```
# ekke's CMakeLists example for mobile Apps (Android, iOS)
# Host: macOS
#
# 2024-05-17
# 0.9
#
# Author: Ekkehard Gentz (ekke) - Independent
# @ekkescorner (X - aka Twitter)
# @ekke (Qt Discord Server)
# LinkedIn: http://linkedin.com/in/ekkehard/
# GitHub: https://github.com/ekke
# Qt6 blog: https://ekkesapps.wordpress.com/

# This CMakeLists.txt couldn't be done without:
# Qt Discord Server: https://discord.gg/DZaVKYhj
#     special thx: @PaulMasri-Stone @MikeWhite @Emeric
# Qt Blogs
# Qt Forums: https://forum.qt.io/category/11/mobile-and-embedded
# and more...

# ##################################################################
# ATTENTION: ekke's CMakeLists.txt is specialized for mobile apps   #
# and projects with nested subfolders (QML, Resources)              #
# For non-mobile QML apps see 'Building a QML Application':         #
# https://doc.qt.io/qt-6/cmake-build-qml-application.html           #
####################################################################

#
# ##################################################################
# ## DISCLAIMER:                                                   #
# ##           W I P - All is work-in-progress.                    #
# ##  Not all variations in ekke's CMakeLists have been tested yet. #
# ##  stay tuned...                                                 #
# ##################################################################
#
# ##################################################################
# ##  ekke's CMakeLists.txt
# ##  Discussions, Downloads and more at Github Project:            #
# ##  https://github.com/ekke/ekkesQtCMakeLists                     #
# ##################################################################
#
# ##################################################################
# ##  ekke's blog about CMake... starts here:                       #
# ##  https://t1p.de/ekkeCMakeMobileApps                            #
# ##################################################################
#
# Background:
# ekke ported 20 mobile business apps (Android, iOS) from 5.15 to 6.6+
# see ekke's CheckList: https://t1p.de/ekkeChecklist
# ekke supports up to 20 mobile business apps, but doesn't want
# to support 20 different CMakeLists.txt ;-)
# While moving from QMake to CMake this CMakeLists.txt was developed
# and will be used to build all of ekke's apps with CMake.
#
# Goal:
# To get a single CMakeLists.txt valid for all of ekke's mobile apps,
# mastering all the complexitiy of Android and iOS to deploy,
# run on device and publish to the App Stores.
# Optimized for ekke's project structure / rules,
# also valid for test- or demo- apps with simple plain structure.
# Well documented, so easy to customize for your needs (hopefully)
# https://t1p.de/ekkeCMakeLists
#
# Common features used by ekke's apps:
#    QuickControls2 (Material style)
```

```
#    Network (QNAM, QNetworkInfo)
#    Bluetooth LE (Barcode Scanner, Printer, BedMotors, WaiterLock, ...)
#    Multimedia (QML Camera)
#    Native FileDialog
#    Sharing Text, Images...
# 3rdParty used:
#    OpenSSL (Android)
#    MobileUI (colorize Statusbar)
#
# Read all the details "HowTo Move from QMake to CMake":
# https://t1p.de/ekkeCMakeMobileApps
#
# This CMakeLists.txt example can be tested with:
#    (will be published soon):
#    https://github.com/ekke/c2gQtWS_Qt6 (ekkes project structure)
#    https://github.com/ekke/cmake_test1 (simple project structure)
#
# Limitations: ekke's example apps are only using one qml module yet,
# which was executed immediately at app-start, so this CMakeLists won't be
# helpful for multi-module-projects, but it's planned for the future
# to modularize the apps. stay tuned...
#
# Next months ekke will provide some more example apps:
# https://t1p.de/ekkesAppsQt6
#
# First steps to modernize projects moved from QMake to CMake immediately:
#    Rename 'main.qml' into 'MyMain.qml' (should now start with uppercase)
#    Refactor Qml Engine 'load from URL' into 'load from Module'
#    Refactor Resources from 'qrc:/...' into 'qrc:/qt/qml/Ekke/Apps/Main/...'
#    Refactor all 'qmlRegisterType(...)' into 'QML_ELEMENT'
#    See ekke's Checklist: https://t1p.de/ekkeRefactorQM2CM
#    See also: https://t1p.de/ekkeQML_ELEMENT
#    Refactor all 'context->setContextProperty()' into 'QML_SINGLETON'
#    See also: https://t1p.de/ekkeQML_SINGLETON
#
# Next steps to modernize and modularize the apps:
#    Adjust UI for Material 3
#    Refactor QML Connections new syntax
#        function onFoo(<arguments>) { ... }
#    Refactor Unqualified Access
#    Then - finally - modularize the apps :)
#    See also: https://t1p.de/ekkeModernizeApps
#
# IMPORTANT:
# Tested on Android and iOS: build, deploy, publish to stores.
# On macOS only tested in QtCreator to debug / test UI without publishing.
#
# Not tested with Qt 6 Version < 6.6, recommended: Qt 6.7,
# final solution expected for 6.8 LTS, all is Work-In-Progress yet !
#
# Remark:
# ekke started with no knowledge about CMake,
# also dealing with build systems isn't ekke's domain.
# If there's something wrong or can be done better/easier,
# please let ekke know about
#
# ekke's CMakeLists.txt overview:
# Here is an overview of the structure of this CMakeLists.txt:
# https://t1p.de/ekkeCMakeListsOverview
#
# ekke's project structure (from 5.15):
#    c++ and qml files and resources are organized in subfolders.
#    This doesn't match Qt's recommended structure for qml modules.
#    per ex. problems with QT_QML_SINGLETON_TYPE
#    more infos: https://t1p.de/ekkesProjectStructure
```

```
#
# Hints:
# Correct sequence must be observed in CMakeLists.txt:
#     "project()" must be defined before checking "if(ANDROID)..." or "if(IOS)..."
#     "project()" must be defined before "find_package(...)"
#     "find_package(...)" must be defined before qt_standard_project_setup(..)
#     "qt_add_executable(${MY_APP}" must be defined before "qt_add_qml_module($
{MY_APP}"
#     "qt_add_executable(${MY_APP}" must be defined before
add_android_openssl_libraries
#         (see 3rdParty below)
# ------------------------------------------------------------

# ####  C M A K E   ####

# Hint: Qt for ios needs minimum 3.21.1
cmake_minimum_required(VERSION 3.24)

set(CMAKE_CXX_STANDARD_REQUIRED ON)
# CMAKE_AUTOMOC is set from qt_standard_project()
set(CMAKE_AUTORCC ON)

# #### V A R I A B L E S ####

# Hint: Variables in this example are set for QtWS Conference App.
# adjust to your needs
# ProjectInfo MS Word Formular or Checklist can help you:
# https://t1p.de/ekkeCMakeProjectInfo

# ----- SOME VARIABLES NEEDED BELOW -----

# The QT VERSION
set(MY_QT_VERSION "6.7")

# My APP TARGET
set(MY_APP "c2gQtWS_x")

# Debug or Release ?
# CMAKE_BUILD_TYPE empty on iOS.
# The Build on iOS is a multi-config-build,
# CMAKE_CONFIGURATION_TYPES is set.
# You can use generator expressions
# not used by ekke yet (ToDo) - found this example:
# target_compile_definitions(${MY_APP} PRIVATE
#     $<$<CONFIG:Release>:NDEBUG>
# )
message(STATUS "CMAKE BUILD TYPE: ${CMAKE_BUILD_TYPE}")
message(STATUS "CMAKE_VERSION: ${CMAKE_VERSION}")
message(STATUS "CMAKE_CONFIGURATION_TYPES: ${CMAKE_CONFIGURATION_TYPES}")

# ATM not used as Android Label - see below
# works only for iOS: ...CFBundleDisplayName
set(MY_DISPLAY_NAME "QtWS")

set(MY_COPYRIGHT "(c) 2024 Ekkehard Gentz, Rosenheim")

# UNIQUE IDs for this app ---
# ideally the unique IDs for Android and iOS should be the same
# but there are apps where both are different
set(MY_DOMAIN "org.ekkescorner")

# will be used per ex as Android Package Name
# set(MY_UNIQUE_ID_ANDROID "${MY_DOMAIN}.${MY_APP}")
set(MY_UNIQUE_ID_ANDROID "${MY_DOMAIN}.c2g.qtws")
```

```
# the ID used as XCODE_ATTRIBUTE_PRODUCT_BUNDLE_IDENTIFIER:
# Attention ! GUI Identifier only A-Z,a-z,- or .
# per ex. "cmake_test1" doesnt work, but "cmake-test1" will.
# set(MY_UNIQUE_ID_IOS "${MY_DOMAIN}.${MY_APP}")
set(MY_UNIQUE_ID_IOS "${MY_DOMAIN}.c2g.qtws")

# THE APP-VERSION
# On ANDROID used to calc QT_ANDROID_VERSION_NAME and ...CODE
# On IOS used for
#    ...CURRENT_PROJECT_VERSION (1.2.3)
#    ...MARKETING_VERSION (1.2)
# MY_MAJOR_VERSION.MY_MINOR_VERSION.MY_PATCH_VERSION
set(MY_APP_VERSION "2.3.0")

# calculate MY_MAJOR_VERSION, MY_MINOR_VERSION, MY_PATCH_VERSION from
MY_APP_VERSION
string(REGEX MATCH "^[0-9]+" MY_MAJOR_VERSION "${MY_APP_VERSION}")
string(REGEX MATCH "([0-9]+)$" MY_PATCH_VERSION "${MY_APP_VERSION}")
string(REGEX REPLACE "^[0-9]+\\.([0-9]+)\\.[0-9]+$" "\\1" MY_MINOR_VERSION "$
{MY_APP_VERSION}")
set(MY_MARKETING_VERSION ${MY_MAJOR_VERSION}.${MY_MINOR_VERSION})

# Project with ekke's project-structure (cpp/qml in subdirs) ?
# https://t1p.de/ekkesProjectStructure
set(EKKES_PROJECT_STRUCTURE ON)

# Adjust to your URIs
if(EKKES_PROJECT_STRUCTURE)
    set(MY_MAIN_MODULE_URI "Ekke.Apps.Main")
else()
    set(MY_MAIN_MODULE_URI "Example.Apps.Main")
endif()

# Used Features
# check languages at qt_standard_project_setup() !
# check location of translations at qt_add_translations() !
set(MY_APP_USES_TRANSLATIONS ON)
set(MY_APP_USES_NETWORK ON)
set(MY_APP_USES_BLUETOOTH OFF)
set(MY_APP_USES_MULTIMEDIA_CAMERA OFF)
# also iOS access to PhotoLibrary:
set(MY_APP_USES_NATIVE_FILE_DIALOG OFF)
set(MY_APP_USES_SHARING OFF)
set(MY_APP_USES_MOBILE_UI ON)

# ----- SOME   A N D R O I D   - ONLY - VARIABLES NEEDED BELOW  -----
# more info: https://t1p.de/ekkeCMakeAndroid

# ANDROID API-Level
#    Qt 6.6 supports MIN_API 26 (Android 8)
#    Qt 6.8 supports MIN_API 28 (Android 9)
#        https://bugreports.qt.io/browse/QTBUG-124890
#    Up to 6.6: Target API 33 (Android 13)
#    Qt 6.7+: Target API 34 (Android 14)
set(MY_MIN_API 26)
set(MY_TARGET_API 34)

# ----- SOME   I O S   - ONLY - VARIABLES NEEDED BELOW  -----
# more info: https://t1p.de/ekkeCMakeIOS

# INFO PLIST
# rename your existing Info.plist into QMake_Info.plist
# use ekke's CMake_Info.plist as base of your Info.plist to start with
# more infos: https://t1p.de/ekkeCMakeIOS
set(MY_INFO_PLIST "${CMAKE_CURRENT_SOURCE_DIR}/ios/Info.plist")
```

```
# DEVICE FAMILIES AND ORIENTATION
# ATM Setting Orientation using XCODE_ATTRIBUTEs doesn't work
# because the checkboxes at General Tab aren't sync'd.
# So for now: set allowed orientation manually in info.plist
# more info: myCMakeSnippts/cmake_ios_orientation.txt
set(MY_APP_USES_IPHONE ON)
set(MY_APP_USES_IPAD ON)

# USAGE DESCRIPTIONS
# Your App doesn't use Bluetooth-, Multimedia/Camera-
# or Native File Dialog- Features?
# The values will be ignored.
# Some more Usage Description Examples from ekke's apps:
# see myCMakeSnippets/cmake_ios_usage_descriptions.txt
# Hint: Usage Descriptions not localized for info.plist yet. ToDo
if(MY_APP_USES_BLUETOOTH)
    set(MY_BLE_PERIPHERAL_USAGE_DESCRIPTION
    "${MY_DISPLAY_NAME} searches Bluetooth LE Devices for Barcode Scanner"
    )
    set(MY_BLE_ALWAYS_USAGE_DESCRIPTION
    "${MY_DISPLAY_NAME} uses Bluetooth LE Barcode Scanner"
    )
endif()
if(MY_APP_USES_MULTIMEDIA_CAMERA)
    set(MY_CAMERA_USAGE_DESCRIPTION
    "${MY_DISPLAY_NAME} requires access to your Phone’s Camera."
    )
endif()
if(MY_APP_USES_NATIVE_FILE_DIALOG)
    set(MY_PHOTO_LIB_USAGE_DESCRIPTION
    "${MY_DISPLAY_NAME} uses Photos"
    )
endif()
if(MY_APP_USES_SHARING)
    # ToDo trying to set via XCODE_ATTRIBUTEs
    # ATM please insert CFBundleDocumentTypes manually into info.plist
    # see my_cmake_snippets/cmake_ios_sharing.txt
endif()

# BUNDLE_NAME, in most cases same as DisplayName
# BundleName recommended length <= 16
set(MY_BUNDLE_NAME ${MY_DISPLAY_NAME})

# EXECUTABLE_NAME - in most cases same as AppName
# in some older projects from ekke,
# the executable name can be different
set(MY_EXECUTABLE_NAME ${MY_APP})

# STORYBOARD and APP ICONS
# Hint: got problems with .xib file as LaunchScreen,
# better to use .storyboard.
# Check below "my_ios_resources" to match with your names and pathes
set(MY_STORYBOARD_NAME "MyLaunchScreen")

# The default value "AppIcon" will be used in most cases
set(MY_APP_ICON_NAME "AppIcon")

# APP CATEGORY
# select one of the categories:
# "business" "developer-tools" "education" "entertainment" "finance"
# "games" "graphics-design" "healthcare-fitness" "lifestyle" "medical"
# "music" "news" "photography" "productivity" "reference" "social-networking"
# "sports" "travel" "utilities" "video" "weather"
set(MY_APP_CATEGORY "developer-tools")
```

```
# ----- M E S S A G E S - VARIABLES
#
message(STATUS "EKKES_PROJECT_STRUCTURE: ${EKKES_PROJECT_STRUCTURE}")
message(STATUS "MY_MAIN_MODULE_URI: ${MY_MAIN_MODULE_URI}")
message(STATUS "MY_APP: ${MY_APP}")
message(STATUS "MY_BUNDLE_NAME: ${MY_BUNDLE_NAME}")
message(STATUS "MY_DISPLAY_NAME: ${MY_DISPLAY_NAME}")
message(STATUS "MY_EXECUTABLE_NAME: ${MY_EXECUTABLE_NAME}")

message(STATUS "MY_APP_VERSION: ${MY_APP_VERSION}")
message(STATUS "MY_MARKETING_VERSION: ${MY_MARKETING_VERSION}")

message(STATUS "USES TRANSLATIONS: ${MY_APP_USES_TRANSLATIONS}")
message(STATUS "USES NETWORK: ${MY_APP_USES_NETWORK}")
message(STATUS "USES BLUETOOTH: ${MY_APP_USES_BLUETOOTH}")
message(STATUS "USES CAMERA/MultiMedia: ${MY_APP_USES_MULTIMEDIA_CAMERA}")
message(STATUS "USES nativeFileDialog: ${MY_APP_USES_NATIVE_FILE_DIALOG}")
message(STATUS "USES SHARING: ${MY_APP_USES_SHARING}")
message(STATUS "USES MOBILE_UI: ${MY_APP_USES_MOBILE_UI}")

message(STATUS "MY_MIN_API: ${MY_MIN_API}")
message(STATUS "MY_TARGET_API: ${MY_TARGET_API}")

message(STATUS "MY_INFO_PLIST: ${MY_INFO_PLIST}")

message(STATUS "MY_DOMAIN: ${MY_DOMAIN}")
message(STATUS "MY_UNIQUE_ID_ANDROID: ${MY_UNIQUE_ID_ANDROID}")
message(STATUS "MY_UNIQUE_ID_IOS: ${MY_UNIQUE_ID_IOS}")

message(STATUS "MY_STORYBOARD_NAME: ${MY_STORYBOARD_NAME}")

message(STATUS "MY_APP_ICON_NAME: ${MY_APP_ICON_NAME}")

message(STATUS "MY_QT_VERSION: ${MY_QT_VERSION}")

message(STATUS "MY_COPYRIGHT: ${MY_COPYRIGHT}")

# #### P R O J E C T   P R O P E R T I E S ####

# THE PROJECT

# Hints:
#   OpenSSL uses ${PROJECT_NAME} as target
#   ${MY_APP} must match translator in main.cpp
#      or set TS_FILE_BASE in qt_add_translations()
project(${MY_APP} VERSION ${MY_APP_VERSION} LANGUAGES CXX)
message(STATUS "PROJECT_NAME: ${PROJECT_NAME}")

# PACKAGES - QT COMPONENTS
# all components needed for QuickControls2 apps:
find_package(Qt6 ${MY_QT_VERSION} REQUIRED COMPONENTS
    Core
    Gui
    Qml
    Quick
    QuickControls2
)

# components if network used
if(MY_APP_USES_NETWORK)
    find_package(Qt6 ${MY_QT_VERSION} REQUIRED COMPONENTS
        Network
    )
endif()
```

```
# components if Bluetooth used
if(MY_APP_USES_BLUETOOTH)
    find_package(Qt6 ${MY_QT_VERSION} REQUIRED COMPONENTS
        Bluetooth
    )
endif()

# components if Camera / Multimedia used
if(MY_APP_USES_MULTIMEDIA_CAMERA)
    find_package(Qt6 ${MY_QT_VERSION} REQUIRED COMPONENTS
        Multimedia
    )
endif()

# STANDARD PROJECT SETUP

# sets CMAKE_AUTOMOC and CMAKE_AUTOUIC as true
# sets I18N languages (Qt 6.7 required)
# ekke's apps need full translation of the source language,
# so 'en' is included in I18N_TRANSLATED_LANGUAGES
# If you only need plurals:
#     I18N_SOURCE_LANGUAGE en
#     I18N_TRANSLATED_LANGUAGES de fr
# https://doc.qt.io/qt-6/qtlinguist-cmake-qt-add-translations.html#plural-forms
if(MY_APP_USES_TRANSLATIONS)
    qt_standard_project_setup(
        REQUIRES ${MY_QT_VERSION}
        I18N_TRANSLATED_LANGUAGES de en fr
    )
else()
    qt_standard_project_setup(
        REQUIRES ${MY_QT_VERSION}
    )
endif()
message(STATUS "QT_I18N_SOURCE_LANGUAGE: ${QT_I18N_SOURCE_LANGUAGE}")
message(STATUS "QT_I18N_TRANSLATED_LANGUAGES: ${QT_I18N_TRANSLATED_LANGUAGES}")
# need the list of .ts filenames later to show in QtC Project View
# and to set_source_files_properties
# (ts filename per ex: translations/${MY_APP}_de.ts)
# so here's a little helper method to create MY_LANGUAGE_TS_FILES
if(MY_APP_USES_TRANSLATIONS)
    set (ALL_MY_LANGUAGES
        ${QT_I18N_SOURCE_LANGUAGE}
        ${QT_I18N_TRANSLATED_LANGUAGES}
    )
    # en can be duplicated (in source and translated language)
    list(REMOVE_DUPLICATES ALL_MY_LANGUAGES)
    set(MY_LANGUAGE_TS_FILES "")
    foreach(lang ${ALL_MY_LANGUAGES})
      list(APPEND MY_LANGUAGE_TS_FILES "translations/${MY_APP}_${lang}.ts")
    endforeach()
    message(STATUS "MY_LANGUAGE_TS_FILES: ${MY_LANGUAGE_TS_FILES}")
endif()

# INCLUDE DIRECTORIES FOR SOURCES
# ekke's projects are using subdirectories.
# Without include_directories(),
# adding QML_ELEMENT causes 'header not found'
# You can also use target_include_directories(${MY_APP}),
# in this case move the code down after qt_add_executable
if(EKKES_PROJECT_STRUCTURE)
    include_directories(
        cpp
        cpp/gen
```

```
    )
    if(ANDROID)
        include_directories(
            cpp/android
        )
    endif()
    if(IOS)
        include_directories(
            cpp/ios
        )
    endif()
    if(MY_APP_USES_BLUETOOTH)
        include_directories(
            cpp/bt
        )
    endif()
    if(MY_APP_USES_MULTIMEDIA_CAMERA)
        include_directories(
            cpp/photo
        )
    endif()
endif()

# #### M A I N   T A S K S ####

# PREPARE LISTS

# used by qt_add_qml_module(), qt_add_executable()

# In ekke's projects all these lists are generated by
# macOS scripts and executed by QtCreator External Tools

# Using "include(ekkesLists/my_xxx_files.cmake OPTIONAL)"
# prevents the CMakeLists.txt file from bloating,
# if many files belong to the project.
# those .cmake files contain per ex:
# set(my_qml_files qml/MyExamplePage.qml... )
# or will be ignored if empty or not exist because of OPTIONAL

# Lists are stored in folder "/ekkesLists"
# more info: https://t1p.de/ekkeCMakeExtTools

# QML Files - sets ${my_qml_files}
include(ekkesLists/my_qml_files.cmake OPTIONAL)

# RESOURCES (QMLDIR) Files - sets ${my_qmldir_files}
# W I P: TEMP solution to support qml in subfolders
# Probably supported with Qt 6.8
# qmldir simply imports URI from QML Module, per ex:
#     import Ekke.Apps.Main auto
include(ekkesLists/my_qmldir_files.cmake OPTIONAL)

# CPP Files - sets ${my_cpp_files}
include(ekkesLists/my_cpp_files.cmake OPTIONAL)
if(ANDROID)
    # Android CPP Files - appends ${my_cpp_files}
    include(ekkesLists/my_android_cpp_files.cmake OPTIONAL)
endif()
if(IOS)
    # iOS CPP Files - appends ${my_cpp_files}
    include(ekkesLists/my_ios_cpp_files.cmake OPTIONAL)
    # iOS ObjectiveC Files - appends ${my_cpp_files}
    include(ekkesLists/my_ios_objc_files.cmake OPTIONAL)
endif()
```

```
# RESOURCES (IMAGES) Files - sets ${my_image_files}
include(ekkesLists/my_image_files.cmake OPTIONAL)

# RESOURCES (DATA ASSETS) Files - sets ${my_data-assets_files}
include(ekkesLists/my_data-assets_files.cmake OPTIONAL)

# RESOURCES (JS Files) - sets ${my_js_files}
include (ekkesLists/my_js_files.cmake OPTIONAL)

# TRANSLATIONS
# see below

# ----- E X E C U T A B L E  +  M A I N   Q M L  M O D U L E

# more infos:
#     https://t1p.de/ekkesProjectStructure
#     Qt Blog https://www.qt.io/blog/implicit-imports-vs.-qml-modules-in-qt-6
#     https://bugreports.qt.io/browse/QTBUG-111763 QML in subdirectories
if(EKKES_PROJECT_STRUCTURE)
    # EXECUTABLE -----
    qt_add_executable(${MY_APP}
        cpp/main.cpp

        # Finalization not needed because CMake > 3.18
        # https://doc.qt.io/qt-6/qt-add-executable.html#finalization:
        # MANUAL_FINALIZATION and qt_finalize_project()
    )

    # MAIN MODULE -----
    # more info: https://t1p.de/ekkeCMakeQMLModule
    qt_add_qml_module(${MY_APP}
        # The executable is used as the backing target
        # QML Module will always be executed at app-start
        # recommended: engine.loadFromModule("Ekke/Apps/Main", "Main")
        # alternativ: engine.load(QUrl(QStringLiteral("qrc:/qml/Main.qml")));
        URI ${MY_MAIN_MODULE_URI}

        # Version is optional and should be omitted in most cases:
        # https://doc.qt.io/qt-6/qt-add-qml-module.html#versions
        # VERSION 1.0

        QML_FILES
            ${my_qml_files}
            ${my_js_files}

        SOURCES
            # ToDo ekke: move sources with no QML affinity to qt_add_executable()
            ${my_cpp_files}

            # Warning: Before using NO_RESOURCE_TARGET_PATH,
            # please read https://t1p.de/ekkeCMakeQMLModule

            ##RESOURCE_PREFIX /
            # better to use default RESOURCE_PREFIX 'qt/qml'
            # see also doc qt_add_qml_module and policy QP0001

            ##NO_RESOURCE_TARGET_PATH
            # Do NOT use NO_RESOURCE_TARGET_PATH
            # Only while testing before Resources are refactored
            # Enables you to easy test project with both: QMake and CMake

        RESOURCES
            ${my_image_files}
            ${my_data-assets_files}
            # ekke ToDo
```

```
            # https://bugreports.qt.io/browse/QTBUG-111763
            # (temp solution before 6.8)
            # W I P
            # ${my_qmldir_files}
    )
endif()

# plain project structure, qml files NOT in subdirectories
if(NOT EKKES_PROJECT_STRUCTURE)
    # modify to your requirements
    set(my_qml_files ${my_qml_files}
        Main.qml
    )
    # more files as needed
    qt_add_executable(${MY_APP}
        main.cpp
    )
    qt_add_qml_module(${MY_APP}
        URI ${MY_MAIN_MODULE_URI}

        QML_FILES
            ${my_qml_files}

        SOURCES
            ${my_cpp_files}

        RESOURCES
            ${my_image_files}
    )
endif()

#message(STATUS "my_qml_files: ${my_qml_files}")
#message(STATUS "my_qmldir_files: ${my_qmldir_files}")
#message(STATUS "my_js_files: ${my_js_files}")
#message(STATUS "my_cpp_files: ${my_cpp_files}")
#message(STATUS "my_image_files: ${my_image_files}")
#message(STATUS "my_data-assets_files: ${my_data-assets_files}")

# ----- O T H E R   F I L E S
# sometimes you need easy access to files from your project,
# but not listed in QtCreator Project View
# add_custom_target() allows Easy Access in QtCreator
# SOURCES NOT used for compiling, only to be accessible in IDE
# Hint: Always test if(EXISTS...), because this won't work
# if there are non-existing files contained !

# SHOW EKKES OTHER FILES
# Example ekkes_other_files (adjust to your needs):
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/.gitignore)
    set(ekkes_other_files ${ekkes_other_files}
       .gitignore
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/README.md)
    set(ekkes_other_files ${ekkes_other_files}
       README.md
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/AUTHOR.md)
    set(ekkes_other_files ${ekkes_other_files}
       AUTHOR.md
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/LICENSE)
    set(ekkes_other_files ${ekkes_other_files}
```

```
       LICENSE
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/gen-model/qt_ws.dtos)
    set(ekkes_other_files ${ekkes_other_files}
       gen-model/qt_ws.dtos
    )
endif()

if(IOS)
    # SHOW 'old' Info.plist from QMake IN PROJECT VIEW
    #     ekkes_other_files/Resources/QMake_Info.plist,
    # CMake Info.plist is found under
    #     CMake Modules/ios/Info.plist
    if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ios/QMake_Info.plist)
        set(ekkes_other_files ${ekkes_other_files}
            ios/QMake_Info.plist
        )
    endif()
endif()

if(IOS)
    # SHOW CMAKE SNIPPETS IN PROJECT VIEW
    # snippets helping to create project-specific CMakeLists.txt
    if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkes_cmake_snippets/
cmake_ios_usage_descriptions.txt)
        set(ekkes_other_files ${ekkes_other_files}
            ekkes_cmake_snippets/cmake_ios_usage_descriptions.txt
        )
    endif()
    if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkes_cmake_snippets/
cmake_ios_orientation.txt)
        set(ekkes_other_files ${ekkes_other_files}
            ekkes_cmake_snippets/cmake_ios_orientation.txt
        )
    endif()
    if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkes_cmake_snippets/
cmake_ios_sharing.txt)
        set(ekkes_other_files ${ekkes_other_files}
            ekkes_cmake_snippets/cmake_ios_sharing.txt
        )
    endif()
endif()

# SHOW EKKES LISTS IN PROJECT VIEW
# then it's easy to control the file names in generated lists
set(ekkesLists
    ekkesLists/my_qml_files.cmake
    ekkesLists/my_cpp_files.cmake
    ekkesLists/my_image_files.cmake
)
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkesLists/my_data-assets_files.cmake)
    set(ekkesLists ${ekkesLists}
        ekkesLists/my_data-assets_files.cmake
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkesLists/my_js_files.cmake)
    set(ekkesLists ${ekkesLists}
        ekkesLists/my_js_files.cmake
    )
endif()
if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/ekkesLists/my_qmldir_files.cmake)
    set(ekkesLists ${ekkesLists}
        ekkesLists/my_qmldir_files.cmake
    )
```

```
endif()

# SHOW .ts FILES IN PROJECT VIEW
# then it's easy to open / edit translations with Qt Linguist
# add ${MY_LANGUAGE_TS_FILES}, where we already have
# collected the languages.

# NOW SHOW ALL OTHER FILES IN QtC
add_custom_target(ekkes_other_files SOURCES
   ${ekkes_other_files}
   ${ekkesLists}
   ${MY_LANGUAGE_TS_FILES}
)

# ----- T R A N S L A T I O N   ---   Qt 6.7

# qt_add_translations on Qt 6.6 ? more details ?
# see https://t1p.de/ekkeCMakeTranslation
if(MY_APP_USES_TRANSLATIONS)
    find_package(Qt6 ${MY_QT_VERSION} REQUIRED COMPONENTS
        LinguistTools
    )
    # qm files will be available under /translations in build dir
    set_source_files_properties(
        ${MY_LANGUAGE_TS_FILES}
        PROPERTIES OUTPUT_LOCATION "${CMAKE_CURRENT_BINARY_DIR}/translations"
    )
    # ts files will be under /translations in source dir
    # always need complete translations of developer language
    # if you only need plurals, remove NO_GENERATE_PLURALS_TS_FILE
    # see also qt_standard_project_setup above
    qt_add_translations(${MY_APP}
        TS_FILE_DIR "translations"
        RESOURCE_PREFIX "/"
        NO_GENERATE_PLURALS_TS_FILE
    )
    # lrelease is running automatically
    # to run lupdate:
    #     In QtCreator type CMD-K cm and select update_translations
endif()

# ####  P L A T F O R M   S P E C I F I C  ####

# -----  A N D R O I D  -----
# more infos: https://t1p.de/ekkeCMakeAndroid

if(ANDROID)
    # ANDROID   A B I S
    # DO NOT USE in QtC: set(QT_ANDROID_ABIS "armeabi-v7a;arm64-v8a;x86_64")
    # DO NOT USE in QtC: set(QT_ANDROID_BUILD_ALL_ABIS "ON")
    # In QtC do it from 'Projects' Page:
    # - check 'Build Android App Bundle *.aab' at 'Build Android APK'
    # - check 'QT_ANDROID_BUILD_ALL_ABIS' at 'CMake Current Configuration'
    message(STATUS "QT_ANDROID_ABIS: ${QT_ANDROID_ABIS}")
    message(STATUS "QT_ANDROID_BUILD_ALL_ABIS: ${QT_ANDROID_BUILD_ALL_ABIS}")

    # ANDROID   V E R S I O N
    # Android VERSION_NAME and VERSION_CODE
    # VersionName per ex. 1.2.3

    # ekke's Apps are deployed in different ways:
    # APKs for armeabi-v7a, arm64-v8a, x86_64
    # App Bundles (aab) for Google Play Store
    # To install APKs for different arch's in MDM solutions
    # (Mobile Device Management)
```

```
    # the VERSION_CODE must be different for each arch:

    # aabcddeef
    # aa: 26 (MY_MIN_API)
    # b: 0, 1, 2 (MY_ARCH)
    # c: 0 (unused)
    # dd: 01 (Major Version)
    # ee: 02 (Minor Version)
    # f: 3 (Patch Version)

    # VersionCode examples
    # VersionCode 32 Bit: 260001023
    # VersionCode 64 Bit: 261001023

    # calculating MY_ARCH based on ANDROID_ABI:
    if(ANDROID_ABI STREQUAL "armeabi-v7a")
        set(MY_ARCH 0)
    elseif(ANDROID_ABI STREQUAL "arm64-v8a")
        set(MY_ARCH 1)
    else()
        set(MY_ARCH 2)
    endif()

    # Function to add leading zeros:
    function(pad_leading_zero number output)
        if(${number} LESS 10)
            set(${output} "0${number}" PARENT_SCOPE)
        else()
            set(${output} "${number}" PARENT_SCOPE)
        endif()
    endfunction()
    # Add leading zeros and create MY_APP_CODE
    pad_leading_zero(${MY_MAJOR_VERSION} MY_MAJOR_VERSION_PADDED)
    pad_leading_zero(${MY_MINOR_VERSION} MY_MINOR_VERSION_PADDED)
    set(MY_APP_CODE "${MY_MIN_API}${MY_ARCH}0${MY_MAJOR_VERSION_PADDED}$
{MY_MINOR_VERSION_PADDED}${MY_PATCH_VERSION}")

    # Important ANDROID   P R O P E R T I E S

    # ANDROID S O U R C E   D I R
    # Important for Android, will contain your custom Android Manifest
    # "Create Templates" (QtC->Projects->Build Steps->Build Android APK)
    # copies the default Qt Android Manifest into /android
    # see https://t1p.de/ekkeAndroidTemplates66
    # see https://t1p.de/ekkeAndroidTemplates67
    # Android Manifest in build dir should contain your customizations
    set(QT_ANDROID_PACKAGE_SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/android)

    # ANDROID   M I N   +   T A R G E T   A P I
    # MIN_SDK and TARGET_SDK will be moved by Qt build into gradle.properties,
    # you can check generated values in your build dir
    set(QT_ANDROID_MIN_SDK_VERSION ${MY_MIN_API})
    set(QT_ANDROID_TARGET_SDK_VERSION ${MY_TARGET_API})

    # ANDROID   P A C K A G E   N A M E
    # ATM the Android Package Name is contained in Android Manifest
    # This is deprecated - starting with Qt 6.8  there will be a
    # variable for the Package Name: QT_ANDROID_PACKAGE_NAME
    # see https://bugreports.qt.io/browse/QTBUG-106907
    # With 6.8 the package name can be removed from Android Manifest and
    # QT_ANDROID_PACKAGE_NAME will be moved by Qt build into gradle
    # uncomment set(QT_ANDROID_PACKAGE_NAME...) with 6.8+,
    # also set_target_properties() below
    ##set(QT_ANDROID_PACKAGE_NAME ${MY_UNIQUE_ID_ANDROID})
```

```
    # VERSION NAME and VERSION CODE also can be checked in build dir
    # output-metadata.json besides the generated APK
    set(QT_ANDROID_VERSION_NAME ${MY_APP_VERSION})
    set(QT_ANDROID_VERSION_CODE ${MY_APP_CODE})

    # ANDROID   A P P   N A M E
    # there's a problem yet. see
    # https://bugreports.qt.io/browse/QTBUG-121825
    # App and Activity Label defaults in the Manifest:
    # android:label="-- %%INSERT_APP_NAME%% --"
    # and cannot be overwritten from CMake using a variable
    # CMake will use the target (per ex. ${MY_APP} instead
    # My plan was to use ${MY_DISPLAY_NAME}, but only works in iOS
    # So you must set android and activity label manually in your manifest
    # per ex. android:label="QtWS"

    # uncomment QT_ANDROID_PACKAGE_NAME with 6.8+
    set_target_properties(${MY_APP}
        PROPERTIES
        QT_ANDROID_PACKAGE_SOURCE_DIR ${QT_ANDROID_PACKAGE_SOURCE_DIR}
        QT_ANDROID_MIN_SDK_VERSION ${QT_ANDROID_MIN_SDK_VERSION}
        QT_ANDROID_TARGET_SDK_VERSION ${QT_ANDROID_TARGET_SDK_VERSION}
        ##QT_ANDROID_PACKAGE_NAME ${QT_ANDROID_PACKAGE_NAME}
        QT_ANDROID_VERSION_NAME ${QT_ANDROID_VERSION_NAME}
        QT_ANDROID_VERSION_CODE ${QT_ANDROID_VERSION_CODE}
    )

    # MESSAGES ANDROID VARIABLES
    message(STATUS "QT_ANDROID_PACKAGE_SOURCE_DIR: $
{QT_ANDROID_PACKAGE_SOURCE_DIR}")
    message(STATUS "QT_ANDROID_MIN_SDK_VERSION: ${QT_ANDROID_MIN_SDK_VERSION}")
    message(STATUS "QT_ANDROID_TARGET_SDK_VERSION: $
{QT_ANDROID_TARGET_SDK_VERSION}")
    message(STATUS "QT_ANDROID_PACKAGE_NAME: ${QT_ANDROID_PACKAGE_NAME}")
    message(STATUS "QT_ANDROID_VERSION_NAME: ${QT_ANDROID_VERSION_NAME}")
    message(STATUS "QT_ANDROID_VERSION_CODE: ${QT_ANDROID_VERSION_CODE}")
endif()
#
# -----  A P P L E  -----
#
# more info about MACOSX_BUNDLE... properties:
# https://cmake.org/cmake/help/latest/prop_tgt/MACOSX_BUNDLE_INFO_PLIST.html
# see below also XCODE Attributes for IOS
if(APPLE)
    set_target_properties(${MY_APP} PROPERTIES
        MACOSX_BUNDLE TRUE
        MACOSX_BUNDLE_INFO_PLIST ${MY_INFO_PLIST}
        MACOSX_BUNDLE_BUNDLE_NAME ${MY_BUNDLE_NAME}
        MACOSX_BUNDLE_GUI_IDENTIFIER ${MY_UNIQUE_ID_IOS}
        MACOSX_BUNDLE_BUNDLE_VERSION ${MY_APP_VERSION}
        MACOSX_BUNDLE_SHORT_VERSION_STRING ${MY_MARKETING_VERSION}
        MACOSX_BUNDLE_COPYRIGHT ${MY_COPYRIGHT}
    )
endif()
#
# -----  I O S  -----
# more infos: https://t1p.de/ekkeCMakeIOS
#
if(IOS)
    # DEVICE FAMILY
    if(MY_APP_USES_IPHONE AND MY_APP_USES_IPAD)
        set(MY_DEVICE_FAMILY "1,2")
    elseif(MY_APP_USES_IPHONE)
        set(MY_DEVICE_FAMILY "1")
    elseif(MY_APP_USES_IPAD)
```

```
        set(MY_DEVICE_FAMILY "2")
    endif()

    # ASSETS - STORYBOARD - ICONS
    # Provide the launch screen and app icon asset catalog to Xcode,
    # that they get copied into bundles
    # 'MyLaunchScreen.storyboard' needs 'MyStoryboardImage.png'
    # (Adjust to your needs)
    set(my_ios_resource_assets
        "${CMAKE_CURRENT_SOURCE_DIR}/ios/${MY_STORYBOARD_NAME}.storyboard"
        "${CMAKE_CURRENT_SOURCE_DIR}/ios/Assets.xcassets"
        "${CMAKE_CURRENT_SOURCE_DIR}/ios/MyStoryboardImage.png"
    )

    target_sources(${MY_APP} PRIVATE ${my_ios_resource_assets})
    set_source_files_properties(${my_ios_resource_assets}
        PROPERTIES
            MACOSX_PACKAGE_LOCATION Resources
    )

    set_target_properties(${MY_APP} PROPERTIES
        MACOSX_BUNDLE_INFO_PLIST    "${MY_INFO_PLIST}"

        QT_IOS_LAUNCH_SCREEN    "${CMAKE_CURRENT_SOURCE_DIR}/ios/$
{MY_STORYBOARD_NAME}.storyboard"
        XCODE_ATTRIBUTE_INFOPLIST_KEY_UILaunchStoryboardName    $
{MY_STORYBOARD_NAME}
        XCODE_ATTRIBUTE_ASSETCATALOG_COMPILER_APPICON_NAME  ${MY_APP_ICON_NAME}

        XCODE_ATTRIBUTE_PRODUCT_BUNDLE_IDENTIFIER   ${MY_UNIQUE_ID_IOS}

        XCODE_ATTRIBUTE_PRODUCT_NAME    ${MY_APP}
        XCODE_ATTRIBUTE_INFOPLIST_KEY_CFBundleDisplayName   ${MY_DISPLAY_NAME}
        XCODE_ATTRIBUTE_TARGETED_DEVICE_FAMILY ${MY_DEVICE_FAMILY}
        XCODE_ATTRIBUTE_INFOPLIST_KEY_LSApplicationCategoryType "public.app-
category.${MY_APP_CATEGORY}"

        XCODE_ATTRIBUTE_CURRENT_PROJECT_VERSION ${MY_APP_VERSION}
        XCODE_ATTRIBUTE_MARKETING_VERSION   ${MY_MARKETING_VERSION}

        XCODE_ATTRIBUTE_INFOPLIST_KEY_NSHumanReadableCopyright  ${MY_COPYRIGHT}

        # Following example of a XCODE_ATTRIBUTE not working.
        # You can successfully set the attribute,
        # then check Xcode Build Settings-> Info.plist Values: OK
        # Unfortunately the checkbox isn't checked at Xcode's General-Tab
        # and it's not working when running on device.
        # So such kind of attributes must be hardcodced into the Info.plist
        # per ex:
        # XCODE_ATTRIBUTE_INFOPLIST_KEY_UIRequiresFullScreen  "YES"

        # Unfortunately same is for the ORIENTATIONS
        # XCODE_ATTRIBUTE_INFOPLIST_KEY_UISupportedInterfaceOrientations_iPhone
        # XCODE_ATTRIBUTE_INFOPLIST_KEY_UISupportedInterfaceOrientations_iPad
        # See my_cmake_snippets/cmake_ios_orientation.txt
    )

    if(MY_APP_USES_BLUETOOTH)
        set_target_properties(${MY_APP} PROPERTIES
            XCODE_ATTRIBUTE_INFOPLIST_KEY_NSBluetoothPeripheralUsageDescription  "$
{MY_BLE_PERIPHERAL_USAGE_DESCRIPTION}"
            XCODE_ATTRIBUTE_INFOPLIST_KEY_NSBluetoothAlwaysUsageDescription  "$
{MY_BLE_ALWAYS_USAGE_DESCRIPTION}"
        )
    endif()
```

```
    if(MY_APP_USES_MULTIMEDIA_CAMERA)
        set_target_properties(${MY_APP} PROPERTIES
            XCODE_ATTRIBUTE_INFOPLIST_KEY_NSCameraUsageDescription  "$
{MY_CAMERA_USAGE_DESCRIPTION}"
        )
    endif()
    if(MY_APP_USES_NATIVE_FILE_DIALOG)
        set_target_properties(${MY_APP} PROPERTIES
            XCODE_ATTRIBUTE_INFOPLIST_KEY_NSPhotoLibraryUsageDescription  "$
{MY_PHOTO_LIB_USAGE_DESCRIPTION}"
        )
    endif()

    message(STATUS "MY_APP_USES_IPHONE: ${MY_APP_USES_IPHONE}")
    message(STATUS "MY_APP_USES_IPAD: ${MY_APP_USES_IPAD}")
    message(STATUS "MY_DEVICE_FAMILY: ${MY_DEVICE_FAMILY}")
    # orientation messages not yet
endif()

if(WIN32)
    # Windows not used yet
    set_target_properties(${MY_APP} PROPERTIES
        WIN32_EXECUTABLE TRUE
    )
endif()

# ####  F I N A L   S T E P S  ####

# ----- 3rd   P A R T Y  -----
# more infos: ekkeCMake3rdParty

# MobileUI - colorize the StatusBar and more
# works on Android, iOS, macOS
# https://github.com/emericg/MobileUI
# in Qt 5.15 JP Nurmi's Statusbar was used
# more info: https://t1p.de/ekkeStatusbar
if(MY_APP_USES_MOBILE_UI)

    # if(ANDROID)
    # MobileUIDemo uses target_link_libraries
    # Qt6::CorePrivate, but seems to work without (ToDo)
    # so ATM Qt6::CorePrivate only used for SHARING or NATIVE_FILE_DIALOG
    # see below

    if(IOS)
        target_link_libraries(${MY_APP} PRIVATE
            Qt6::GuiPrivate
            "-framework UIKit"
        )
    endif()

    # Hint: MobileUI is outside the project directory,
    # in this case, 'add_subdirectory()' needs 2nd argument:
    add_subdirectory(../../_qt_ws/MobileUI ../../_qt_ws/MobileUI)
    target_link_libraries(${MY_APP} PRIVATE
        MobileUI
    )

endif()

if(ANDROID)
    if(MY_APP_USES_NETWORK)
        # OPENSSL
        # https://github.com/KDAB/android_openssl
        # points to OpenSSL installed by QtCreator with Android SDK Tools
```

```
        include(/Applications/daten/_android_sdk_tools/android_openssl/
android_openssl.cmake)
        add_android_openssl_libraries(${MY_APP})
    endif()
endif()

# ----- L I N K I N G  -----
# link to Qt6::Core automagically linked from qt_add_executable()
target_link_libraries(${MY_APP}
    PRIVATE
        Qt6::Gui
        Qt6::Qml
        Qt6::Quick
        Qt6::QuickControls2
)
if(MY_APP_USES_NETWORK)
    target_link_libraries(${MY_APP}
    PRIVATE
        Qt6::Network
    )
endif()
if(MY_APP_USES_BLUETOOTH)
    target_link_libraries(${MY_APP}
    PRIVATE
        Qt6::Bluetooth
    )
endif()
if(MY_APP_USES_MULTIMEDIA_CAMERA)
    target_link_libraries(${MY_APP}
    PRIVATE
        Qt6::Multimedia
    )
endif()
if(MY_APP_USES_NATIVE_FILE_DIALOG)
    if(IOS)
        # info from @MikeWhite:
        # https://bugreports.qt.io/browse/QTBUG-105954
        # docs still wrong - only QMake infos
        qt_import_plugins(${MY_APP}
            INCLUDE
            Qt::QIosOptionalPlugin_NSPhotoLibraryPlugin
        )
        # ToDo must we link to these frameworks ?
        # Photos found in some other CMakeLists
        # AVFoundation in QMake for PhotoLibraryPermissions
        target_link_libraries(${MY_APP}
        PRIVATE
            # "-framework Photos"
            # "-framework AVFoundation"
        )
    endif()
endif()
if(MY_APP_USES_SHARING OR MY_APP_USES_NATIVE_FILE_DIALOG)
    if(ANDROID)
        # see also 'Android ScopedStorage + FileDialog':
        # https://t1p.de/ekkeAndroidFileDialog
        target_link_libraries(${MY_APP}
        PRIVATE
            Qt6::CorePrivate
        )
    endif()
endif()

# ToDo  SOMETHING SPECIAL ? - MORE FRAMEWORKS NEEDED ?
if(IOS)
```

```
    # Security needed for RSA ObjC classes
    target_link_libraries(${MY_APP}
    PRIVATE
        # "-framework Security"
    )
endif()

# ----- I N S T A L L  -----

# don't know if include(GNUInstallDirs) really needed
# for Android and iOS apps built on macOS
# but it is harmless ;-)
include(GNUInstallDirs)

install(TARGETS ${MY_APP}
    # Bundle used by macOS
    BUNDLE  DESTINATION .
    # Library Destination used by Android
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)

# In some projects I found qt_generate_deploy_qml_app_script()
# My apps seem to work without, but I got the advice,
# for my QML - Android, iOS, macOS Apps
# it's better to use the deploy_script:
qt_generate_deploy_qml_app_script(
    TARGET ${MY_APP}
    OUTPUT_SCRIPT deploy_script
    MACOS_BUNDLE_POST_BUILD
    NO_UNSUPPORTED_PLATFORM_ERROR
    DEPLOY_USER_QML_MODULES_ON_UNSUPPORTED_PLATFORM
)
install(SCRIPT ${deploy_script})

# ---------------------------- this is the end, my friend --------------
```