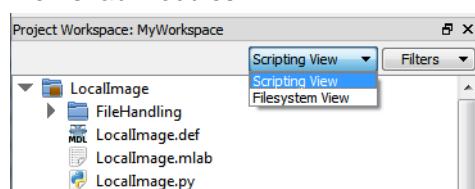
Usually you don't need this since this is done in regular intervals.

- **Show File Locator** (available everywhere):

Shows the file locator, which is the search input line below the workspace view.

26.11.3. Views

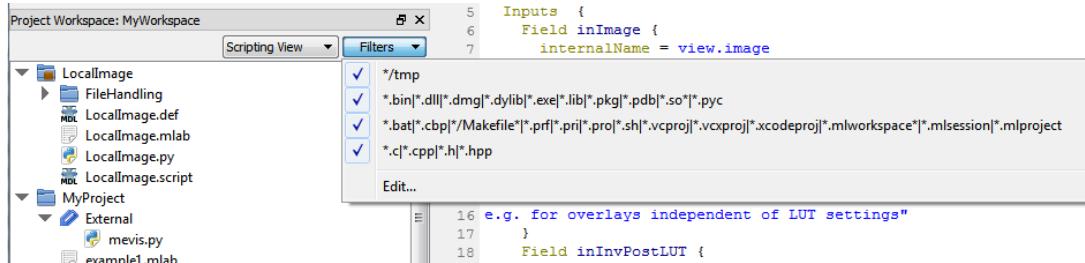
MATE offers two different views for the project workspace: the **Scripting View** and **Filesystem View**. The views allow to manage exclude filter rules for the displayed files and directories. The Scripting View includes several rules by default, which aim at showing only relevant files and directories for scripting MeVisLab modules.



- **Scripting View** only shows the relevant files of your projects; many temporary or generated files are filtered away.

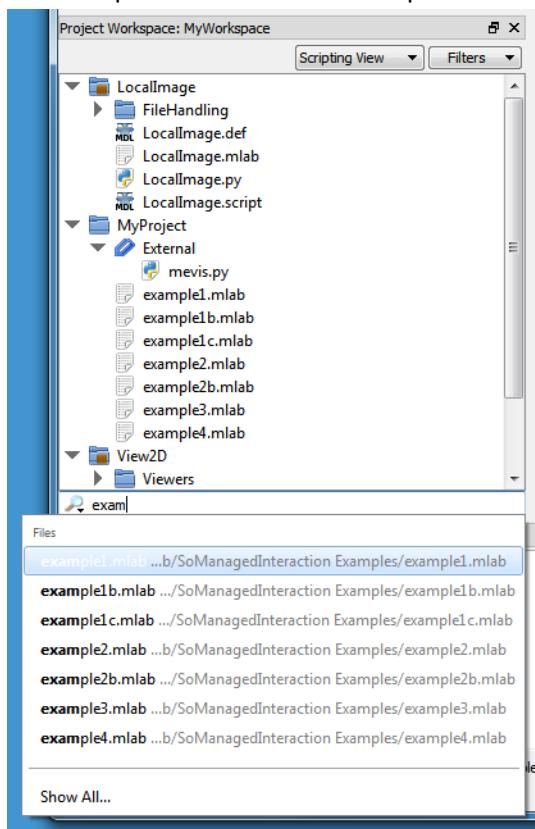
- **Filesystem View** shows all files in your project by default.

The set of filtered files can be configured for both views:



26.11.4. File Locator

A lot of files can be contained in the workspace view. To quickly find certain files you can use the file locator input field below the workspace view:



This input field can be closed by pressing **Escape** when the input focus is in the input field. It can be opened again with **Show File Locator** from the context menu in the workspace view, or by pressing **CTRL+K**.

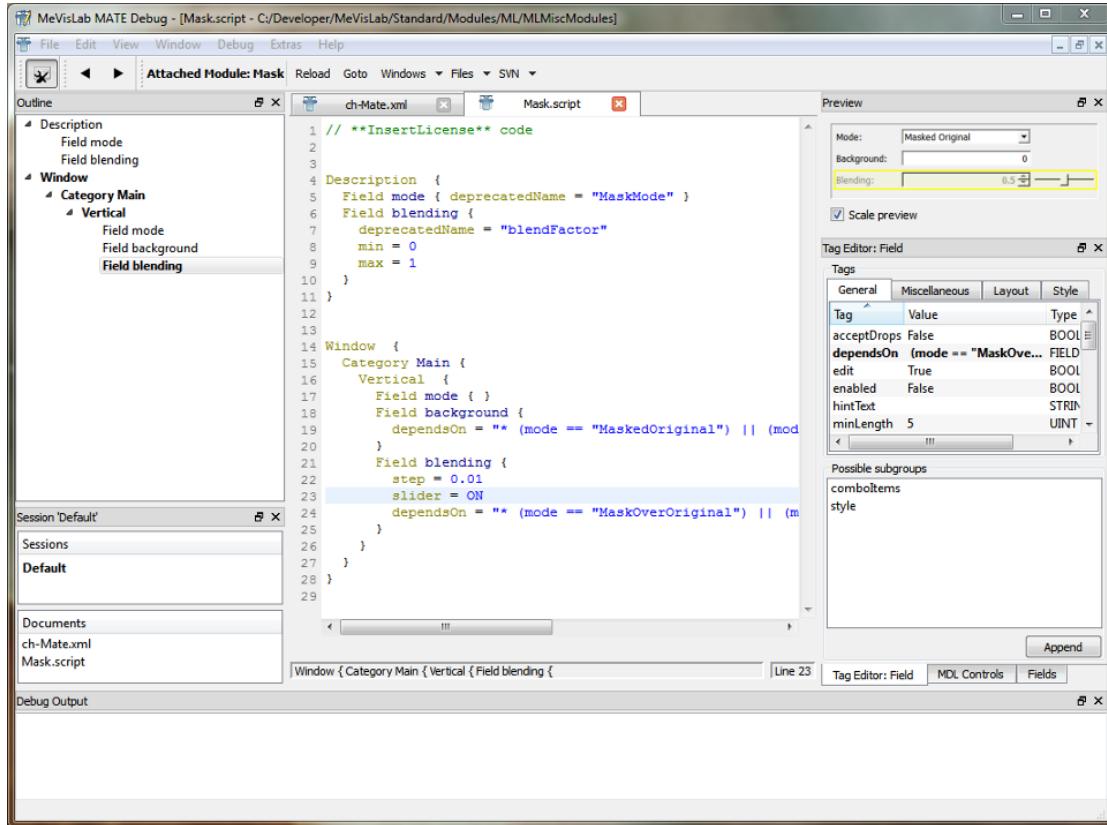
26.12. GUI Editor

MATE offers basic GUI editor functionality. The GUI editor is not a straight up WYSIWYG editor, but offers lists of defined fields, available GUI controls, a preview window, and a tag editor. All those views and data representations are synchronized and most of them offer basic drag-and-drop functionality.

The GUI editor in MATE is not meant to support the creation of large and complex user interfaces, such as application interfaces, but should provide a fast and easy way to create a simple module panel in a short amount of time. Most importantly, the GUI editor provides a panel preview that is updated in real time while writing the GUI description, while configuring it via drag-and-drop, or while adjusting

parameters of single controls. The preview also shows the scope of edited GUI controls and layout groups for an easy orientation which also helps learning how to build a panel for MeVisLab modules.

In the screenshot below, the GUI description of the Mask module is being edited in MATE's GUI editor. In the editing window, the text cursor is set to the description of the field `blending`; the same field is automatically highlighted in the outline to the left, and also highlighted with a yellow rectangle in the preview to the right. Those three representations are all synchronized. Selecting another field in the outline leads the preview to show the new selection and sets the text cursor in the editing area to the according field. Also, clicking onto a field control in the preview leads to a highlighting of the field in the outline and to the text cursor being placed to the field's description in the editing area.



Drag-and-drop is implemented for the most views of the GUI editor. In the outline, controls can be re-arranged by drag-and-drop, fields from the **Fields** can be dropped onto the editing area to generate a control description (another window pops up to offer a selection how that field should be displayed), and general controls can be dragged out of the **MDL Controls** view onto the editing area.

If the text cursor is in a control's description in the editing area, the **Tag Editor** offers a list of all available tags with the currently set tags printed in a bold face. All shown tags' values can be edited in the **Tag Editor**. If an edited tag changes the visualization of the field, the preview is updated accordingly.

26.13. Scripting

MATE can be augmented by Python scripting, very similar to the [Section 4.8, “User Scripts”](#) in MeVisLab. For the scripting API, please refer to the MATE and the MATEDocument scripting references.

In addition to the user scripts, MATE can also execute a startup script. To edit that script, choose **Edit MATE Startup Script File** from the **Extras** menu.

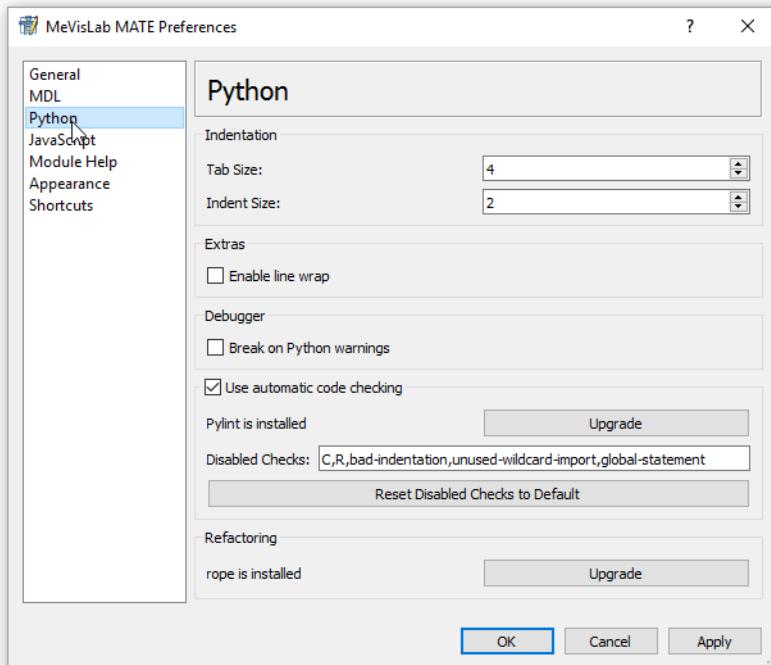
Other than that, own user scripts for MATE are just like the user scripts in MeVisLab. MATE comes with some predefined user scripts that manage SVN operations on files and directories, as well as implementing some string functions on text selections. Have a look at those scripts to learn more.

26.14. Pylint Integration

MATE has [Pylint](#) integration: Python files are automatically checked for errors with Pylint if Pylint is installed for Python 3 in the user's Python package directory. (Not the system's Python package directory!)

26.14.1. Installation

Installation of Pylint can be done with the pip tool, e.g., from a Python 3 installation with the command line `pip install --user pylint` or from the **Preferences** dialog of MATE in the **Python** section:



Note



Automatic installation from the **Preferences** dialog requires an Internet connection.

Python code checking can be disabled from the preferences even if Pylint is installed. You can also configure which checks should be suppressed (see the Pylint [message documentation](#) for this); code convention warnings, recommendations, and some other warnings are not displayed by default.

26.14.2. Usage

Pylint checks are performed automatically on Python code if Pylint is installed and activated. Results are displayed on the left side of the text file and also indicated on the right side of the scroll bar for a complete overview. Result symbols are displayed according to the highest message category for that line, and the tool tip for each symbol contains the error message(s) for that line:

```

26 def nameChanged():
27     exp = ctx.expandFilename(ctx.field("name").stringValue())
28     if MLABFileManager.exists(exp):
29         if MLABFileManager.isDir(exp):
30             MLAB.logWarning("LocalImage.name - File is a directory (" + exp + ")")
31         else:
32             ctx.field("ImgLoad.filename").setStringValue(exp)
33     else:
34         if exp:
35             MLAB.logWarning("LocalImage.name - File does not exist (" + exp + ")")
36         ctx.field("ImgLoad.filename").setStringValue("")
37     return
38
39
40
41 def fileDialog():
42     exp = ctx.expandFilename(ctx.field("name").stringValue());
43     file = MLABFileDialog.getOpenFileName(exp, "", "Open file")
44     if files:
45         ctx.field("name").value = ctx.unexpandFilename(file)
46     Undefined variable 'files'
47

```

def fileDialog : if:
1 red, 2 yellow, 1 blue Line 44

Below the Python text is also an indicator showing how many messages were generated in total by Pylint. Clicking this indicator will jump to the next result in the text. A tooltip will also show how much time was spent in the last check run.

```

41 def fileDialog():
42     exp = ctx.expandFilename(ctx.field("name").stringValue());
43     file = MLABFileDialog.getOpenFileName(exp, "", "Open file")
44     if files:
45         ctx.field("name").value = ctx.unexpandFilename(file)
46     return
47

```

def fileDialog : if:
1 red, 2 yellow, 1 blue Line 44

Errors were detected by Pylint.
Click to jump to next error line.
Time spent in check: 0.94s

A busy indicator will also be shown in this place while a Pylint check is performed.



Note

The Pylint integration is not perfect, and Pylint will not see the code the same way as MeVisLab, so Pylint might complain about unknown identifiers even if the code executes perfectly well in MeVisLab.



Note

Pylint runs constantly while the currently edited Python file changes and the process will run full steam in the background. If your computer only has one available CPU core you don't want to use this feature.

Chapter 27. Tips and Tricks

27.1. Command-Line Options

It is possible to start MeVisLab with command-line options.

Windows: To start MeVisLab from the command line, enter:

```
Mevislab [OPTIONS] [networkfile].mlab [networkfile2].mlab ...
```

Linux: To start MeVisLab from the command line, enter:

```
<mevislab installpath>/bin/Mevislab [OPTIONS] [networkfile].mlab [networkfile2].mlab ...
```

To get a list of all available options, start MeVisLab with the `-help` option. This will open a window with all commands.

-scan	Scans Modules directories for new files.
-noscan	Uses existing Modules cache files and does not scan Modules directories for new files. Tips and tricks
-quick	Uses existing Modules cache files and does not check for any file changes (allows very quick restart).
Table 27.1. Command-Line Options	
-diagnosis	Shows a diagnosis console while starting.
-logfile FILENAME	Sets the logfile. Overwrites any logfile from the prefs file and the registry.
-userscript FILENAME [arg1 arg2....]	Runs the given user script (see Section 4.8, "User Scripts"). Instead of the filename, the name of the Action can also be given.
-runmacro MACRONAME arg1 arg2....	Runs a macro by calling its consoleCommand (requires a license with 'cmdline' feature).
-runapp APPNAME arg1 arg2....	Runs a MeVisLab application, passing arguments to its runApplicationCommand and showing the apps window afterwards (requires a license with 'cmdline' feature).
-runappbatch APPNAME arg1 arg2....	Like -runapp, but does not start the application window and exists after execution of runApplicationCommand in conjunction with the -noide option.
-appname APPNAME	Specifies the application name that is used when accessing registry keys and settings.
-showfullscreen	Shows application fullscreen.
-showmaximized	Shows application maximized.
-help	Shows all available command line options.
-nosplash	Does not show the MeVisLab splash screen.
-nowelcome	Does not show the MeVisLab Welcome screen (debugging option).
-singleinstance	Only allows one MeVisLab instance to be started and passes the files of the command line to another running MeVisLab (debugging option).
-noninteractive	Prints error messages instead of showing error dialogs.
-hide-diagnostic-mevislab-messages	Prevents MeVisLab from printing diagnostic messages like "Loading package MeVisLab/IDE (Installed) from ...".
-show-diagnostic-mevislab-messages	Tells MeVisLab to print diagnostic messages like "Loading package MeVisLab/IDE (Installed) from ...".
-disable-logging-timestamp	Prevents MeVisLab from printing the timestamp when logging output.
-enable-logging-timestamp	Tells MeVisLab to print the timestamp when logging output.
-exec EXECUTABLE arg1 arg2... ---	Runs the given executable with given args on startup of MeVisLab. The --- delimiter tells MeVisLab that the args end here and that the rest of the commandline are normal MeVisLab options. The executables are killed again when MeVisLab is exited.
-v/-version/--version	prints the version

Some of these options are not available when using the MeVisLabStarter executable on Windows, if required, use MeVisLab instead.

27.2. MeVisLabPackageScanner.exe

`MeVisLabPackageScanner.exe` is a tool with which you can search for and analyze packages. It is used by various other tools, for example `ToolRunner`.

`MeVisLabScanner.exe` is located in `MeVisLab/Packages/MeVisLab/IDE/bin/`.

If you enter no arguments or `-help`, the help file will be displayed automatically.

Figure 27.1. MeVisLabPackageScanner Help

```
C:\>c:\programme\mevislab2.0avc8\packages\mevislab\ide\bin\mevislabpackagescanner.exe
MeVisLabPackageScanner [arguments]
Available arguments:
  -help/-h
    print this help page
  -run program arg1 arg2 ...
    run the given program with given args (sets the MeVisLab Package environment variables)
  -runForEachPackage program arg1 arg2 ...
    run the given program for each MeVisLab Package (sets the MeVisLab Package environment variables)
    replaces @PACKAGE@ in the arguments with the root directory of each package
  -requirePath RELATIVE_PATH_IN_PACKAGE
    filters the packages, only uses packages that contain the requiredPath, e.g.
  -requirePath Modules
  -verbose
    prints verbose scanning information
  -info
    prints information about all found packages
  -scanPath
    sets a scan path to search for packages, the default paths are ignored when
  scanPath is set
  -printXMLInfo
    prints the environment variables in XML format
  -cleanupPackages
    cleans the installed packages by removing the Modules cache file and the Module Reference html files
  -printColonSeparatedLibPaths
    prints the library paths of all packages joined by colon
```

27.3. EatDicom.exe (deprecated)

`EatDicom.exe` is a tool to sort DICOM files consistently and construct 2d, 3d, and 4d (time series of 3d images) images on output. Image data is stored in a TIFF file (`.tif`), supplementary information summarized in a DICOM file with same filename (`.dcm`). Missing slices are bridged with empty slices up to a maximum gap size (i. e. number of missing slices).



Note

More convenient alternatives for importing DICOM are the modules `DicomImport` or `DirectDicomImport`, provided by Fraunhofer MEVIS. Development of `EatDicom` has been abandoned in favor of these modules.



Note

Currently, `EatDicom.exe` cannot process tilted data (as typically done for head shots). The error message is “`stderr: The dicom data contains no differences in acquisition time nor in content time, found in [filename].`”

The tool comes with a high number of options, which can also be entered as an option file.

To run `EatDicom.exe` from the command line, the full MeVisLab application has to be installed, because the tool depends on the MeVisLab libraries. In addition, paths have to be set.

On Windows: Due to the package structure, `EatDicom.exe` is located in `MeVisLab/Packages/MeVis/Foundation/bin/`, a folder usually not included in PATH. This may result in problems with missing DLLs when starting the executable. Three solutions are possible:

- Add the PATH to the registry.
- Go via the MeVisLabPackageScanner. For example, enter this in the command line: `C:\Programme\MeVisLab\Packages\MeVisLab\IDE\bin\MeVisLabPackageScanner.exe -run C:\Programme`

```
\MeVisLab\Packages\MeVis\Foundation\bin\eatdicom.exe [arguments] (correct the paths as
necessary)
```

- Copy EatDicom.exe from the package MeVis/Foundation/bin to the package MeVisLab/IDE/bin and start it from the command line.

If no arguments or -help are entered, the help file will be displayed automatically.

Figure 27.2. Excerpt of the EatDicom help

```
C:\>c:\programme\mevislab2.0avc8\packages\mevislab\ide\bin\eatdicom.exe
No input directory or files specified.
eatDicom v3.00 beta - Nov 24 2000
Author H.K.Hahn, Copyright <c> 2000, MeVis Bremen
options:
Usage: c:\programme\mevislab2.0avc8\packages\mevislab\ide\bin\eatdicom.exe [+f <
option-file>] [options] [<in-dir> [<out-dir>]]

Description:
  to consistently sort DICOM files and construct 2d, 3d, and
  4d (time series of 3d images) images on output. Image data
  is stored in a TIFF file (*.tif), supplementary information
  summarized in a DICOM file with same filename (*.dcm).
  Missing slices are bridged with empty slices up to a maximum
  gap size (i. e. number of missing slices).

Option File:
If "+f" is given as first argument, another argument <option-file> must
follow naming a valid text file. This file must contain a white-space
separated list of options and is read before parsing the rest of command line.

General Options:
-v<0!1!2>    set verbose mode; if the digit is omitted, <1> is used
  e.g.: "-v" (<"-v1">, "-v0" <default>, "-v2" <full verbose>
-help          show this help
-l             list only (do not convert or delete files)
-r             recursively read all sub-directories of <in-dir>
-i <file>      add <file> to the list of input files
-id <dir>      use <dir> as <in-dir>
-od <dir>      use <dir> as <out-dir>
-2             convert also datasets from less than 3 input files (default: at
  least 3 files in order to dismiss 2d files and scout images)
-dt t          leaves blocks untouched that contain files newer than
  <t> seconds, e.g.: "-dt 120"
-ignore <gggg,tttt> ignore user defined tag (by hexadecimal DICOM
  group and tag id) for image partitioning and sorting,
  e.g.: "-ignore <0010,0010>" for ignoring PatientName
-rules         show list of dicom tags used for image partitioning and sorting

Output Options:
-anon          anonymize output <patient name and birthdate>
-t              TIFF only to <out-dir> (no *.dcm file)
```

When the executable is run, an import log is displayed.

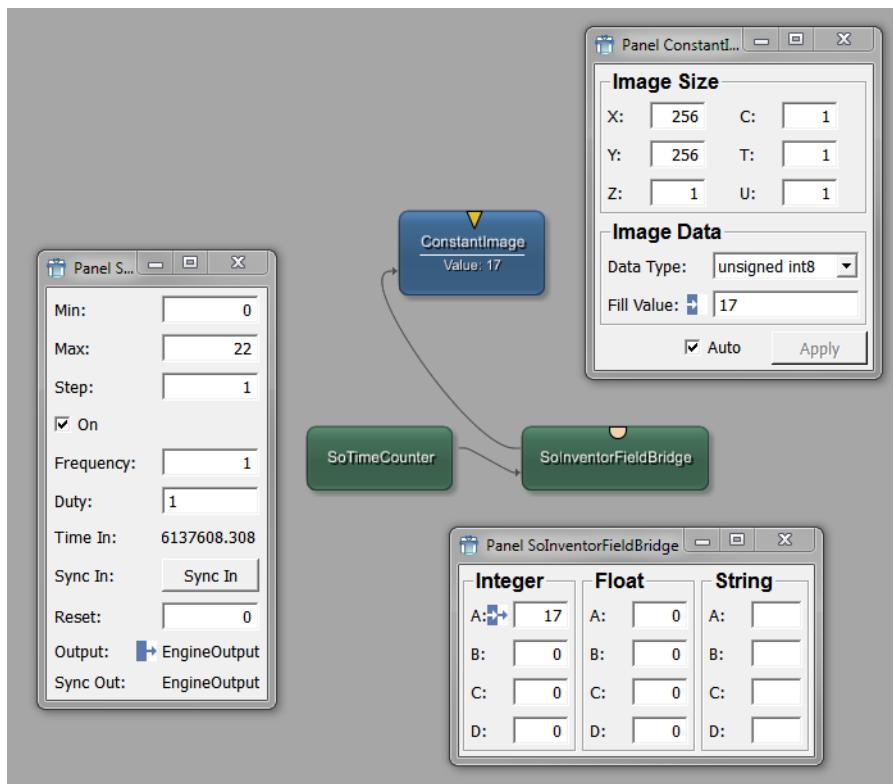
Figure 27.3. Dicom Import Log

```
C:\Programme\MeVisLab2.0avc8\Packages\MeVisLab\IDE\bin\eatdicom c:\tmp\input_dic
on c:\tmp\output_dicom
# save image to <c:\tmp\output_dicom\Doe_John_3845_0001.tif>
# save DICOM info to <c:\tmp\output_dicom\Doe_John_3845_0001.dcm>
# dicom base image from <c:\tmp\input_dicom\1.2.840.113619.2.5.1762872244.1440.9
75243113.11.dcm>
The dicom data contains no differences in acquisition time nor in content time,
found in .\eatDicomMain.cpp:1277
# * patients name: Doe^John
# * patients ID: 3845
# * patients birth date: 19510101
# * patients sex: F
# * institution name: Mass General Hospital
# * acquisition date: 20001128
# * acquisition time: 125057
# * study description: 5686/GI
# * series description: SSFSE AXIAL LOC
# * pixel x/y spacing: 1.015625\1.015625
# * pixel z spacing: 9
# * number of frames: 13
# * number of image positions: 13

eatDicom v3.00 beta - Nov 24 2000
Author H.K.Hahn, Copyright <c> 2000, MeVis Bremen
Successfully terminated.
```

27.4. Connecting Inventor Engines to ML Modules

There are two kinds of Inventor modules: nodes (basically objects with a state) and engines (basically functions/actions, for example a calculator or a time counter). Inventor modules of the engine type cannot be directly connected to ML connectors. In this case, a field bridge has to be used.

Figure 27.4. Field Bridge Example

27.5. Using SyncFloat to Reduce System Load

When creating parameter connections between ML modules and Inventor modules, system load may increase considerably. This is due to the delay queue handling of Open Inventor modules that may cause notification loops. To avoid this, the `SyncFloat` or `SyncVector` modules can be used.

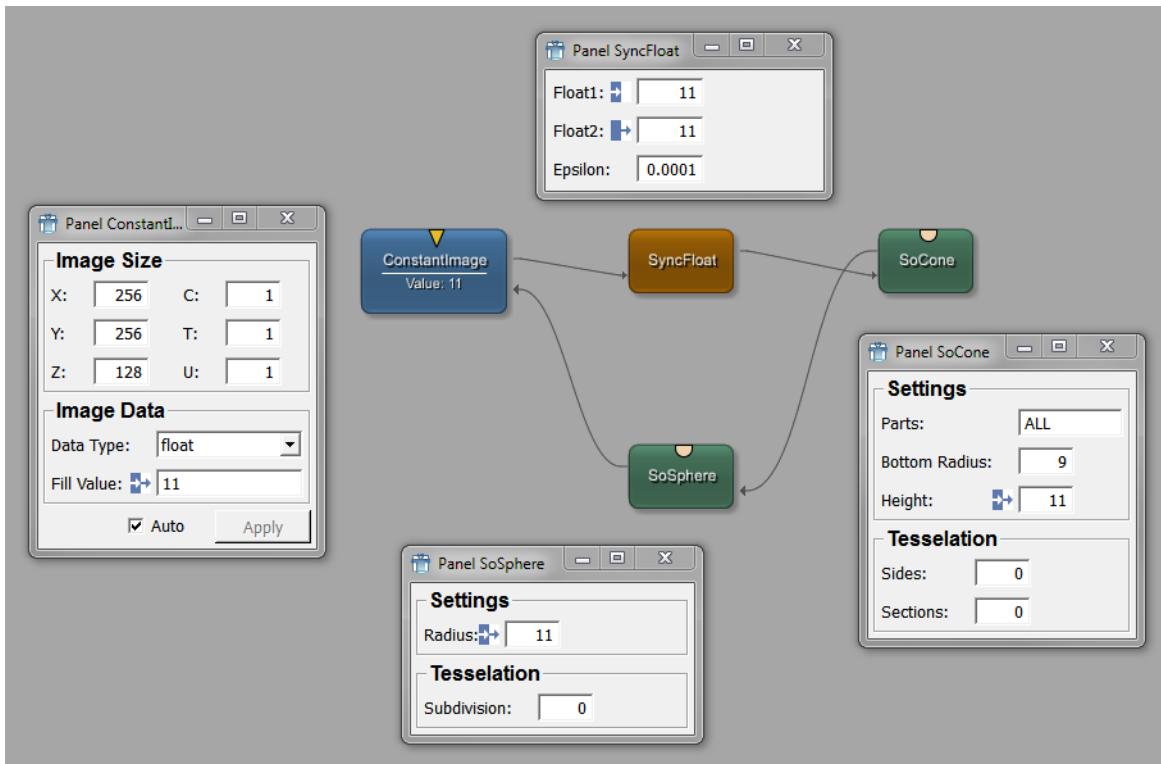
`SyncFloat` is a macro module offering an input parameter `Float1`, an output parameter `Float2` and an `Epsilon` parameter. As long as the difference between the floating point fields `Float1` and `Float2` is smaller than `Epsilon`, `SyncFloat` filters the notifications. Only when the difference between `Float1` and `Float2` is larger than `Epsilon`, notifications are sent. For vectors, the similar module `SyncVector` is available.

In the following paragraphs, the two basic situations in which `SyncFloat` reduces load are described.

27.5.1. Case 1: Two Inventor and One ML Module Connected in a Circle

If a circular parameter connection between two Inventor modules and one ML module is created, the system load will increase because of a constant notification of the fields. To avoid this, the `SyncFloat` module needs to be inserted, either between the Inventor modules or the ML and an Inventor module.

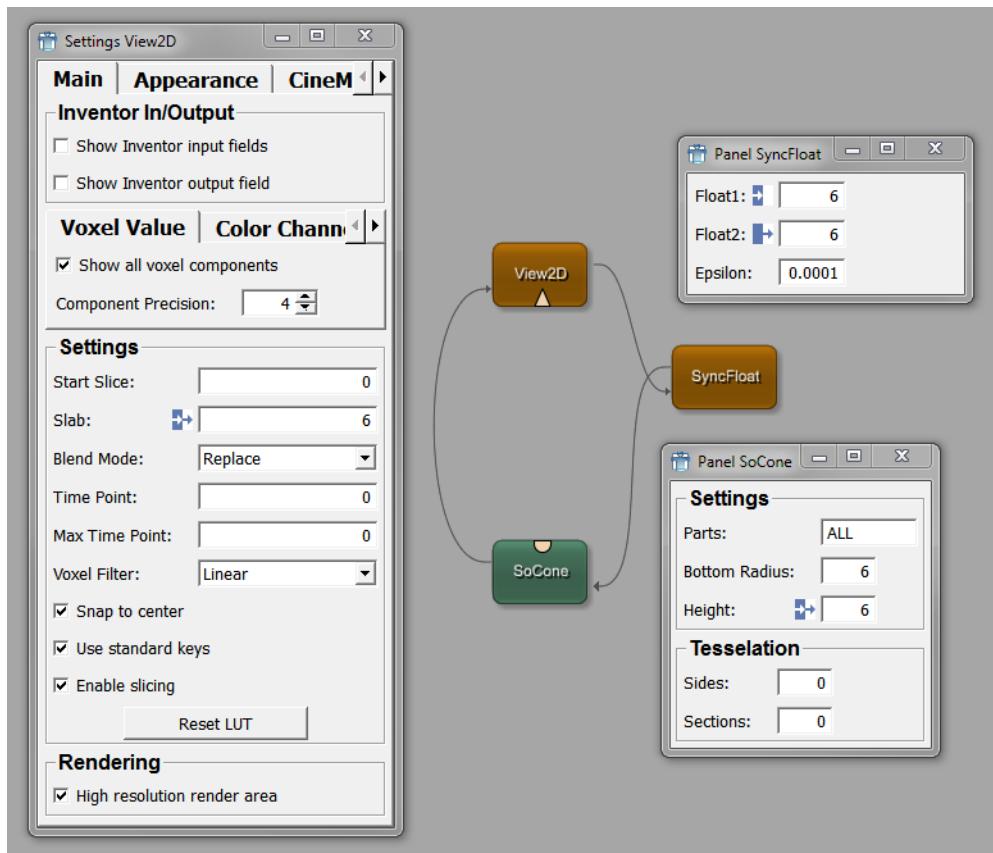
Figure 27.5. SyncFloat Example — ML and Inventor Modules



27.5.2. Case 2: A Macro Module (Including an Inventor Module) and Another Inventor Module Connected in a Circle

If a macro module encapsulates an Inventor module of which some fields can be connected through, and a circular parameter connection is established between the Inventor field of the macro and another Inventor module, this also increases the system load. To avoid this, the `SyncFloat` module needs to be inserted between the macro module and the inventor module.

Figure 27.6. SyncFloat Example — Macro and Inventor Modules



27.6. Printing MeVisLab Networks

MeVisLab offers no print function. Networks have to be captured as image and printed via image processing software.

On Windows, press **Print** and paste the result to Paintbrush or a similar program.

On Linux, use your tool of choice, for example Gimp.



Note

For better printing result, the **Network Rendering Style** in the Preferences can be set to print styles, see [Section 4.3.7, “Preferences — Network Appearance”](#).

27.7. Multi-threading in MeVisLab

27.7.1. Multi-threading in the ML

Since MeVisLab 2.5, a new ML host is available that optimizes multi-threading for page-based image operations.

To enable the new optimized host, go to the **General** tab of the **Preferences** panel ([Section 4.3.1, “Preferences — General”](#)) and uncheck the option to use the classic ML host.

In order to profile modules using multi-threading, use the view described in [Chapter 9, ML Parallel Processing Profiler View](#).

27.7.2. Background Tasks

The new feature Background Tasks allows for sending complete tasks into the background of MeVislab, for example to disconnect calculation tasks from the GUI functionality. However, Background Tasks handle only complete tasks and will not allow breaking a task into a number of parallel processes.

For more about Background Tasks, see [Chapter 7, Background Tasks](#) and the ToolBox Reference, chapter “Background Tasks”.

27.7.3. Modules for Multi-threading

Many modules do not have enabled multi-threading, because it requires good knowledge about common multi-threading pitfalls to decide if a module is already thread-safe or what needs to be done.

27.8. Set Open Inventor Override Flag (Inventor Modules)

In the context menu of Open Inventor modules, the option **Set Open Inventor Override Flag** is available (see [Section 3.9.1, “Module Context Menu”](#)). This option has an effect on how Open Inventor scenes are rendered.

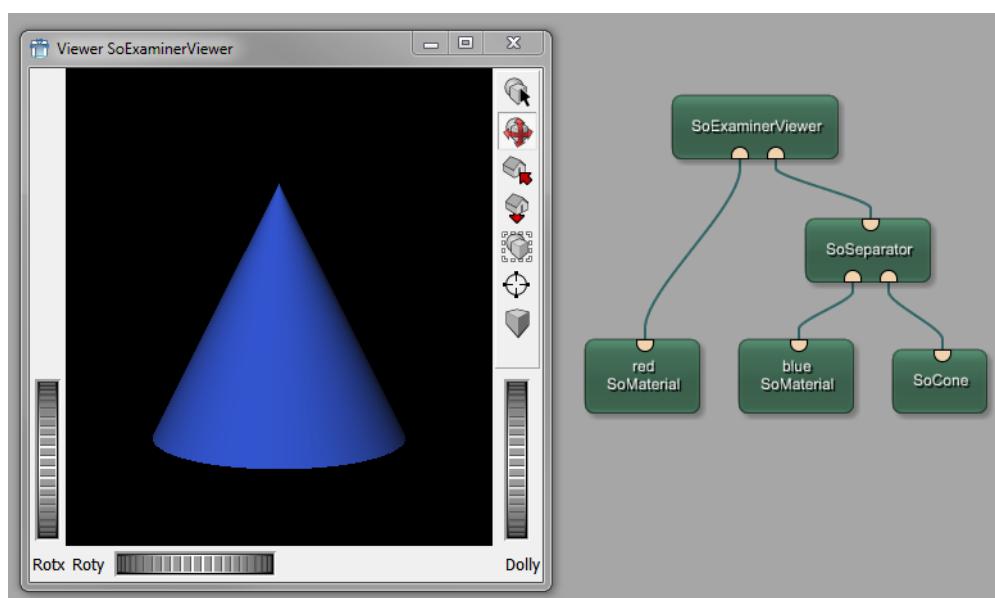


Tip

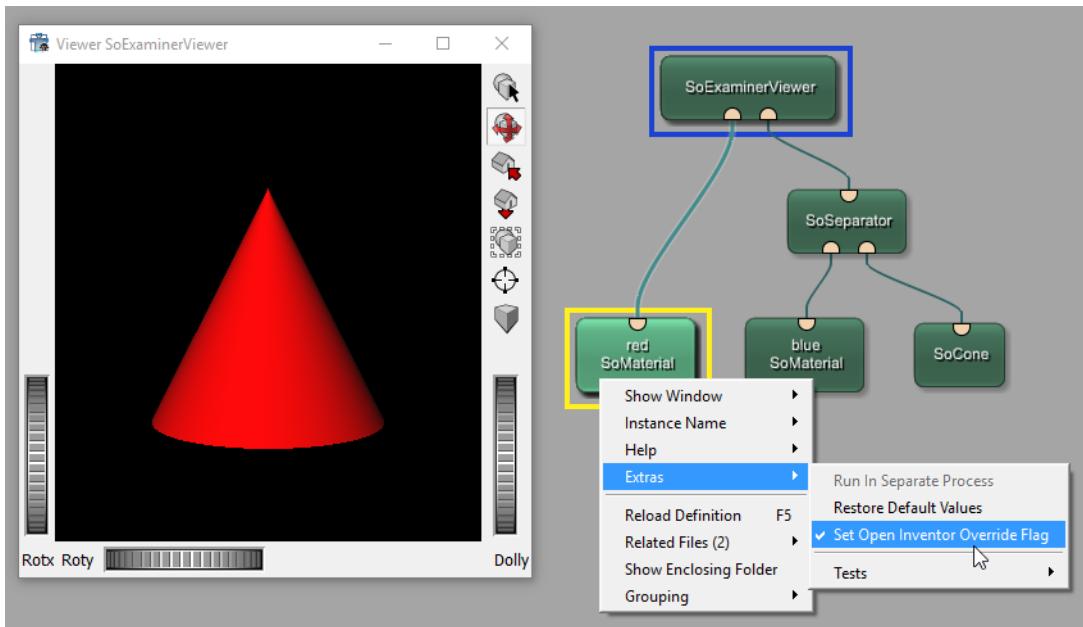
For information on Open Inventor scenes, see Getting Started, chapter “Creating an Open Inventor Scene”, or the Inventor Module help, first chapter.

Here an example for the effects of the **Override** option:

Figure 27.7. Open Inventor Scene Without Override

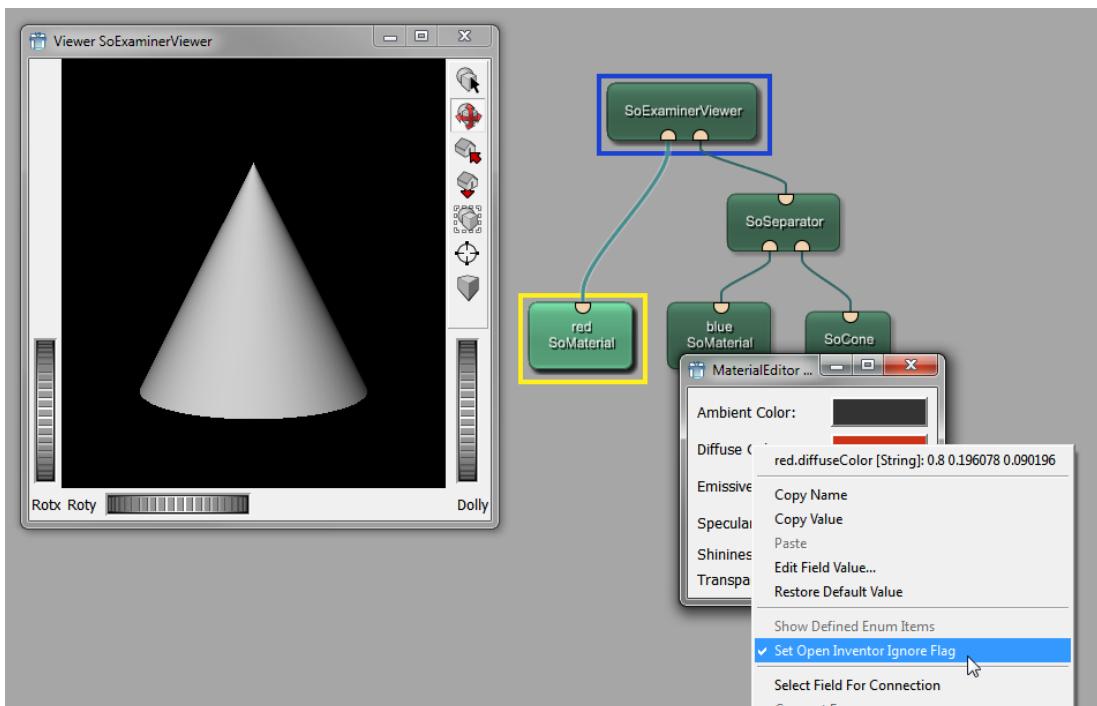


In a normal scene rendering, the blue color overrides the red color. However, if in the context menu of SoMaterial (red) the Override Flag is set, the red color overrides the blue color.

Figure 27.8. Open Inventor Scene With Override**Note**

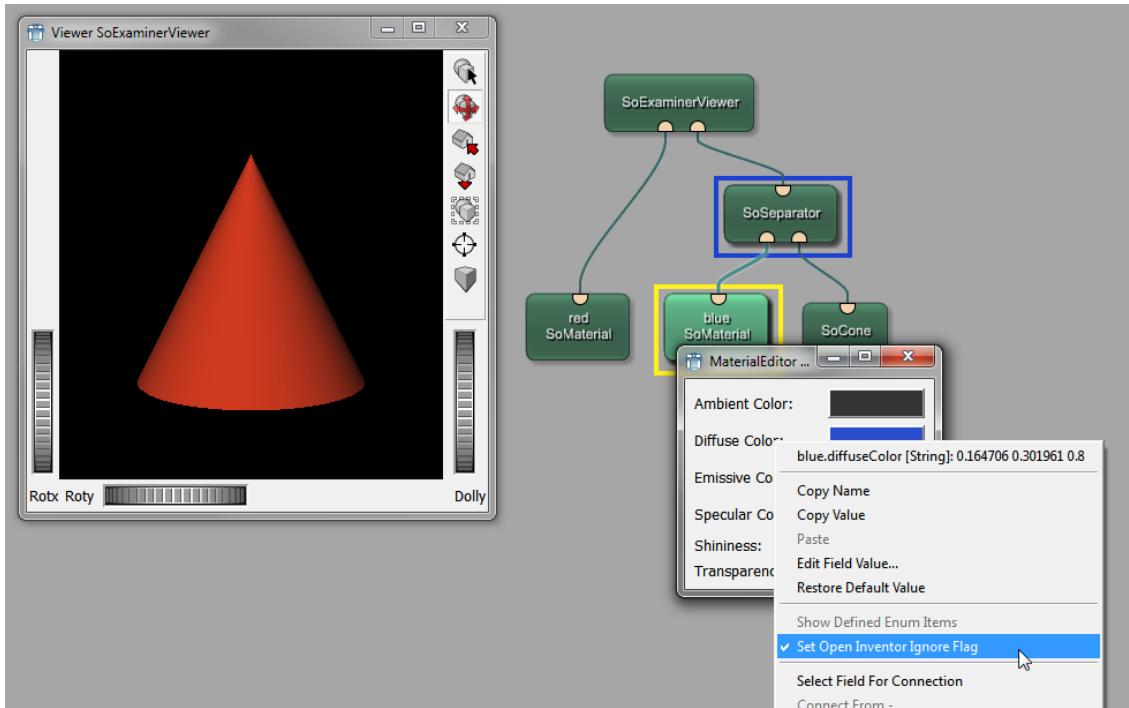
The Override Flag only works for modules within the traversing route. For example, if SoMaterial (red) were connected to a SoSeparator module, the override would have no effect outside this separator.

In addition to the override on module level, an ignore flag for each parameter can be set in the context menu of the automatic panel. In the example with the module override, if the red color is ignored, all colors are overridden with the default gray.

Figure 27.9. Open Inventor Scene With Ignore Flag (Red)

The ignore flag can also be set in the `SoMaterial` (blue) panel. In the example without module override, if the blue color is ignored, the red color is visible.

Figure 27.10. Open Inventor Scene With Ignore Flag (Blue)



Chapter 28. Settings File and Environment Variables

MeVisLab reads a settings file `mevislab.prefs` on start-up that can be used to set certain settings that in some case are not available through the GUI.

There are also some environment variables that change the behavior of MeVisLab in certain places.

28.1. Possible Locations of `mevislab.prefs`

This settings file is searched for in various places. If the file exists in more than one place, all files are evaluated, but if contradicting settings are given, the last file wins. The searched locations are:

- First the directory of the MeVisLab executable (`.../MeVisLab/IDE/bin`).
- Then, depending on the platform:
 - Windows
 1. The user's home directory
 2. The user's document folder
 3. The folder "MeVis" in the user's document folder
 - Linux
 1. Directory `$HOME/.local/share/MeVis/MeVisLab`
 2. The user's document folder
 3. The folder "MeVis" in the user's document folder

(This can be suppressed by giving the `-ignoreprefs` command line option.)

- The file given through the `MEVISLAB_PREFS` environment variable. Also suppressed by `-ignoreprefs`.
- The file given after the `-prefs` command line option.

For applications created with MeVisLab the same search rules apply, but the "MeVisLab" part in the file/directory/environment variable name must be replaced by the application name, with the use of upper/lower/camel-case case corresponding to how MeVisLab is spelled in the rules given above.

28.2. Options in `mevislab.prefs`

The syntax of `mevislab.prefs` is the syntax described in the MDL Reference. All settings must be contained in a "Settings" element like this (example):

```
Settings {  
    // show windows when network is loaded  
    RestorePanels = YES  
  
    // autoreload the mdl/script files when a window of a module is opened  
    AutoReload = YES
```

```
// autosave networks when changed
AutoSave = YES

}
```

The settings file can contain any variable name. The value of a variable can be queried from scripting with `MLAB.hasVariable`, `MLAB.variable`, and `MLAB.variableIsTrue`. Variable names are case sensitive. Besides self-defined variables there are some pre-defined variables for various - sometimes very specialized - purposes, which are listed in the following table. This list does not include (with some exceptions) variables that can also be set from the [Preferences](#) dialog in MeVisLab.

Name	Type	Description
Development		
PackagePaths	complex	<p>This allows one to define additional user packages that should be available in MeVisLab.</p> <p>This attribute has two sub-attributes <code>pathRoot</code> and <code>path</code>. <code>pathRoot</code> gives a root path to which all subsequent <code>path</code> are relative. <code>path</code> should usually be a pair of package group and package, separated by a slash. Example:</p> <pre>PackagePaths { pathRoot = "C:/Users/a_user/Documents/Packages" path = MyPackageGroup/MyPackage pathRoot = "C:/Users/a_user/Documents/Development/MeVisLab" path = MyOrganization/Package1 path = MyOrganization/Package2 }</pre>
LogFile	file path	Write all console output to the file given. Can use <code>\$(LOCAL)</code> to address files relative to the settings file.
RestorePanels	bool	Show module panels when network is loaded. Also available from the Preferences dialog.
AutoReload	bool	Autoreload the <code>mdl</code> /script files of a module when a panel of a module is opened (recommended for fast prototyping, no explicit module "reload" is required). Also available from Preferences dialog.
AutoSave	bool	Auto-save networks when changed. Also available from Preferences dialog.
VerboseScripting	bool	Will print messages when entering and leaving command handling scripting functions.
ShowModulesByUser	bool	By default deprecated modules (or modules in other hidden module groups) do not appear in the module search. If this option is set modules by the current user will always be shown.

Name	Type	Description
		 Note The author name for a module must be exactly the same as the user name in the Preferences dialog.
ExternalDocumentationUrl	url	Override the location of the MeVisLab documentation, giving the base path as an URL. A suitable default would, e.g. be <code>https://www.mevislab.de/docs/current</code> .
UserSpellcheckFile	file path	MeVisLab has an integrated spell-checker (based on hunspell) for english texts in the module help editor. With this option one can add an additional dictionary file.
PreloadModule	module name	With this option, modules can be loaded at startup of the IDE (they are created in some invisible network), which can in turn execute some code and, e.g., initialize preferences variables. This tag can be given several times.
disableImmediateDebug- OutputConsoleRefresh	bool	The MeVisLab debug console by default refreshes immediately after each output to be always up-to-date (e.g. in case of crashes), but this can lead to problems with event handling and is slow. Disable immediate refreshes by setting to YES (then the console is only updated when MeVisLab is idle or processes events for other reasons).
General		
ExtraDllLoadPaths	path	<i>Windows only.</i> Set extra directories from which to load DLLs. (MeVisLab ignores the PATH environment variable since version 3.7.) Several paths can be given separated by ';', but this entry can also be given more than once. It is also possible to reference environment variables in a path with \${}, e.g. \${CUDA_PATH_V11_4}, or even \${PATH}.
MLCacheSizeInMB	integer	Set the cache size for the ML image processing (in MB). Also available from Preferences dialog.
OverrideCursorPosition	directory path	Allows to specify a directory which can contain alternative mouse cursors to be used in Inventor scenes. See the comments in <code>\$(MLAB_ROOT)/MeVis/Foundation/Sources/MLInventorGUICBinding/SoQtViewerProxy.h</code> .

Settings File and
Environment Variables

Name	Type	Description
View2DEnableQtFontRendering	bool	Use Qt for rendering fonts in SoView2D and similar viewers if YES (default). Otherwise uses simpler font rendering engine without Unicode support.
GL2DFont_DefaultSystemFont	name	Font to use for font rendering.
GLDefaultSystemFontProportional	name	Same as above, but only affects the simpler, not Qt-based font rendering.
GlobalScaleFactor	float	Scale all MDL panels with this factor. Default is 1.
qmlStyle	name	Used by the QuickView control, can define the QML style to use.
Deployment		
ReleaseOptimized	bool	If set to YES, MeVisLab will not look for updated mdl/script files after first load. Used for application deployment. (Defaults to YES for applications, NO for IDE.)
LowPriority	bool	Run MeVisLab with a low process priority.
SplashPenColor	hex value	Set the color with which text is drawn on the splash screen. Must be a 6-digit hex value (without '#').
SplashHideMessages	bool	Don't show initialization messages on splash screen.
Advanced		
PythonMultiThreading	bool	Set to NO to disable Python multi-threading support in case of problems.
MLRestrictMaxNumThreads	integer	Set the maximum number of threads to use by the ML image processing pipeline for image for standard image processing. The default is the number of virtual cores for the system.
CoreMaxNumThreads	integer	Set the maximum number of threads to use by the GVR framework. The default is the number of virtual cores for the system. Also available from Preferences dialog.
EnableHighPrecisionLogging-TimeStamps	bool	If enabled time stamps in the log will be printed with a higher precision (precision depending on the system). This can be useful if the exact timing of events is important.
DisableModuleWindowsPersistence	bool	If this option is on, .mlab files won't store the position and state of module panels. This can reduce the changes displayed in diff tools when using version control systems.
FullscreenFlickerFix	bool	Fix flickering of fullscreen panels with OpenGL content (for some setups).
PreferReleaseModeExecutables	bool	If NO, call Debug mode tools (according to debug suffix of executable name) from MeVisLab if MeVisLab is in Debug mode. Otherwise always prefer the faster release mode variant. Default is YES.

Name	Type	Description
StoreModuleCacheInUserLibrary	bool	Whether to store the module cache files in a directory outside the installation directory. Default is NO.
DontShowIconsInMenus	bool	If this option is set, pop-up menus don't show icons for their entries (on some platforms this is a no-op, as there are no icons shown anyway).
DisableLicenseExpirationWarning	bool	If this option is set, MeVisLab (and applications based on MeVisLab) will not display a warning if the current license is about to expire.

28.3. Environment variables

Similar to the values in the `mevislab.prefs` file there are also some settings which can be set through environment variables, mostly to correct some misbehavior of the platform. We have collected some here that might be of use for you:

Name	Type	Description
General		
MLAB_ROOT	path	The path where the MeVisLab packages are installed. This must be set if you call some shell scripts that, e.g., create compiler projects or generate installers. The Windows installer will set this automatically.
Fixes		
MLAB_FORCE_MESA	bool	(Windows only) If set to 1, MeVisLab will always try to use the Mesa software OpenGL driver supplied with the SDK. This can be used if the system OpenGL driver does not work.
MLAB_QT_OPENGL_WIDGET	"old" or "new"	Qt deprecated the old OpenGL widget with Qt 5. Since the first versions of the new widget had problems on some systems, you can switch back to the old widget by setting the value "old" - but since the widget is deprecated you might run into other problems.
MLAB_OPENGL_10BIT	bool	Force MeVisLab to support 10bit color depth in OpenGL in case the auto-detection fails.
MLAB_TTF_FONT	path	Override font to use for font rendering. Must be a true type font.
MLAB_DISABLE_BUSY_CURSOR	bool	(Linux and Windows only) Disable display of busy cursor when MeVisLab is calculating.
MLAB_SOQT_ROUNDUP_WHEEL_DELTA	bool	Fixes misbehaving mouse wheel in Inventor views (for some buggy mouse drivers).
SOVIEW2D_NO_SHADER	bool	Do not use OpenGL shaders in SoView2D views.

Name	Type	Description
Debugging		
MEVISLAB_DEBUGGER	string	(Linux only) Set debugger command to call if MeVisLab crashes. Command is called with executable name and process ID.
MLAB_DEBUG_PYTHON_IMPORT	string	Print additional information about imports happening in Python code.
IV_DEBUG_SHADER, IV_DEBUG_SHADER_LOG, IV_DEBUG_SHADER_STRING	bool	(Used by SoShader framework) Get additional log output to debug shaders.
GVR (Giga Voxel Renderer) - OpenGL Compatibility		
GVR_NO_3D_TEXTURES	bool	Do not use 3D textures in the GVR rendering.
GVR_NO_GLSL	bool	Do not use OpenGL shader language.
GVR_NO_NONPOW2	bool	Do not use texture that do not have a size that is a power of two.
GVR_NO_GEOMETRYSHADER	bool	Do not use geometry shaders.
GVR_USE_FLOAT_LUT	bool	Use look-up-tables with float values (instead of integer).
GVR_NO_BINDLESS	bool	Do not use bindless textures.
GVR_PRINT_SHADER_WARNINGS	bool	Prints additional diagnostic messages when OpenGL shaders are compiled.
Special Settings		
MLAB_GPU_AFFINITY	int	(NVidia graphics only) Force MeVisLab to run on a specific graphics card if several are installed on a system.
MLAB_CUDA_DEVICE	int	(Used by PathTracer framework) Select the CUDA device to do calculations on.
MLAB_NUMBER_CONCURRENT_PANEL_RENDER	int	(Used by RemotePanelRendering module) Serialize GPU access of parallel running processes.

Chapter 29. Coding Style Guide

(Version as of 2010/04/22.)

Aspects covered in this chapter:

- [Section 29.1, “General Rules”](#)
- [Section 29.2, “Naming Conventions”](#)
- [Section 29.2.1, “Code”](#)
- [Section 29.2.2, “Files”](#)
- [Section 29.2.3, “MeVisLab/ML Identifiers”](#)
- [Section 29.2.4, “MDL Layout”](#)
- [Section 29.3, “General Programming”](#)
- [Section 29.4, “C++”](#)
- [Section 29.5, “Python”](#)
- [Section 29.6, “ML Programming”](#)
- [Section 29.7, “MDL”](#)

Legend:

- (!) - Major rule: It is really important that your code conforms to these rules. You must have a very good justification for the violation of a major rule.
- (+) - Minor rule: Conformance to these rules is important, but there may be common situations in which the violation of a minor rule is well justified. It is not allowed, however, to violate a minor rule without a good reason.

Meta Rules

- (!) M1: Existing, non-compliant code should be adapted when large changes are applied. Consider refactoring.
- (!) M2: Changes to other people's code should preserve their style closely.
- (!) M3: Personal source code does not fall under this Coding Style Guide, but it is good practice to consider this guide's rules in personal code as well.

29.1. General Rules

- (!) 01.01: Write everything in English (comments, names, variables, documentation...).
- (!) 01.02: Be consistent. Even if you decide not to follow some of the rules, do it at least consistently. Never produce code with different styles of programming.
- (!) 01.03: Insert the copyright placeholder and the recommended file header at the top of files. Do not use old copyright headers.

29.2. Naming Conventions

The following spelling schemes are used:

- UpperCamelCase: Upper case at beginning and new words

- lowerCamelCase: Lower case at beginning, upper case at new words
- ALL_UPPER_CASE: All upper case, words separated by underscore ("_")

29.2.1. Code

- (!) 02.01.01: Class names: UpperCamelCase
- (!) 02.01.02: Typedefs: UpperCamelCase or lowerCamelCase
- (!) 02.01.03: Defines: ALL_UPPER_CASE
- (!) 02.01.04: Enum values: UpperCamelCase or ALL_UPPER_CASE
- (!) 02.01.05: Methods, functions, variables: lowerCamelCase
- (!) 02.01.06: Private and protected member variables are preceded by an underscore ("_")
- (+) 02.01.08: Use an underscore prefix ("_") for methods, macros and variables that are "semantically private", i.e. are meant for internal usage by the class and not as part of the class interface.
- (+) 02.01.09: Use common speaking names for methods:
 - get/set for methods that get/set member variables/states.
 - Instead of get, the alternative of using just the variable name as method name is allowed (as in Qt).
 - is/has/are/should to access boolean states depending on the context.
 - Only use common abbreviations, keep the code readable to others, don't be creative here.

29.2.2. Files

- (!) 02.02.01: Do not use umlauts, whitespace or other uncommon characters in file or directory names.
- (!) 02.02.02: Use identical upper/lower case spelling in header and source files.
- (!) 02.02.03: Use "/" instead of "\" as path separator in include statements.
- (!) 02.02.04: Use "ml" prefix for everything that links to the ML.
- (+) 02.02.05: Upper and lower case prefixes: Use the lower case prefix "ml" for all C/C++ files (for example mlMyModule.h) and the upper case prefix "ML" for the rest (for example .bat, .dll, .def, etc.) and for directory names.

29.2.3. MeVisLab/ML Identifiers

- (!) 02.03.01: Field variables: lowerCamelCase with "Fld" at the end (register field names without "Fld").
- (!) 02.03.02: Enum item tokens: ALL_UPPER_CASE in Inventor fields, UpperCamelCase in other fields
- (!) 02.03.03: Module input/output fields: Prefix "in"/"out" (not "input" or "output").
- (+) 02.03.04: Recommended names for input/output fields:
 - inImage, inObject, inMask, inLabel, inMarker, ...
 - outImage, outObject, ...
 - outNode for Inventor outputs, possibly outViewNode, outEditNode, ...
- (+) 02.03.05: Apply and Update:
 - o Use Apply for the action of applying modifications made through a module's UI (parameter fields)
 - o Use Update for the action of updating a result from a modified input object (for example input image).
- (+) 02.03.06: Auto-Apply/Update: If appropriate, provide boolean fields autoApply and enum fields updateMode with items "Off", "AutoClear" and "AutoUpdate".

29.2.4. MDL Layout

- (!) 02.04.01: Field titles: Use "Headline Spelling", i.e. upper case at beginning and new words (except for words as "in", "and", etc.), words separated by spaces. Add a trailing colon (:).
- (!) 02.04.02: CheckBox labels: Use correct orthographic spelling, i.e. upper case only at the beginning, words separated by spaces. No trailing colon.
- (!) 02.04.03: Box, tabs and other grouping labels: Same as field titles ("Headline Spelling"), but no trailing colon.
- (+) 02.04.04: Use "CheckBox" instead of "Field" controls for boolean fields.
- (+) 02.04.05: Avoid inconsistencies between UI labels and field names. Let MeVisLab generate the UI label from the field name, or at least choose a label that is close to the field name.

29.3. General Programming

- (!) 03.01: Use spaces instead of tab characters for indentation. The recommendation is two space characters for each level of indentation (four for Python code).
- (!) 03.02: Comment your code using Doxygen-style formatting. Read the Doxygen documentation.
- (!) 03.03: Provide appropriate error handling.
- (!) 03.04: Use speaking and consistent variable names and other identifiers.
- (!) 03.05: Enclose code blocks in braces after all flow control statements, even in one-line constructs.
- (+) 03.06: Discuss your design decisions with other developers in an early stage of your project.
- (+) 03.07: Keep methods and functions overseeable. Split long code segments into functions, use refactoring for "spaghetti code".
- (+) 03.08: Put concatenated boolean expressions in parentheses.
- (+) 03.10: Avoid long lines. 80-100 characters for a line of code should be enough. Split longer lines into several statements or split them up into several lines using one level of indentation.
- (+) 03.11: Avoid redundancy in class data storage as far as possible.
- (+) 03.12: Use "return" only at the end of methods or functions in order to gain a better control of resource management.

29.4. C++

- (!) 04.01: Never leave pointers uninitialized.
- (!) 04.02: Never allow dangling pointers. If objects are destroyed or invalidated, all pointers to it must be cleared.
- (!) 04.03: Use NULL for null pointers, not the numerical value 0.
- (!) 04.04: Always initialize variables at declaration.
- (!) 04.05: Provide a virtual destructor if a class contains virtual methods.
- (!) 04.06: Define typedef and enumerated types in an appropriate class scope to avoid name space conflicts/pollution.
- (!) 04.07: Keep the scope of objects as small as possible.
- (!) 04.08: Use "#pragma once" in header files to avoid duplicate inclusion.
- (!) 04.09: Initialize all member variables in the constructor.
- (+) 04.10: If the initializer list is used in a constructor, initialize members in the correct order.
- (+) 04.11: Consider use of exception handling to catch runtime errors.
- (+) 04.12: Be careful with public member variables. Consider using get/set methods instead.
- (+) 04.13: Use minimal and appropriate types.
- (+) 04.14: Use references instead of pointers whenever possible.
- (+) 04.15: "Maximum const" philosophy: Use as much const qualifiers as possible to avoid unintentional changes.
- (+) 04.16: Use "const" constants instead of #define. Defines tend to conflict with other peoples defines and are not type safe.
- (+) 04.17: Use inline functions instead of macros. Macros are difficult to debug and often suffer from different semantics in the pre-processors.
- (+) 04.18: Put all code into .cpp file. No code in include files, except for short inline and template code.
- (+) 04.19: One class per source and include file.
- (+) 04.20: Append a newline at the end of your files to avoid compiler warnings.
- (+) 04.21: Switch statements should always have a default case, if not all cases are handled explicitly.

29.5. Python

- (+) 05.01: Create modules instead of plain scripts.
- (+) 05.02: Do not use optimization for Python.
- (+) 05.03: Use Python style documentation for Python.
- (+) 05.04: Document all imports at the top of the file.

29.6. ML Programming

- (+) 06.01: Use ML error handling macros (for example ML_PRINT_FATAL_ERROR) to make post mortem analysis easier.
- (+) 06.02: Consider using ML datatypes instead of C/C++-types. ML datatypes are more platform-independent.

29.7. MDL

- (!) 07.01: Do not embed scripting in MDL files, except for very simple Python one-liners.
- (+) 07.02: Use one line per statement.