Search

Qt 5 and Qt 6 compatibility

The semantics of the CMake API in Qt 5 and Qt 6 are largely compatible. However, up to Qt 5.14, all imported Qt library targets and commands contained the version number as part of the name. This makes writing CMake code that should work with both Qt 5 and Qt 6 somewhat cumbersome. Qt 5.15 therefore introduced *versionless* targets and commands to enable writing CMake code that is largely agnostic to the different Qt versions.

Versionless targets

In addition to the existing imported targets, Qt 5.15 introduced *versionless* targets. That is, to link against Qt Core one can both reference , or :`Qt6::Core`Qt::Core

```
find_package(Qt6 COMPONENTS Core)
if (NOT Qt6_FOUND)
    find_package(Qt5 5.15 REQUIRED COMPONENTS Core)
endif()

add_executable(helloworld
    ...
)

target_link_libraries(helloworld PRIVATE Qt::Core)
```

Above snippet first tries to find a Qt 6 installation. If that fails, it tries to find a Qt 5.15 package. Independent of whether Qt 6 or Qt 5 is used, we can use the imported target.`Qt::Core`

The versionless targets are defined by default. Set QT_NO_CREATE_VERSIONLESS_TARGETS before the first call to disable them.`find_package()`

> **Note:** The imported Qt::Core target will not feature the target properties that are available in the Qt6::Core target.

Versionless commands

Since Qt 5.15, the Qt modules also provide versionless variants of their commands. You can for instance now use qt_add_translation to compile translation files, independent of whether you use Qt 5 or Qt 6.

Mixing Qt 5 and Qt 6

There might be projects that need to load both Qt 5 and Qt 6 in one CMake context (though mixing Qt versions in one library or executable is not supported, so be careful there).

In such a setup the versionless targets and commands will be implicitly referring to the first Qt version that was found via . Set the QT_DEFAULT_MAJOR_VERSION CMake variable before the first call to make the version explicit.`find_package`find_package

Supporting older Qt 5 versions

If you need to support also Qt 5 versions older than Qt 5.15, you can do so by storing the current version in an CMake variable:

```
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core)

qt_add_executable(helloworld
    ...
)

target_link_libraries(helloworld PRIVATE Qt${QT_VERSION_MAJOR}::Core)
```

在这里，我们尝试找到第一个Qt 6，如果Qt 5失败，则以名称 。如果找到其中任何一个，则 将成功，并且 CMake 变量将定义为 或 。`find_package(<PackageName>...)`QT`find_package`QT_VERSION_MAJOR`5`6

然后，我们通过动态创建名称来再次加载已确定的Qt版本的包。这是必需的，因为期望包名称为 或 ，否则将打印错误。`Qt${QT_VERSION_MAJOR}`CMAKE_AUTOMOC`Qt5`Qt6

我们可以使用相同的模式来指定导入库的名称。在调用 之前，CMake 将解析为 或 。`target_link_libraries`Qt${QT_VERSION_MAJOR}::Widgets`Qt5::Widgets`Qt6::Widgets

建议的做法

尽可能使用 CMake 命令的无版本变体。

无版本导入的目标对于需要同时使用 Qt 5 和 Qt 6 进行编译的项目非常有用。由于缺少目标属性，因此不建议默认使用它们。

如果您需要支持早于 Qt 5.15 的 Qt 5 版本，或者如果无法控制 CMake 代码是在可能定义了 QT_NO_CREATE_VERSIONLESS_FUNCTIONS或QT_NO_CREATE_VERSIONLESS_TARGETS的上下文中加载的，请使用 CMake 命令和目标的版本化版本。在这种情况下，您仍可以通过变量确定实际命令或目标名称来简化代码。

视窗中的统一码支持

在 Qt 6 中，默认情况下，为链接到 Qt 模块的目标设置和编译器定义。这与 qmake 行为一致，但与 Qt 5 中的 CMake API 行为相比，这是一个变化。`UNICODE`_`UNICODE`

在目标上调用qt_disable_unicode_defines （） 以不设置定义。

```
add_executable(helloworld
    ...
)

qt_disable_unicode_defines(helloworld)
```

‹ 导入的目标 命令参考 ›