ToolRunner Documentation

ToolRunner Documentation

Table of Contents

List of Figures

Chapter 1. Introduction to ToolRunner

1.1. What is ToolRunner?

ToolRunner is a meta tool to

- from `.pro` files, create project files. Via these platform-dependent files, libraries, DLLs, as well as MS Visual Studio solutions (for Windows) and Xcode Workspaces (for macOS) can be created. The platform-*dependent* project files may be of the type
  - (Windows) `.vcxproj` for the Visual Studio version of the installed MeVisLab. Set in the environment variable MLAB_COMPILER_VERSION.
  - (macOS) `.xcodeproj` for an Xcode project or a `Makefile`. (Note: On macOS the MeVisLab Project Generator offers a Finder integration to create Xcode projects)
  - (Linux) Makefile.
- via `.mldoc` files, build documentation based on Doxygen or Docbook.
- via `.mlinstall` files, build an installer for compiled sources (requires an MeVisLab ADK licence).
- via `.mlbuild` files, build the projects with the Master Builder.

ToolRunner is available for all supported platforms but the options may differ.

1.1.1. ToolRunner Features and Utilities

Basically, ToolRunner handles a lot of scripts. In addition, however, it offers supporting features and utilities like

- an easy-to-use file list, see File menu,
- a Tools Check utility, see Tools Check,
- check and dependency analysis for `.pro` files,
- a Package Manager for checking whether the packages are "visible" to the tools, see Package Manager
- license creation (needs the MeVisLab ADK).

1.1.2. How ToolRunner Finds Packages

ToolRunner typically works on MeVisLab packages but can be used for any supported file type. It checks for available packages in the same way as MeVisLab

- the directories given in the *PackagePaths* settings of `mevislab.prefs`.
- the *Packages* directory in which MeVisLab was installed.
- the *UserPackagePath* (as set in the MeVisLab Preferences dialog)

Note

Packages installed with MeVisLab are usually not offered in ToolRunner dialogs as they do not contain sources.

1.2. Starting ToolRunner

ToolRunner is installed with MeVisLab. You can start ToolRunner

- from within MeVisLab, menu **File** → **Run ToolRunner**
- from the MeVisLab Installer Wizard and other similar places where ToolRunner features are used.

1.2.1. Windows

On Windows, you can also start ToolRunner

- by clicking the ToolRunner desktop icon (if installed accordingly).
- from the start menu (if installed accordingly).
- from the context menu of Windows Explorer (if "Developer file association" was selected in the MeVisLab SDK setup).

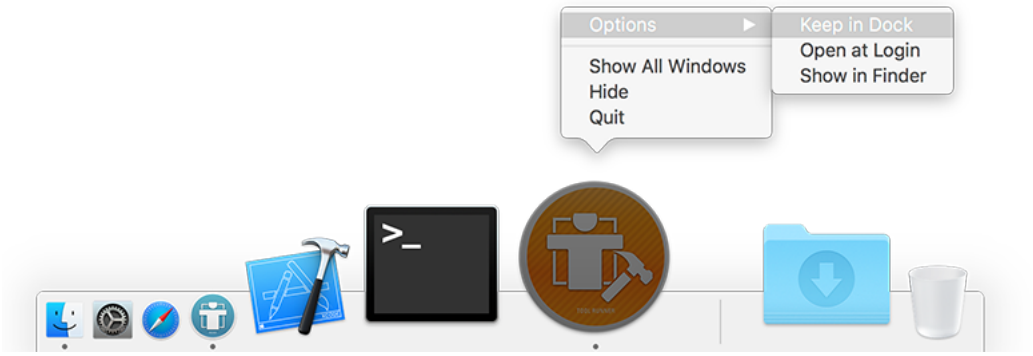
1.2.2. macOS

On macOS, you can also start ToolRunner

- via the associated files. Simply double-click a file with a type that is supported by ToolRunner.
- from the Dock, if added. To add the ToolRunner icon to the Dock

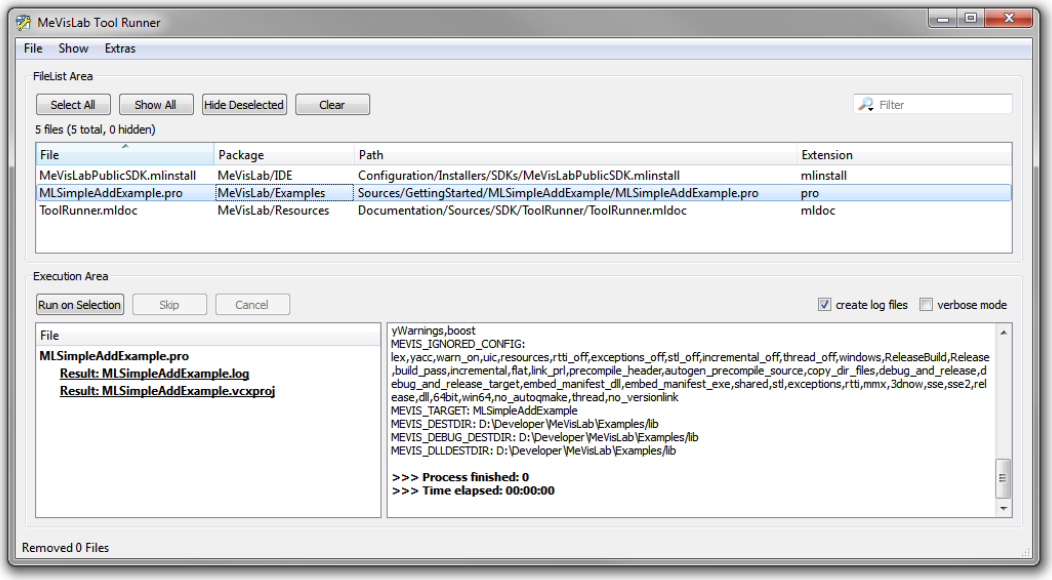
  1. start ToolRunner from the File menu of MeVisLab.
  2. choose **Keep in Dock** from the Dock context menu to permanently glue the ToolRunner icon to the Dock.

Figure 1.1. Adding ToolRunner to the Dock



1.3. The ToolRunner GUI

Figure 1.2. ToolRunner Main Window



The main menu offers general file handling, preferences, display options, and extras. See Main menu.

1.3.1. FileList Area

The FileList area is dedicated to file handling:

- Button menu for file list: Shortcut buttons for some typical functions; also a filter option for filtering the displayed files.
- File list: Lists files for the process. Can be saved as `.mlfilelist` file. For each file, the name, package, path, and extension are displayed.

It is possible to select several or all files in the file list and start the according processes.

To add files to the file list,

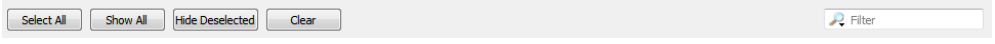
- select **File** → **Add Files from Directory** or **File** → **Add Files from Packages**.
- or directly drag files from a file browser into the file list.

If more files than necessary are in the list, you can

- delete files from the list by selecting them and pressing DEL.
- click **Clear** to remove them all.
- select the wanted files and click **Hide Deselected** to hide the others.
- enter a filter term to narrow the list.

1.3.1.1. File List Button Menu

Figure 1.3. File List Button Menu



Select All: Selects all files in the file list.

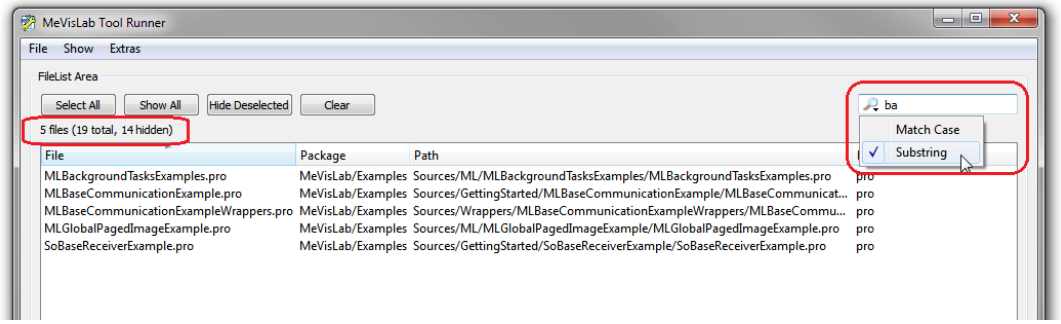
Show All: Displays all files. Same effect as **Show** → **Show All**.

Hide Deselected: Hides all files in the file list that are currently not selected. This function works repeatedly so that you can hide more and more files in the process. Same effect as **Show** → **Hide Deselected**.

Clear: Clears the currently displayed list of files (will not delete saved `.mlfilelist` files). Same effect as **File** → **Clear FileList**.

Filter: In the filter field, you can enter terms to filter the list. The filter will be applied immediately.

Figure 1.4. Filter



Options:

- **Match Case**: Check to take the lower/upper case of the entered filter term into account.
- **Substring**: Check to use the filter term like in a wildcard search.

1.3.2. Execution Area

The Execution area is dedicated to the execution of build processes.

- Button menu for starting the process(es): Starts, skips, and cancels the default processes for the selected files (e.g., **Run on Selection** will generate, e.g., a `.vcxproj` file from a `.pro` file but HTML and PDF files from `.mldoc` (documentation) files); also contains options for log file creation and verbose mode.
- Output area for files: Displays the results of the process and the link to the report. The output file names are links; double-click them to open the files.
- Output area for process information: Displays information for each process in order of processing. If several processes are running at the same time, the information will get mixed up.

You can edit the file list while a process is still running. We recommend that you check the **create log files** option in this case, which will result in one log file per processed file and facilitates later analysis of failed processes.

1.3.2.1. Run Button Menu

Figure 1.5. Run Button Menu

Run on Selection: Starts the process based on the currently selected files.

Skip: Skips the currently processed file and proceeds with the next file.

Cancel: Cancels the whole process.

Options:

- **Create log files**: Check to create log files (`.log`) for each processed file. The log files will be stored in same directory as the output.
- **Verbose mode**: Check to set the verbose mode. Every processing step will be listed explicitly.

1.4. Using ToolRunner

1.4.1. Add Files

First, add files to the file list. They can be of the four file types, see File Types.

1.4.2. Run the Process

This can either mean you "run" the files, or you build an installer or executable with the Master Builder. Select your options from the context menu.

The packages processed will be displayed in the area on the lower left. The tool output during the run will be displayed in the area on the lower right.

1.4.3. Check Results

The resulting files are saved. Their location depends on the file types.

After a successful run/build, a report is created as an HTML file at the destination specified in **URL to Report Directory** in **File** → **ToolRunner Preferences**.

Chapter 2. Supported File Types

ToolRunner supports the following file types:

- `.pro`: MeVisLab profile file containing the platform-independent project description.
- `.mldoc`: MeVisLab configuration file for Docbook or Doxygen projects, to create help systems (HTML and PDF output).
- `.mlinstall`: MeVisLab installer file containing the platform-independent information for the build of an installer (i.e., combining compiled files).
- `.mlbuild`: MeVisLab build configuration file created in a Master Builder run. Can be used to start the build process.

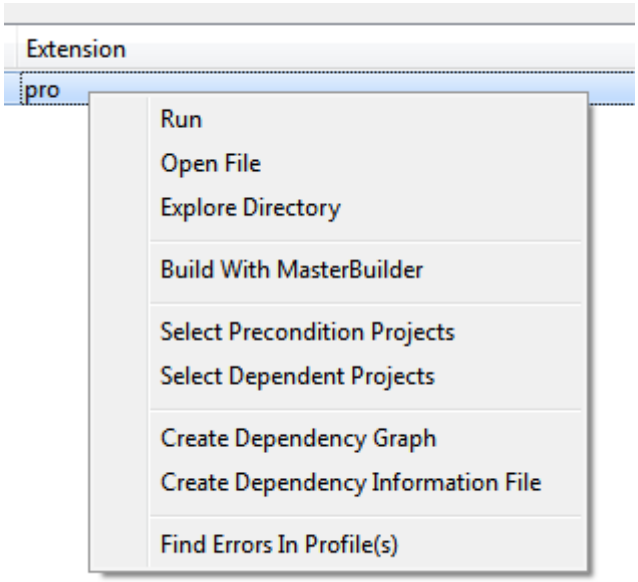
In the context menu of each file you can find the options available for this file type. In case of multiple selected files, the resulting context menu will show all available options.

In addition you can save the file list in the `.mlfilelist` format (see File menu). `.mlfilelist` is a list in plain ASCII of all file paths relative to the `.mlfilelist` file.

Only on Windows: if the relative path cannot be resolved (i.e., in case of a drive change), an absolute path will be used.

2.1. .pro

Figure 2.1. .pro Context Menu



Run: Run the `.pro` file to create the platform-dependent project file:

- (Windows): `.vcxproj` for for the Visual Studio version of the installed MeVisLab. Set in the environment variable MLAB_COMPILER_VERSION.
- (macOS) Makefile (use the MeVisLab Project Generator to create Xcode projects).
- (Linux) Makefile.

Project files (and log files, if written) are saved in the same path as the `.pro` file.

Open File: Opens the file in a program of your choice, e.g., a text editor.

Explore Directory: Opens the current directory with a standard file browser.

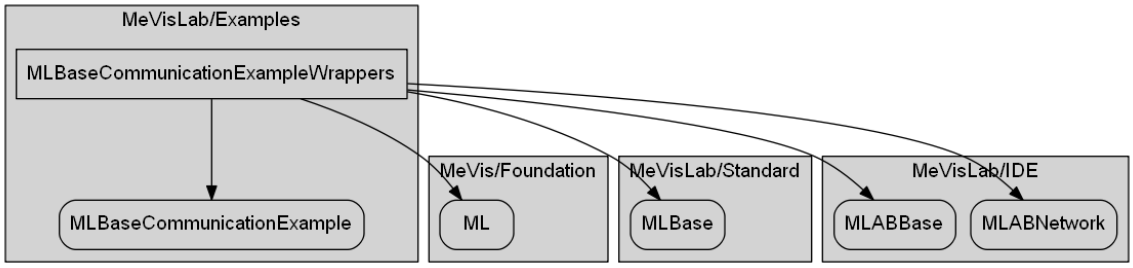
Build With Master Builder: Builds with the Master Builder (all tools of the Master Builder must be available). The Master Builder generates the platform-*dependent* project file, an `.mlbuild` Master Builder file, and starts the build itself.

Select Precondition Projects:: Displays and selects the precondition project(s) in the file list. Precondition projects are necessary for the successful build of the current project. They are listed in the `CONFIG` entries of the currently selected `.pro` file(s).

Select Dependent Projects: Displays and selects the dependent project(s) in the file list. Dependent projects rely on the successful build of the current project. The `CONFIG` entries of the dependent `.pro` file(s) list the currently selected `.pro` file(s).

Create Dependency Graph: Generates a graphic of the dependencies (uses GraphViz) and a document with the processed script. Example:

Figure 2.2. Dependency Graph for the an Example Project

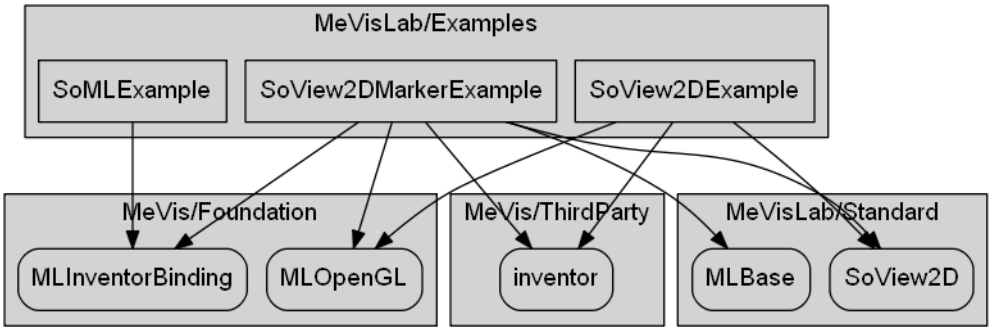


Corresponding entry:

```
CONFIG += dll ML MLBase MLABNetwork MLABBase QtNeeded
CONFIG += MLBaseCommunicationExample
```

You can also select several files at the same time and create a dependency graph. This will result in complex overview graphs:

Figure 2.3. Dependency Graph for Multiple Example Projects



In the graphs, modules drawn with round edges are only known to exist by the dependent project but their own dependencies are unknown/unresolved. Modules in a rectangle are "known", i.e., their `.pro` files were read and the dependencies resolved.

Create Dependency Information File: Creates and saves an ASCII file that contains the dependency information of the selected project(s).

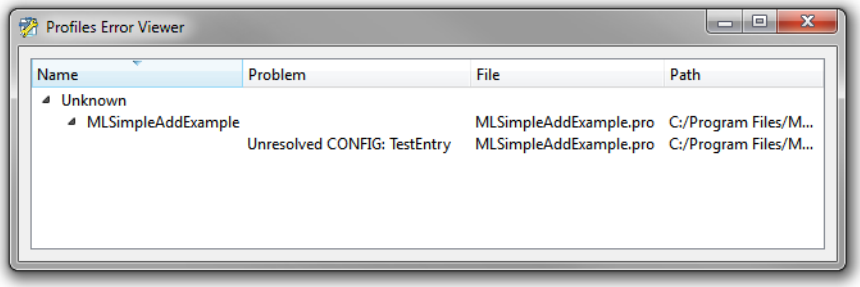
Find Error(s) in Profile: Checks the selected `.pro` files for errors in the dependency definitions, especially in the following lines:

```
CONFIG += dll ML MLTools MLBase ...

MLAB_PACKAGES += MeVisLab_Standard ...
```

The Profiles Error Viewer opens.

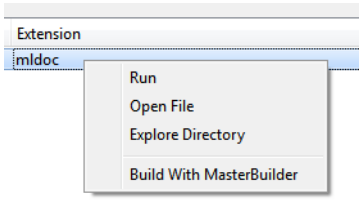
Figure 2.4. Profiles Error Viewer



The Error Viewer is non-modal, so you can keep several of them open while you edit the profiles.

2.2. .mldoc

Figure 2.5. .mldoc Context Menu



Run: Runs the `.mldoc` file to build the documentation with the Documentation tools.

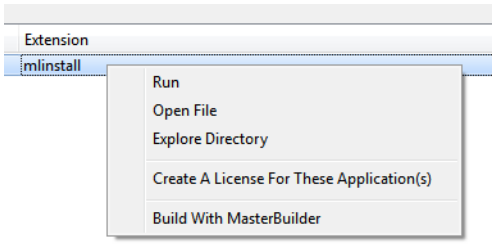
Open File: Opens the file in a program of your choice, e.g., a text editor.

Explore Directory: Opens the current directory with a standard file browser.

Build With Master Builder: Builds the documentation with the Master Builder.

2.3. .mlinstall

Figure 2.6. .mlinstall Context Menu



Run: Runs the `.mlinstall` file to build the install file with the Installer tools.

Open File: Opens the file in a program of your choice, e.g., a text editor.

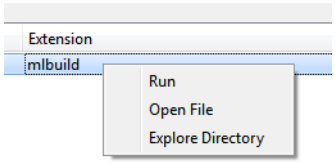
Explore Directory: Opens the current directory with a standard file browser.

Create A License For These Application(s): Creates a license for the applications that are part of the installer. Requires the Application License Manager program.

Build With Master Builder: Builds the installer with the Master Builder.

2.4. .mlbuild

Figure 2.7. .mlbuild Context Menu



Run: Runs the `.mlbuild` file to start the build process.

Open File: Opens the file in a program of your choice, e.g., a text editor.

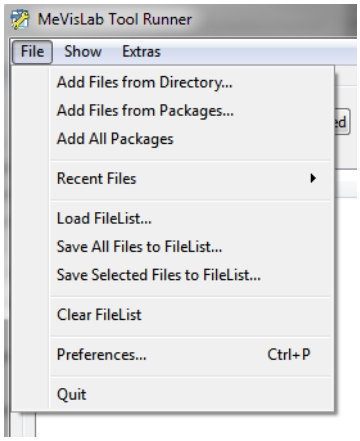
Explore Directory: Opens the current directory with a standard file browser.

Build With Master Builder: Builds the installer with the Master Builder.

Chapter 3. Main Menu

3.1. File Menu

Figure 3.1. File Menu



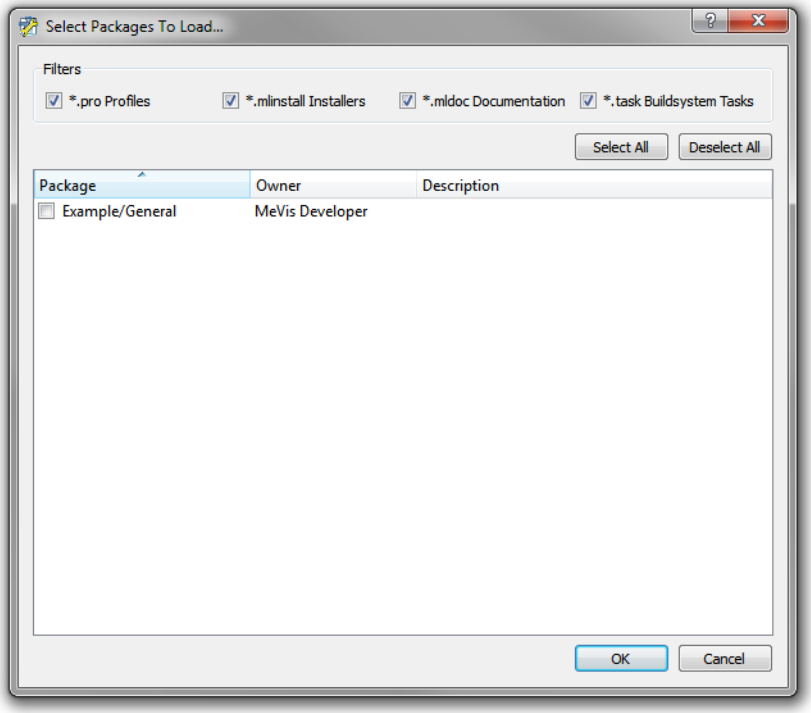
3.1.1. Add Files from Directory...

Adds files from any directory to the file list. Opens a standard file browser. Use this option if you want to add `.mlbuild` files.

3.1.2. Add Files from Packages...

Adds files from packages found in the scanned paths to the file list. Opens the Package Loader dialog.

Figure 3.2. Package Loader



3.1.3. Add All Packages

Directly adds all packages found in the scanned paths to the file list.

3.1.4. Recent Files

Opens the list of recently used files, e.g., an `.mlfilelist` file.

3.1.5. Load FileList...

Loads a previously saved `.mlfilelist` file. Opens a standard file browser.

3.1.6. Save all Files to FileList

Saves all files in a `.mlfilelist` file. Opens a standard file dialog.

3.1.7. Save Selected Files to FileList

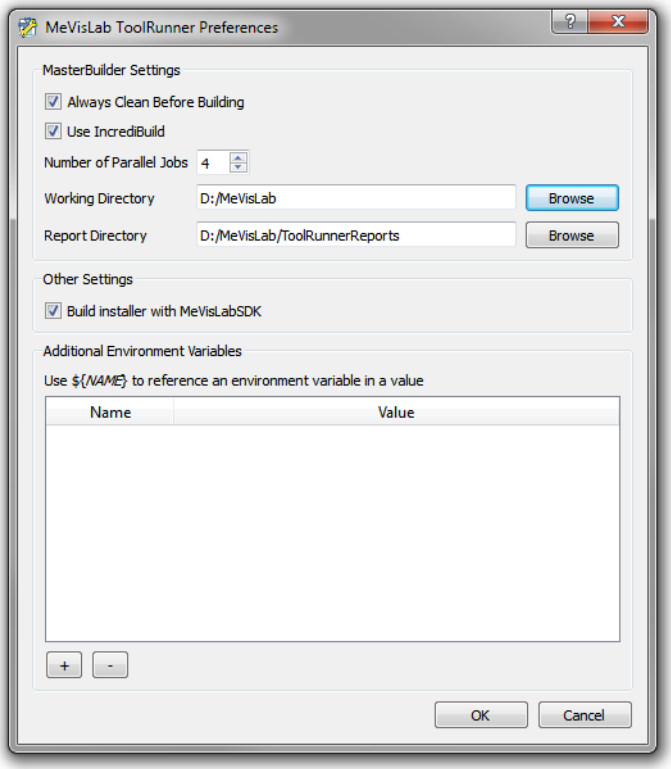
Saves the selected files only in a `.mlfilelist` file. Opens a standard file dialog.

3.1.8. Clear FileList

Clears the currently displayed list of files (will not delete saved `.mlfilelist` files).

3.1.9. Preferences...

Figure 3.3. ToolRunner Preferences



Always Clean Before Building: (only relevant for .pro files) Check if already built project files should be discarded. Otherwise, the build is done incrementally.

Use IncrediBuild: (Windows) Check if you want to build with IncrediBuild (must be installed to work).

Number of Parallel Make Jobs: Enter the number of parallel make jobs. Unix platforms use makefiles here.

Working Directory: Path to the working directory of the Master Builder in which local files are stored. The contents of this directory will be discarded for each new build process.

Report Directory: Path to the report output of the Master Builder.

Enter all paths or click **Browse** to open the standard file browser.

Build Installer with MeVisLabSDK: Builds installers with files from the installed MeVisLab SDK. If not checked, a special build setup is needed on the computer.

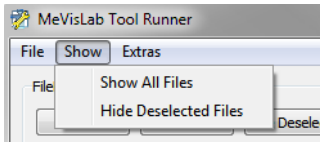
Additional Environment Variables: Allows to set additional environment variables for the build process from the ToolRunner. Press the **+** button to add a new entry, press **-** button to remove the currently selected entry.

3.1.10. Quit

Quit ToolRunner.

3.2. Show Menu

Figure 3.4. Show Menu



3.2.1. Show All Files

Displays all files, including those that have been hidden.

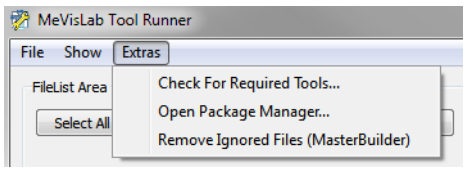
3.2.2. Hide Deselected Files

Hides all files in the file list that are currently not selected. This function works repeatedly so that you can hide more and more files in the process.

Hidden files will not be used for a build.

3.3. Extras Menu

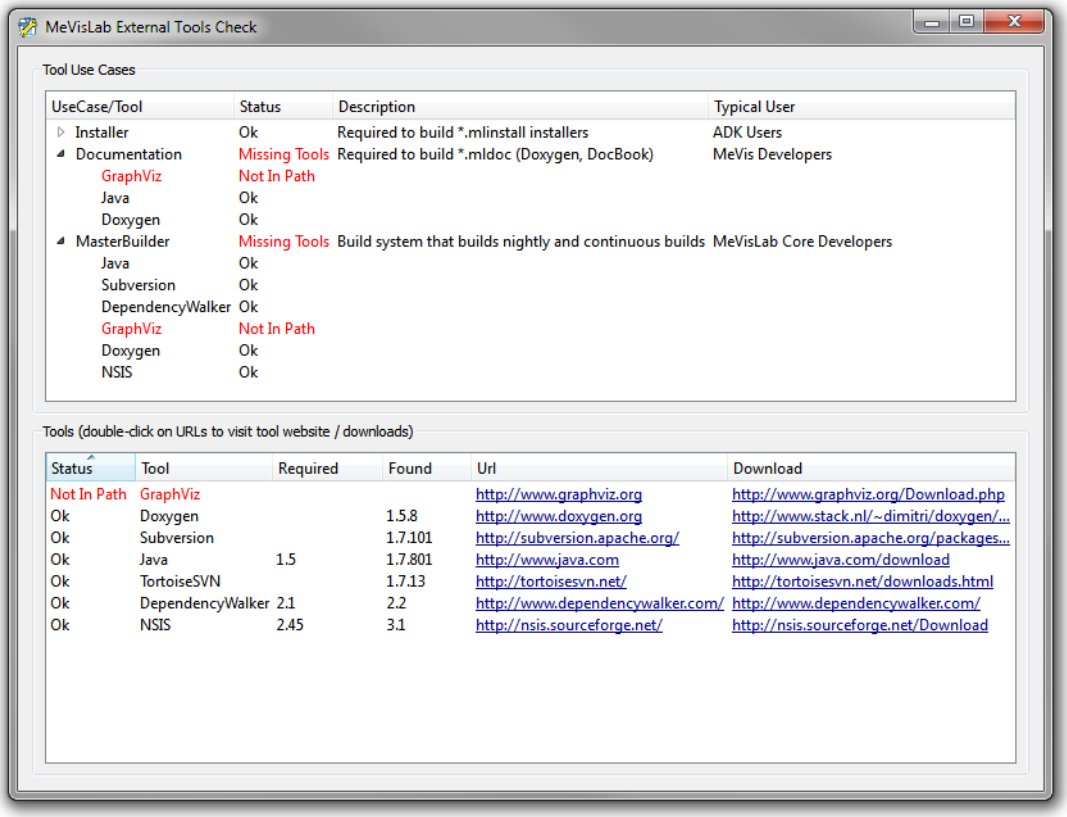
Figure 3.5. Extras Menu



3.3.1. Check for Required Tools

Check for Required Tools: Depending on what you want to use ToolRunner for, various tools are necessary. This function runs a check on the available tools. Example output:

Figure 3.6. External Tools Check



In the top area, you can find the list of checked-for tools, grouped by use cases. Each use case consists of various tools. Tools may be necessary for more than one group. A use case can be run as long as all tools in it are available ("Ok").

UseCase and their tools (may be subject to changes)

- **ProjectFiles**: Python.
- **Installer**: Python, NSIS (Windows), DependencyWalker (Windows).
- **Documentation**: Python, GraphViz, Java, Doxygen.
- **Master Builder**: All of the above, as the Master Builder will build all file types supported by ToolRunner.

Status

- **Ok**: Tool found and ready.
- **Missing Tools**: (for use case): One or more tools of this use case are missing.
- **Not in Path**: (for tools): Tool not found, either because it is not installed or because it is not found in the expected path (e.g., Windows system-wide paths). See also Chapter 5, *Troubleshooting*.

Description: Displays a brief description of what the specific tool is required for.

Typical User: Displays the typical user for this use case.

In the bottom area, you can find the overall results (including the required and found versions) and the URL and download URL to the software to help you install all necessary tools.

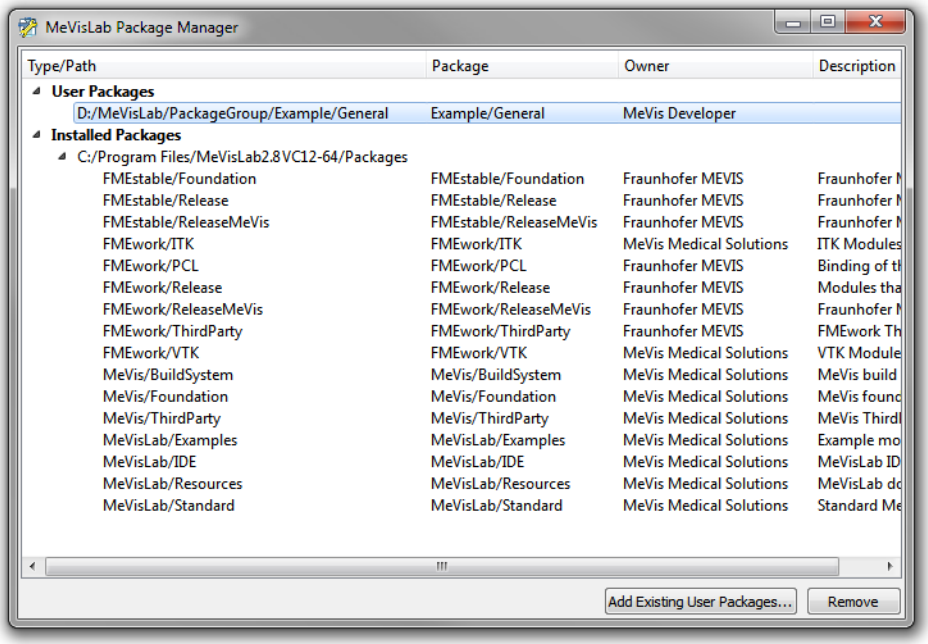
3.3.2. Open Package Manager...

The Package Manager gives an overview over the available packages. They are separated into

- **User Packages**: packages found in the user path.
- **mevislab.prefs**: packages resulting from the paths given in the prefs file.
- **Installed Packages**: packages resulting from an installation of, e.g., MeVisLab SDK.

The Package Manager does not handle the physical files; it manages only the search path for each package. Therefore, removing a package will only remove the package search path from the list.

Figure 3.7. Package Manager



To add your own packages, click **Add User Packages**. The standard file browser opens. Select and add folders or packages.

Folders are read recursively and all packages below them are automatically included. If you remove specific packages from a folder, you will be asked if you want to remove the complete folder from the list.

If a package with the same *PackageIdentifier* is found more than once, the last package found will overwrite the earlier packages. See the Package Structure documentation for details. These will be grayed out and marked with "Overwritten".

User Packages can be removed from the list. For this, select the user package and click **Remove**. You will be asked if you want to remove the package search path for the selected package. You can always re-add user packages.

3.3.3. Remove Ignored Files (Master Builder)

Removes the files from the file list that are explicitly designated to be ignored for a build, typically listed per package under `Configuration/Masterbuilder/MLAB_[PackageGroup_Package]/ignoredProfiles.txt`.

Chapter 4. Command Line Options

The ToolRunner can be used on the command line as well. It offers the following command line options:

```
    ToolRunner [-norun] [--exit-after-run] [-toolcheck] [-scanpath path] [-prefs prefsfile] [-i
```

- *-norun*: By default, the ToolRunner will run on all files that are passed in. To prevent this, use this option.
- *--exit-after-run*: Closes the ToolRunner after all files are processed.
- *-toolcheck*: Runs the tool check and show the result.
- *-scanpath*: Sets a scan path to be scanned for packages. Multiple -scanpath options can provided.
- *-prefs*: Specifies a prefs file to load, which is used for the package configuration.
- *-ignoreprefs*: Specifies to ignore the default MeVisLab prefs file.
- *file.{mlinstall|pro|mldoc}*: Specifies a file that should be processed.
- *directory*: Specifies a directory that should be processed. The directory will be scanned for known file types and these will be added to the ToolRunners file list.

Chapter 5. Troubleshooting

5.1. Windows

Q: A tool was installed but is still not recognized by the Tools Check utility, resulting in a "Not In Path" message.

A: Add the path of the program to the PATH variable for Windows in **System** → **System settings** → **Extended** → **Environment Variables**.