What's New in MeVisLab 2.0?

What's New in MeVisLab 2.0?

Table of Contents

1. What changed in MeVisLab 2.0? 5
    1.1. Overview of Changes between MeVisLab 1.6 and MeVisLab 2.0 5
    1.2. Packages 5
    1.3. ModulePath / $(MODULES) 6
    1.4. Module distrib 6
    1.5. New MLAB_ROOT environment variable 7
    1.6. New MLAB_COMPILER_VERSION environment variable 7
    1.7. DLL/Library placement 7
    1.8. qmake build system 7

List of Tables

1.1. 5

Chapter 1. What changed in MeVisLab 2.0?

In the following chapter, you can find information about the changes and how to adapt your existing projects to the new package structure.

For switching your user modules to the MeVisLab 2.0 structure, you need to

- adapt the package structure in your folder system.
- define a `Package.def` file for your package.
- substitute the old ModulePath by the new package-based $(Modules) variable.

1.1. Overview of Changes between MeVisLab 1.6 and MeVisLab 2.0

Table 1.1.

MeVisLab 1.6	MeVisLab 2.0
MEVIS_ROOT	MLAB_ROOT
MLABModules	Package/Modules
ModulePath	PackagePaths
UserModulePath	UserPackagePath
std/mevisPackages.pro	Package/PackageGroup_Package.pri
SDK very different to internal version	SDK and internal version mostly identical
Specialized MeVis Wizards	Unified Wizards that work on Packages
Different ADK and internal installers	Unified installer creation
Difficult to extend by SDK users	Easy to extend with user packages
Difficult to split into separate source/modules sections	Easy to split and move projects across Packages

1.2. Packages

The package structure is described in the Package Structure Documentation.

To move your old user modules to the new structure, you need to build a new PackageGroup/Package structure and add your files to it. You also need to create new `Package.def` files.

Tip

The easiest way of moving and converting your modules to the new structure is using the Package Wizard (**Project Wizard** → **New Package**). Enter the information for the new package, select the option **Import MeVisLab 1.6 UserprojectPath into this Package** and browse to your UserProjectPath to get your modules imported into the new package on the fly.

1.3. ModulePath / $(MODULES)

The ModulePath is no longer used. It is completely replaced by packages. This also means that related variables are gone: `$(MODULES), $(ModulePath), $(ModulePathURL)`.

As of MeVisLab 2.0, these variables are substituted by the unique PackageIdentifiers. The convention is:

`$(MLAB_PackageGroup_PackageName)`

This way the packages can be located no matter if they are installed with MeVisLab or in a SVN checkout. The MDL automatically defines package variables for each loaded package. These variables are available in all MDL `.def` and `.script` files, regardless in which order the modules of packages are parsed.

Example: The old code

```
Commands
   {
    source = "$(LOCAL)/ImportMasks.js"
    source = "$(MODULES)/std/scripts/xmlClass.js"
    source = "$(MODULES)/std/scripts/FormatFunctions.js"

    initCommand = initApplication
   }
```

...becomes...

```
Commands
  {
   source = "$(LOCAL)/ImportMasks.js"
   source = "$(MLAB_MeVisLab_Standard)/Modules/Scripts/xmlClass.js"
   source = "$(MLAB_MeVisLab_Standard)/Modules/Scripts/FormatFunctions.js"

   initCommand = initApplication
  }
```

Note

The variable points to the root of the package, which is one level higher than the Modules subdirectory (which contains the `.def` and `.script` files that MeVisLab scans).

1.4. Module distrib

In the past, modules used to have a distrib flag, which was `std/mevis/other`.

As of MeVisLab 2.0, each module in MeVisLab is part of exactly one package. The package of a module can be read using

```
package = ctx.package()
package.packageIdentifier() "MeVisLab/Standard"
package.path()
```

may result in "C:/Programme/MeVisLab/Packages/MeVisLab/Standard"

package() returns an MLABPackage object (see MeVisLab Scripting API).

```
MLAB.moduleInfo("ImgLoad").package
```

may result in "MeVisLab/Standard"

"ModuleSearch" and "Search in Network" no longer support the distrib flag and search for package identifiers instead.

1.5. New MLAB_ROOT environment variable

In the past, MEVIS_LIB used to be required to point the build scripts to the SVN checkout or the MeVisLab SDK directory.

As of MeVisLab 2.0, this has been replaced by the new MLAB_ROOT variable, which typically points to the "Packages" directory of an installed MeVisLab SDK. The MLAB_ROOT variable is the only variable that MeVisLab and the build scripts need to find all scripts and packages. MEVISLAB_INSTALL_PATH is gone as well.

Usually you do not need to set the variable in most use cases (except for MeVisLab Core developers, who may want to point the variable to their repository checkout).

1.6. New MLAB_COMPILER_VERSION environment variable

(currently Windows only)

This variable is automatically set by the MeVisLab Installer (on Windows), depending on the compiler the installer was build with. Possible values are:

```
MLAB_COMPILER_VERSION = VC8
MLAB_COMPILER_VERSION = VC8-64
MLAB_COMPILER_VERSION = VC9
MLAB_COMPILER_VERSION = VC9-64
```

The previous variable MEVIS_VCVER is no longer used. VC6 and VC7 support has been discontinued.

1.7. DLL/Library placement

In the past, DLLs were placed under the "Modules" directory and needed to be copied.

As of MeVisLab 2.0 with the new package concept, each Package contains a "lib" and "bin" directory. All `.pro` profiles inside of the sources have these two directories as targets for shared libraries, static libraries and executables.

```
PackageXYZ/Sources/...
PackageXYZ/lib/*.dll,*.dylib,*.so,*.pdb,*.lib,*.a,*.framework
PackageXYZ/bin/*[|.exe]
```

Each package is self-contained concerning the linking and dynamic loading of modules. MeVisLab does not need to update hundreds of DLLs after updating to newer DLLs anymore.

The DLL locking problem has been solved by placing DLLs temporarily into the "lib/updated" directory when they can not be written to the "lib" directory because MeVisLab is running. Upon restart, MeVisLab will move the dlls from the "lib/updated" directory to the "lib" directory.

1.8. qmake build system

In the past, the dependencies were included into the projects source files based on the `stdPackages.pro` and `mevisPackages.pro` files.

As of MeVisLab 2.0, these files are gone. With the new package structure, each package now has its own qmake configuration file that allows other to link to libraries/DLLs in that package.

The new way of linking to a package looks as follows:

```
SomeProfile.pro:
...
CONFIG += ML MLUtilities MLLUT
include ($(MLAB_MeVisLab_Standard)/Configuration/MeVisLab_Standard.pri )
include ( $(MLAB_MeVisLab_Foundation)/Configuration/MeVisLab_Foundation.pri )
...
```

The naming of the package configuration *.pri files is standardized using the syntax [PackageGroup]_[PackageName].pri.

The variables need to be set as environment variables, which is done by the scripts that run qmake. They automatically scan the available packages and set all environment variables before calling qmake.