Packaging MITK

## What (not) to package?

* Modules
  + Standard modules/libraries, nothing fancy
    - Dependencies to other modules
    - Dependencies to external projects
  + Command line apps
    - Executables that a logically part of a module
    - **NOT** test drivers, though
    - Classification apps call MITK\_INSTALL\_TARGETS() manually
  + Auto-load modules
    - Loaded during run-time, no link-time dependencies, so basically “plugins”
    - Located in a subfolder named after the module that is supposed to load it
    - Can be queried by target properties (both dependers and dependees)
  + MatchPointRegistration also has algorithm plugins
  + IGT has STL files
* Plugins
  + Dependencies to modules (including auto-load modules)
  + Dependencies to external projects
  + No dependencies to other plugins?
  + DICOM plugins needs storescp helper app
  + XNAT plugin installs SSL
* Applications
  + BlueBerry applications, not modules executables
  + Provisioning and ini files list plugins to load at run-time
* External projects
  + CppMicroServices is disguised as module but is external project
  + Most external projects should be collected by BundleUtilities
  + There are also macOS frameworks like Python or Qt
    - Resource executables are not resolved by BundleUtilities
  + Qt is complex and comes with own deploy helper apps
    - Found blog article on how to integrate these helper apps in packaging
  + CTK has subprojects like PythonQt
  + Some external projects are not pre-installed to ep folder
  + MatchPoint has plugins

## When to package?

* Package script should be assembled only late at configure time when all other dependencies are set up (otherwise plugin dependencies may not be completely established)
* Probably work with GLOBAL and TARGET properties that can be queried after everything is set up

## How to package?

* Currently restricted to…
  + Windows: ZIP archive, NSIS 2 installer
  + Linux: Tarball
  + macOS: dmg
* Choose modern alternatives if they exist
* Windows and Linux package “everything” into a single package
* macOS packages each BlueBerry application separately or everything else if no application is enabled

## And then there is…

* PluginGenerator
* ProjectTemplate
* Using MITK as toolkit

## Where is the current install/package code?

* CMake/mitkInstallRules.cmake
  + mitk.ico and mitk.bmp
  + New: Auto-load dependencies of executables
  + Executables
  + PythonQt
  + Qt
    - SQL drivers
    - Image formats
    - Icon engines
    - Platform plugins
    - Styles
    - QtWebEngine including locales
  + MatchPoint utilities and algorithms
* MITK\_INSTALL\_TARGETS(EXECUTABLES)
  + CMake/mitkFunctionCreateBlueBerryApplication.cmake
  + Modules/Classification/CLMiniApps/CMakeLists.txt
* MITK\_INSTALL(TARGETS)
  + CMake/mitkFunctionCreateMatchPointDeployedAlgorithm.cmake
* MITK\_INSTALL(FILES)
  + CMake/mitkFunctionInstallProvisioningFiles.cmake
  + CMake/MITKIGTHardware.cmake (SDKs)
  + CMake/mitkToFHardwareInstallRules.cmake (SDKs)
  + Modules/IGT/CMakeLists.txt (STL files)
  + Plugins/org.mitk.gui.qt.xnat/CMakeLists.txt (SSL)
* MITK\_INSTALL\_HELPER\_APP(EXECUTABLES)
  + Plugins/org.mit.gui.qt.dicom/CMakeLists.txt (storescp)

# Install functions and macros

## mitkFunctionInstallAutoloadModules(<destination> <plugins…>)

Called in mitkFunctionCreateBlueBerryApplication() via BlueBerryApplicationInstallHook(). Uses the MITK\_AUTOLOAD\_TARGETS target property of the passed plugins to install auto-load dependencies in the corresponding subfolder.

## mitkFunctionInstallCTKPlugin([plugins])

Called in mitkFunctionInstallThirdPartyCTKPlugins(). See below.

## mitkFunctionInstallThirdPartyCTKPlugins([plugins…])

Install given plugins (or all if no plugins given) into all macOS bundles (MACOSX\_BUNDLE\_NAMES) or bin/plugins in Linux and Windows. Includes ctkMacroGetAllNonProjectTargetLibraries(). Actual installation is done in mitkFunctionInstallCTKPlugin().

## mitkFunctionInstallExternalCMakeProject(<ep\_name>)

Not used for packaging. Invokes <BINARY\_DIR>/cmake\_install.cmake of an external project during installation (via install(SCRIPT)).

## mitkFunctionInstallProvisioningFiles(<files...>)

Called in mitkFunctionCreateBlueBerryApplication(). For each provisioning file (filled with mitkFunctionCreateProvisioningFile()) call MITK\_INSTALL() for the .provisioning.install variant and remove the .install extension at the destination. The paths in the .provisioning.install file are relative to the BlueBerry application executable (@EXECUTABLE\_DIR/plugins/…).

## mitkMacroInstall(…)

Wrapper for install() that determines DESTINATION. On macOS call install() for each bundle (MACOSX\_BUNDLE\_NAMES) to install into <bundle>.app/Contents/MacOS. Install into bin on other platforms. A subfolder can be set with set(\_install\_DESTINATION …).

## mitkMacroInstallHelperApp(<targets>)

Basically a wrapper around install(PROGRAMS) and install(CODE) to call file(RPATH\_REMOVE) on Linux for each helper app. Also, \_fixup\_target() is called for each target and a qt.conf file is written just like in mitkMacroInstallTargets() (see below).

## mitkMacroInstallTargets([executables])

For each executable call install(TARGETS … [BUNDLE|RUNTIME]) for each bundle and install a qt.conf with install(CODE) that set the Prefix in the Paths section to either “.” or “./MacOS” on macOS. \_fixup\_target() is also called for each target.

# FixupBundle

## \_fixup\_target()

Defined in CMake/mitkMacroInstall.cmake.