Package Structure in MeVisLab - Documentation

Package Structure in MeVisLab - Documentation

Table of Contents

List of Figures

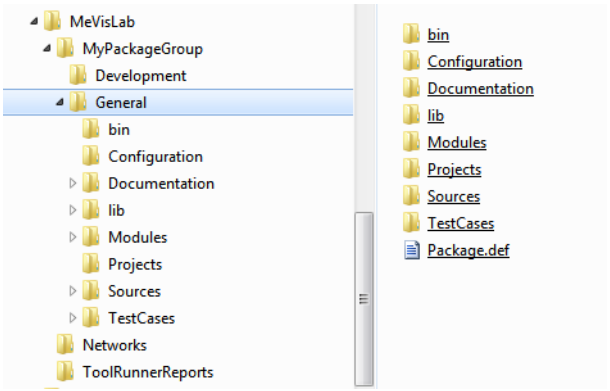
Chapter 1. Package Structure in MeVisLab

1.1. What is a Package?

A self-contained directory structure that consists of the following components:

- `PackageGroup`
  - `PackageName`
    - `Package.def`
    - `bin`
    - `Configuration`
    - `Documentation`
    - `lib`
    - `Modules`
    - `Projects`
    - `site-packages`
    - `Sources`
    - `TestCases`

Figure 1.1. Example for a Package Tree



In this example, we have a *PackageGroup* named `MyPackageGroup`. Below it, two packages can be found: `General` and `Development`. Below each package, the typical folders can be found as shown for the `General` package. This example was generated with the Project Wizard in MeVisLab.

The folders `Networks` and `ToolRunnerReports` in the figure above are on the same level as the *PackageGroup*.

A *PackageGroup* can contain any number of packages, and of course there can be different *PackageGroups*.

The *PackageIdentifier* is defined by `<PackageGroupName>/<PackageName>`, e.g., the MeVisLab Standard package has the identifier `MeVisLab/Standard`.

1.2. How Does MeVisLab Find Packages?

MeVisLab searches in

- the packages directory in which MeVisLab was installed.
- the directories given in the `PackagePaths` settings of `mevislab.prefs`.
- the *UserPackage* path (as set in the MeVisLab Preferences dialog.

Scanning is always two levels deep, never deeper. If a package with the same *PackageIdentifier* is found more than once, the last package found will overwrite the earlier packages (in the order given above). This way, your packages given by `mevislab.prefs` file or your *UserPackages* can overwrite installed packages.

Note

You can check your effective package structure with the tool ToolRunner.

To compile a running MeVisLab, the packages `ThirdParty`, `Foundation`, `IDE`, and `Standard` are required. All other packages are optional and not required for a fully working MeVisLab installation (except that you will not have all the nice modules from the other packages).

1.3. Who "Knows" About Packages?

Packages are supported in the complete tool chain

- `CMake` knows about the packages using the MeVisLab PackageScanner. It especially extends the CMake modules path to the `cmake` directory found in any package.
- associated tools like ToolRunner know about packages.
- a MeVisLab module, accessible with `ctx` in scripting, knows its package: `ctx.package().packageIdentifier()`.
- MDL knows about packages using `MLAB_PackageGroup_PackageName` variables.
- MLABPackageManager provides package information to Python scripting.
- wizards use packages as their target.
- installers know about packages (`SWITCH_PACKAGE`, etc.).

Chapter 2. Package Components

2.1. The `mevislab.prefs` File

```
Settings {
  PackagePaths {
    pathRoot = MY_CHECKOUT_PATH

    path = FMEstable/Foundation
    path = FMEstable/General
    path = FMEstable/Release

    path = FMEwork/General

    //path = MeVisLab/Standard
    //path = MeVisLab/Foundation
  }
  ...
}
```

If you only need certain packages, enable/disable individual packages here by (un)commenting them. `MY_CHECKOUT_PATH` has to be set to the root of the repository checkout (where the `MeVisLab` and `FMEwork`/`FMEstable` directories are located).

2.2. The `Package.def` File

The file `Package.def` is part of every package. It defines the position of the package in the *PackageGroup*.

A typical example (excerpt from the `MeVisLab/Standard/Package.def`):

```
Package {
  packageGroup = MeVisLab
  packageName  = Standard

  owner        = "MeVis Medical Solutions"
  description  = "Standard MeVisLab Modules"
}
```

Note

In principle, the package structure is defined by `Package.def`. However, the implementation of the package handling expects that all packages are below their *PackageGroup*.

2.3. Modules

The `Modules` directory of a package contains all files that MeVisLab needs to know at runtime (`*.def`, `*.script`, `*.mlab`, `*.py`, etc.).

Shared Libraries (DLLs) are stored in a `lib` directory.

A typical sub-structure is

- `ML`
- `Inventor`
- `Macros`

- Applications
- Resources
- Scripts
- Shared
- Wrappers

In the Scripts directory, there is typically a python directory where you can store shared Python code. The import statements of a module's Python code searches in this directory for the Python module to import.

You can also set additional Python import paths in your module's Commands-section in the .script file.

```
Commands {
  importPath = $(LOCAL)/Python/
  source = $(LOCAL)/MyModule.py
}
```

This sets an additional import path to a Python directory that is located relative to the module's definition.

You can have multiple importPath statements in the Commands-section.

2.4. Sources

The Sources directory of a package contains all source files that are used to build the shared libraries or executables.

CMakeLists.txt files are used to specify DLL / executable projects.

Dependencies to other packages or projects are given in the MLAB_PACKAGE and CONFIG variables of the profile.

A typical sub-structure is

- ML
- Inventor
- Shared
- Wrappers

2.5. TestCases

The TestCases directory contains the files for automatic tests that are executed with the TestCaseManager. Please see the documentation for the TestCenter for how to define tests.

Usually there is a FunctionalTests directory in this directory, which in turn contains directories that match the directories from the Modules directory. This is purely optional, though.

2.6. Projects

In the Projects directory, you can store self-contained projects for an easy moving of projects. MeVisLab searches in this directory for projects in a depth of two, so there can be top-level directories containing a number of actual projects directories, and/or just the actual projects directories.

Each projects directory contains a Modules, and optional Sources and TestCases directories (similar to the top-level directory structure of a MeVisLab package). The structure of the Modules directory is

similar to a directory of the top-level `Modules` directory, i.e., it can contain `mhelp`, `networks`, and `Scripts` directories. It also contains the `.def`, `.script`, `.mlab`, and `.py` files of the module(s) that are defined in a project.

Your project directory can contain the sub-directory `Modules/Scripts/python`, but to import Python modules from this directory, you have to use a MeVisLab-specific virtual package: If you, e.g., want to import the file `Projects/MyProject/Modules/Scripts/python/MyPythonModule.py` in your Python code, you have to use the import statement

```
import mlab_projects.MyProject.MyPythonModule
```

i.e., you must prefix your import with `mlab_projects.<project-directory-name>`. For convenience you probably would rather use

```
import mlab_projects.MyProject.MyPythonModule as MyPythonModule
```

This also allows to import Python modules/packages from other projects.

Note

Nowadays the use of the `Projects` directory is recommended over the old directory structure where the files for a certain module were scattered over the top-level `Modules`, `Sources`, and `TestCases` directories.

2.7. cmake

The `cmake` directory can contain `<PackageGroup>_Settings.cmake` and `<PackageGroup>_<PackageName>_Settings.cmake` files that define (compiler) settings for C++ projects of the given project.

Note

These settings files don't need to reside in the package that they are intended for. You just need to make sure that the package where they reside is always available when the package for which they are applied is used.

You can also put files like `<ProjectName>Config.cmake` here, which are needed for `findPackage()` calls in CMake files.

2.8. Configuration/Installer

The `Configuration/Installer` directory contains installer definition files (`*.mlinstall` and `*.mli` files):

2.9. Documentation

The `Documentation` directory contains all package documentation, except for the individual module documentation, which is part of the `Modules` folder. The documentation can be either in Doxygen or DocBook format.

- `/Documentation/Sources` contains the sources for building documentation.
- `/Documentation/Publish` contains the result documentation (and is NOT checked into the repository).
- `/Documentation/Index` allows to configure additional entries on the *MeVisLab Help Page* dynamically.

- the `*.mldoc` file format facilitates configuring and building of Doxygen and DocBook documents.

2.10. Lib/Bin

The `lib` and `bin` directory of a package contain the shared libraries and executables.

- `lib/` contains all shared libraries and static library files of the package.
- `bin/` contains all executables.

Profiles in sources are set up to copy result files to `lib`/`bin`.

If a DLL cannot be overwritten, it is copied to the `lib/updated` subdirectory and is moved to `lib/` on the next MeVisLab startup. This way you can compile your project while MeVisLab is still running (which would otherwise fail).

2.11. site-packages

The `site-packages` directory contains external Python packages that were installed with the PythonPip module into a MeVisLab package. This directory – if it exists – is automatically added to the PYTHONPATH.