Sourcetree Users Guide

Table of Contents

Introduction

The *Sourcetree* utility creates a folder populated with all files and dependent projects necessary to compile a specific Qt project. The created folder can then be compressed and distributed as the source code for the Qt project. The generated source tree optionally includes a top-level project file and compilation instructions. Starting with version 2.0 of the *Sourcetree* utility the cmake *CMakeLists.txt* files are automatically generated from all Qt project files allowing makefiles to be created from *QMake* or *CMake* depending on user preferences.

Prior to writing this utility the process to create the source tree was done manually with mixed results, no instructions, and no top-level project file. One obvious example of a missed file was *info.plist* which is needed when compiled on MacOS systems. It was assumed the end user would have enough background with Qt to be able to figure out what steps were needed to compile a project but, particularly for projects with sub-dependent projects, this would require some trial and error. When using the *Sourcetree* utility with a top-level project file this kind of problem is eliminated.

Although binaries for macOS are no longer provided they can be created easily enough provided the necessary prerequisites are met for Qt. The primary development systems used by SCI are GNU/Linux and macOS with only testing done on Windows.

Overview

The *Sourcetree* utility is a single window sectioned into different functions. Illustration 1 shows the *Sourcetree* utility with the Qt project *MeasureDirect* loaded.

Creating a source tree for an existing Qt project involves the following steps:

1. Select the Qt project file from the Information section of the *Sourcetree* utility.
2. Click the *Load* button to read the Qt project file and all dependent files.
3. Change the Name and Version fields in the Information section as necessary. The name field is automatically set based on the name of the source project where the version field contains the last used entry.
4. Review the list of included source directories in the source section of the *Sourcetree* utility and uncheck anything that should be excluded from the final source tree.
5. Review the output options for target location, readme file, and the option for the creation of the top-level project.
6. Click the *Create* button to have the source tree created.

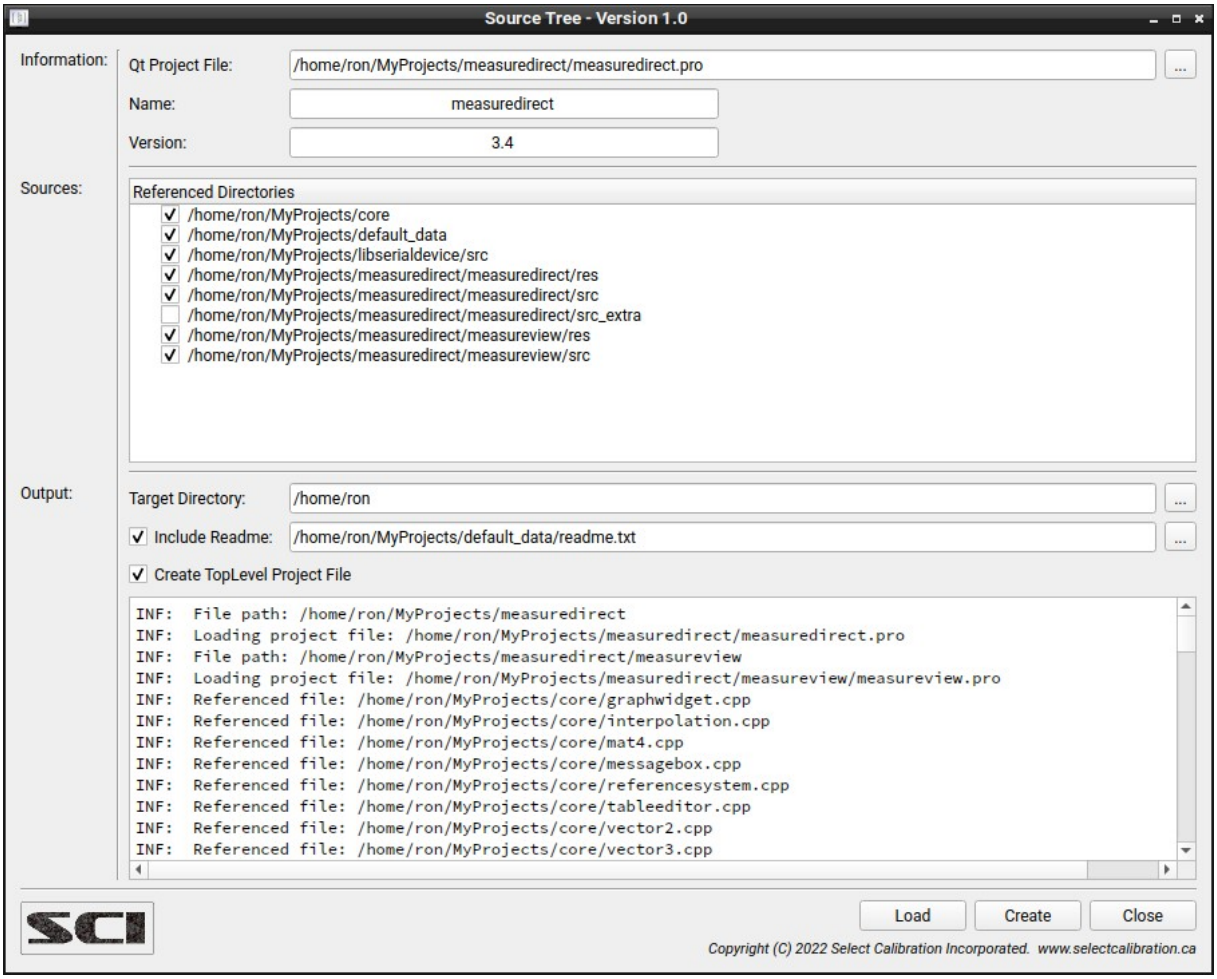

Illustration 1: Main window of the Sourcetree utility processing MeasureDirect version 3.4

Table 1: Options:

Section	*Option*	*Description*
Information	Qt Project File	Name and location of the existing Qt project file that will be used to generate the source tree.
	Name	Name associated to the project. When a project is loaded the name of the containing folder of the project file is used but this can be changed.
	Version	Version of the program. This value is static and must be set by the user.
Sources	Referenced Directories	List of directories that contain one or more required files. Any references to files in the specific directory can be ignored by unchecking the entry. See section Source Directory Exclusion for details.
Output	Target Directory	Location where the source tree folder will be created.

Section	*Option*	*Description*
	Include Readme	Option to include a readme file. The specified file will be copied into the top level of the source tree.
	Create Top-Level Project File	Option to create a top-level project file. If created the entire project and all dependencies can by created in one step.
	Load	Load the specified Qt project file.
	Create	Create the source tree.
	Close	Close the utility.

Source Directory Exclusion

Source directories containing one or more referenced files can be ignored when creating the source tree by unchecking the specific folder. Illustration 2 shows an example of the exclusion of referenced files for a specific project.

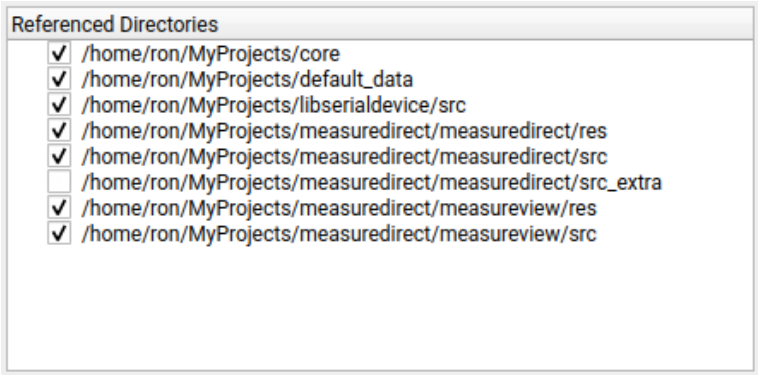

Illustration 2: List of directories from which referenced files will be added to the source tree.

Only directories can be excluded and not individual files. Depending on the type of file(s) contained in the excluded directory the source code may not compile.

One method to deal with problems of missing source files is by using define statements. From the example shown in illustration 2 where the sub folder src_*extra* is excluded the project file and source code has the following entries:

Project File:

```
DEFINES +=              INCLUDE_SRC_EXTRA
…
contains(DEFINES, INCLUDE_SRC_EXTRA) {
    SOURCES +=          src_extra/controller_dc_code.cpp
    HEADERS +=          src_extra/controller_dc_code.h
}
```

Source File:

```
#if defined  INCLUDE_SRC_EXTRA
```

```
#include "../src_extra/controller_dc_code.h"
#endif
```

Excluding directories doesn't make sense in most cases but there are, on occasion, reasons why it might be necessary.

Output

Using the *Sourcetree* utility project as an input the following shows the structure of the generated source tree:

```
sourcetree-2.0
├── CMakeLists.txt
├── core
│   ├── messagebox.cpp
│   └── messagebox.h
├── default_data
│   └── Info.plist
├── libcmakelists
│   ├── CMakeLists.txt
│   ├── libcmakelists.pro
│   └── src
│       ├── libcmakelists.cpp
│       ├── libcmakelists.h
│       ├── libcmakelistsimpl.cpp
│       └── libcmakelistsimpl.h
├── project.pro
├── readme.txt
└── sourcetree
    ├── CMakeLists.txt
    ├── res
    │   ├── icons.icns
    │   ├── logo.png
    │   ├── resource.rc
    │   ├── sourcetree.png
    │   ├── sourcetree.qrc
    │   └── winicon.ico
    ├── sourcetree.pro
    └── src
        ├── main.cpp
        ├── sourcetree.cpp
        └── sourcetree.h

8 directories, 23 files
```

The files included in the source tree contain operating specific references files such as *info.plist* (macOS) or *resource.rc* (Windows) regardless of the operating system the utility is run on. Files such as the Windows resource file are also scanned for dependent files and included in the final source tree.

Starting with version 2.0 of the *SourceTree* utility the *CMakeLists.txt* files are created for each each project and sub project. Use of cmake to generate the make files has been available in Qt for some time but with Qt version 6.0 and up the default is to use cmake instead of qmake.

The following is an example of the Qt project file from the library used to generate the CMakeLists.txt file and the generated output.

QMake Project File:

```
#
# libcmakelists project file
#
TEMPLATE =              lib
CONFIG +=               staticlib
CONFIG +=               release
CONFIG +=               c++11

# DEFINES    +=         DEBUG_DATA_PROCESSING

win32 {
CONFIG -=               staticlib
CONFIG +=               dll
}

DESTDIR =               lib

CONFIG(debug, debug|release):TARGET = cmakelistsd
else:TARGET =           cmakelists

QT +=                   core

QT -=                   gui


OBJECTS_DIR =           build/obj
MOC_DIR =               build/moc

HEADERS +=              src/libcmakelists.h \
                        src/libcmakelistsimpl.h


SOURCES +=              src/libcmakelists.cpp \
                        src/libcmakelistsimpl.cpp

DEFINES +=              CONFIGURE_DLL_EXPORT
```

CMake Project File:

```
cmake_minimum_required(VERSION 3.16)

project(cmakelists)

if(WIN32)

    add_definitions(-DUNICODE -D_UNICODE)

    if(MSVC)
        set(CMAKE_WINDOWS_EXPORT_ALL_SYMBOLS ON)
    endif()

endif()

if(DEFINED TARGET_BIN_LOCATION)
    set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${TARGET_BIN_LOCATION})
    set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${TARGET_BIN_LOCATION})
else()
    set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/bin)
    set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)
```

```
endif()

set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_AUTOMOC ON)

find_package(QT NAMES Qt6 Qt5)

message(STATUS "Using prefix Qt${QT_VERSION_MAJOR} for components...")

find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

set(sources src/libcmakelists.cpp
            src/libcmakelistsimpl.cpp)

set(headers src/libcmakelists.h
            src/libcmakelistsimpl.h)

if(WIN32)
    add_library(cmakelists SHARED ${sources} ${headers})
else()
    add_library(cmakelists STATIC ${sources} ${headers})
endif()

if(WIN32)

endif()

target_link_libraries(cmakelists PRIVATE Qt${QT_VERSION_MAJOR}::Core)
target_link_libraries(cmakelists PRIVATE Qt${QT_VERSION_MAJOR}::Widgets)

target_compile_definitions(cmakelists PRIVATE CONFIGURE_DLL_EXPORT)

set_target_properties(cmakelists PROPERTIES DEBUG_POSTFIX "d")

install(TARGETS cmakelists DESTINATION Libs)
```

The options available for Qt project files are vast and not all cases are covered by this utility. The differences between qmake and cmake project files are also just as vast so a suitable translator must be more than a simple translation from one syntax form to another.

The CMakeLists.txt file for any source project from SCI will compile without changes. Qt project files with more complex options may need to be manually edited. The potential to handle a variety of Qt project file states and modes exist and will be updated as needed.

Program Compilation Using QMake

Compiling the program from the source tree using the Qt project file can be done by running the following commands from the terminal window:

```
qmake
make
```

It is necessary to have a version of Qt installed and accessible from the command line in order to perform this step.

Compilation Example Using QMake

The following shows the *Sourcetree* utility compiled from the generated source tree on GNU/Linux using QMake. The source tree is created in the folder ~/sourcetree-2.0 in this example:

```
ron@linux-4o1p:~/MyProjects/sourcetree-2.0> qmake
Info: creating stash file /home/ron/MyProjects/sourcetree-2.0/.qmake.stash
ron@linux-4o1p:~/MyProjects/sourcetree-2.0> make
cd libcmakelists/ && ( test -e Makefile || /usr/lib/qt5/bin/qmake -o Makefile
/home/ron/MyProjects/sourcetree-2.0/libcmakelists/libcmakelists.pro ) && make -f Makefile
make[1]: Entering directory '/home/ron/MyProjects/sourcetree-2.0/libcmakelists'
g++ -c -pipe -O2 -fPIC -std=gnu++11 -Wall -W -D_REENTRANT -DCONFIGURE_DLL_EXPORT
-DQT_NO_DEBUG -DQT_CORE_LIB -I. -I/usr/lib/qt5/include -I/usr/lib/qt5/include/QtCore
-Ibuild/moc -I/usr/lib/qt5/mkspecs/linux-g++ -o build/obj/libcmakelists.o
src/libcmakelists.cpp
g++ -c -pipe -O2 -fPIC -std=gnu++11 -Wall -W -D_REENTRANT -DCONFIGURE_DLL_EXPORT
-DQT_NO_DEBUG -DQT_CORE_LIB -I. -I/usr/lib/qt5/include -I/usr/lib/qt5/include/QtCore
-Ibuild/moc -I/usr/lib/qt5/mkspecs/linux-g++ -o build/obj/libcmakelistsimpl.o
src/libcmakelistsimpl.cpp
rm -f lib/libcmakelists.a
ar cqs lib/libcmakelists.a build/obj/libcmakelists.o build/obj/libcmakelistsimpl.o
make[1]: Leaving directory '/home/ron/MyProjects/sourcetree-2.0/libcmakelists'
cd sourcetree/ && ( test -e Makefile || /usr/lib/qt5/bin/qmake -o Makefile
/home/ron/MyProjects/sourcetree-2.0/sourcetree/sourcetree.pro ) && make -f Makefile
make[1]: Entering directory '/home/ron/MyProjects/sourcetree-2.0/sourcetree'
…
<output removed for size>
…
g++ -c -pipe -O2 -std=gnu++11 -D_REENTRANT -Wall -W -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB
-DQT_GUI_LIB -DQT_CORE_LIB -I. -I../libcmakelists/src -I/usr/lib/qt5/include
-I/usr/lib/qt5/include/QtWidgets -I/usr/lib/qt5/include/QtGui
-I/usr/lib/qt5/include/QtCore -Ibuild/moc -isystem /usr/include/libdrm
-I/usr/lib/qt5/mkspecs/linux-g++ -o build/obj/moc_sourcetree.o
build/moc/moc_sourcetree.cpp
g++ -Wl,-O1 -Wl,-rpath,/usr/lib/qt5/lib -o bin/sourcetree build/obj/messagebox.o
build/obj/main.o build/obj/sourcetree.o build/obj/qrc_sourcetree.o
build/obj/moc_sourcetree.o   -L../libcmakelists/lib -lcmakelists -lpthread
/usr/lib/qt5/lib/libQt5Widgets.so /usr/lib/qt5/lib/libQt5Gui.so
/usr/lib/qt5/lib/libQt5Core.so /usr/lib64/libGL.so
make[1]: Leaving directory '/home/ron/MyProjects/sourcetree-2.0/sourcetree'
ron@linux-4o1p:~/MyProjects/sourcetree-2.0>
ron@linux-4o1p:~/MyProjects/sourcetree-2.0> sourcetree/bin/sourcetree   ← running program
ron@linux-4o1p:~/MyProjects/sourcetree-2.0>
```

Compilation Example Using CMake

The following shows the *Sourcetree* utility compiled from the generated source tree on GNU/Linux using CMake. The source tree is created in the folder ~/sourcetree-2.0 in this example:

```
ron@linux-4o1p:~/MyProjects/sourcetree-2.0> cmake .      ← generator is g++ on GNU/Linux
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc - works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
```

```
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ - works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using prefix Qt5 for components...
-- Using prefix Qt5 for components...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ron/MyProjects/sourcetree-2.0

ron@linux-4o1p:~/MyProjects/sourcetree-2.0> make
Scanning dependencies of target cmakelists_autogen
[  7%] Automatic MOC for target cmakelists
[  7%] Built target cmakelists_autogen
Scanning dependencies of target cmakelists
[ 15%] Building CXX object
libcmakelists/CMakeFiles/cmakelists.dir/cmakelists_autogen/mocs_compilation.cpp.o
[ 23%] Building CXX object libcmakelists/CMakeFiles/cmakelists.dir/src/libcmakelists.cpp.o
[ 30%] Building CXX object
libcmakelists/CMakeFiles/cmakelists.dir/src/libcmakelistsimpl.cpp.o
[ 38%] Linking CXX static library lib/libcmakelists.a
[ 38%] Built target cmakelists
Scanning dependencies of target sourcetree_autogen
[ 46%] Automatic MOC for target sourcetree
[ 46%] Built target sourcetree_autogen
[ 53%] Automatic RCC for res/sourcetree.qrc
Scanning dependencies of target sourcetree
[ 61%] Building CXX object
sourcetree/CMakeFiles/sourcetree.dir/sourcetree_autogen/mocs_compilation.cpp.o
[ 69%] Building CXX object sourcetree/CMakeFiles/sourcetree.dir/__/core/messagebox.cpp.o
[ 76%] Building CXX object sourcetree/CMakeFiles/sourcetree.dir/src/main.cpp.o
[ 84%] Building CXX object sourcetree/CMakeFiles/sourcetree.dir/src/sourcetree.cpp.o
[ 92%] Building CXX object
sourcetree/CMakeFiles/sourcetree.dir/sourcetree_autogen/PNK5WDWK6L/qrc_sourcetree.cpp.o
[100%] Linking CXX executable ../bin/sourcetree
[100%] Built target sourcetree
ron@linux-4o1p:~/MyProjects/sourcetree-2.0> bin/sourcetree              ← running program
ron@linux-4o1p:~/MyProjects/sourcetree-2.0>
```

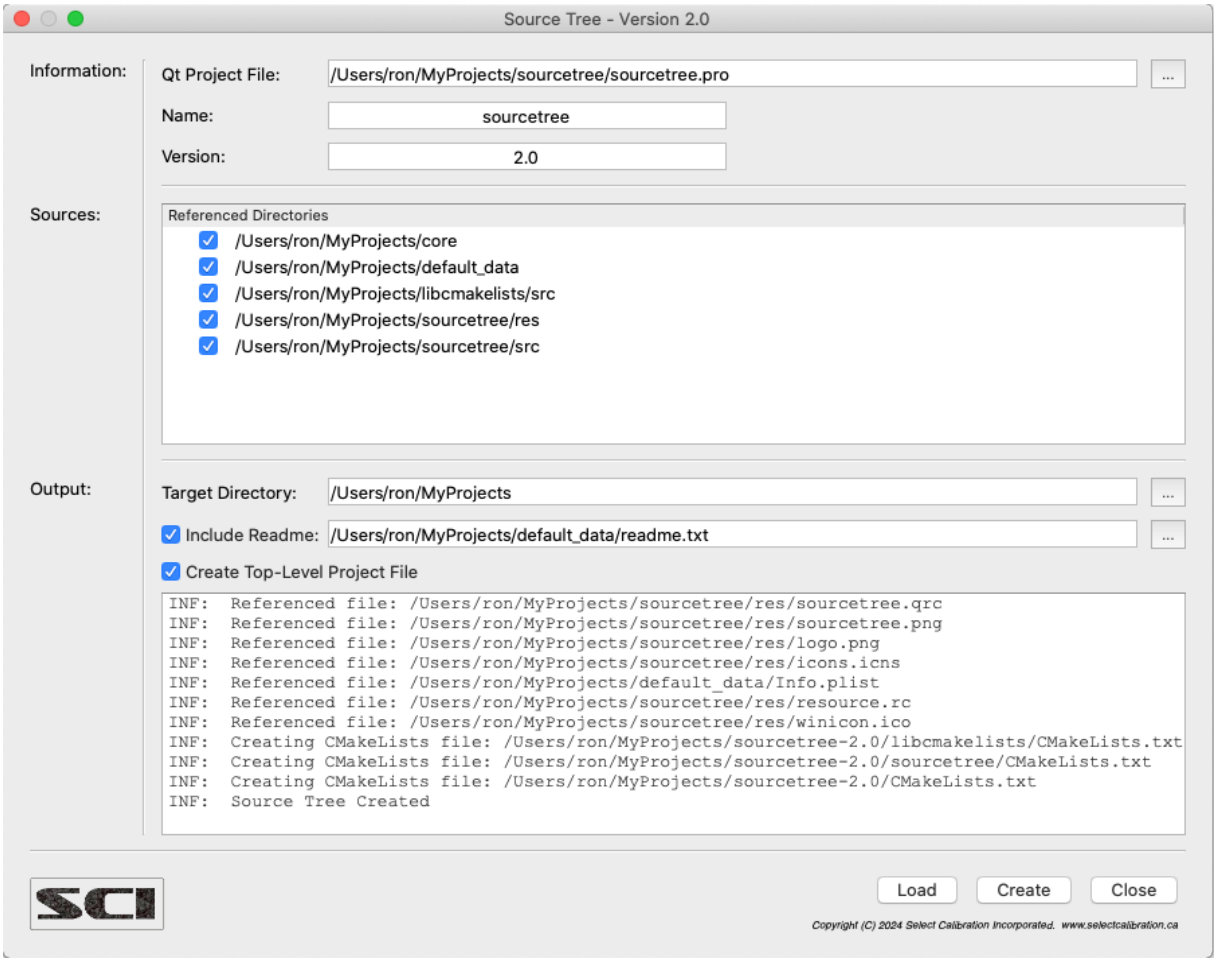

Illustration 3: Execution of the compiled CMake sourcetree utility from the command line on MacOS.

Revision History

Date	*Version*	*Changes*
July 25, 2022	1.0	New Program
Dec 25, 2023	1.1	[bugfix] Better handling of poorly formatted Qt project files.
Jan 6, 2025	2.0	[bugfix] Prevent recursive loading of the same project file Added option to generate CMakeList.txt file.
Jan 16, 2025	2.1	[bugfix] Issues with top level project file with mult-level sub projects. [bugfix] Var TARGET_BIN_LOCATION not conditionally set in subdir projects.
Jan 17, 2025	2.2	[bugfix] Blank WIN32 scope created in cmake file. Added processing scope for Qt CONFIG(...). Added processing scope for Qt CONTAINS(...).