Distribution of Qt apps to other platforms

...

Jan Grulich

About me

- Red Hat
- Fedora
- KDE

About this presentation

- Overview of tools I have some experience with
- Examples of CMake usage
- Not much about Linux

CMake support

- Platform defines:
  - APPLE, WIN32, UNIX
- Compiler defines:
  - MINGW, MSVC
- Window library with no exported symbol has to be a MODULE
  - add_library(target MODULE ${SOURCES})

Qt support

- Platform defines:
  - Q_OS_MACOS, Q_OS_LINUX, Q_OS_WIN (Q_OS_WIN32/64)
- Compiler defines:
  - Q_CC_MSVC, Q_CC_GNU, Q_CC_CLANG

Windows requirements overview

- Manifest file
  - XML file
  - Informs Windows how to handle your application
    - Privileges
    - Theming, DPI awareness, Compatibility etc.
- Installer
  - Probably no description needed
- Signing installer
  - Let users know where your application comes from

First step: Getting Qt

- From official Qt installer
- Using MinGW packages on Linux
- GitHub tip:
  - When using Github Actions to produce Windows builds, you can use:
    - jurplel/install-qt-action@v2
      - More Qt versions
      - More compilers: MSVC (2015, 2017, 2019), MingW (7.3, 8.1)

CMake: Adding Windows manifest file

- target_sources(<targetname> PRIVATE <path/to/windows.rc>)

```
#include “winuser.h”

CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST “windows.manifest”
IDI_ICON1 ICON “icons/icon.ico”
```

Example of Windows manifest file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
   <security>
     <requestedPrivileges>
       <requestedExecutionLevel level="requireAdministrator" uiAccess="false"/>
     </requestedPrivileges>
   </security>
 </trustInfo>
</assembly>
```

CMake tips: disabling terminal window

```
set_target_properties(<targetname> PROPERTIES WIN32_EXECUTABLE TRUE)

add_executable(<targetname> WIN32 ${SOURCES})
```

Preparing files for installer - manually

- Include system libraries like:

```
libstdc++-6.dll, libgcc_s_dw2-1.dll, libssp-0.dll,
libwinpthread-1.dll, libcrypto-1_1.dll, libssl-1_1.dll
```

- Include Qt plugins like:

```
qjpeg.dll, qsvg.dll, qwindows.dll
```

- Include Qt libraries like:

```
Qt5Core.dll, Qt5Gui.dll, Qt5Quick.dll
```

- Include QML modules like:

```
Controls.2, Dialogs, Layouts
```

Preparing files for installer - cleverly

- Run windeployqt:

```
windeployqt -qmldir path/to/qml/files your_windows_binary
```

- Running the command above will bundle:
  - Qt libraries
  - Qt plugins
  - QML modules
- System libraries still have to be copied manually

Creating installer 1/2

- NSIS
  - Open source system to create Windows installers
  - Script based
  - Allow you to define:
    - What gets installed/uninstalled
    - Application name, version, link, size etc.
    - Shortcut

Creating installer 2/2

- MakeNSIS
  - Generates an installer executable from a NSIS script
  - Get it as MinGW package on Linux or using Chocolatey on Windows
- Osslsigncode
  - Tool to sign your installer
  - Example:

```
osslsigncode sign -pkcs12 path/authenticode.pfx -readpass “pass”
-h sha256 -n “App Name” -i “https://app.org” -t
“https://timestamp.comodoca.com/authenticode” -in installer.exe
-out installer.exe.signed
```

Ship It!

Mac OS

MacOS requirements overview

- Creating MacOS bundle:
  - Writing Info.plist file
  - Bundling all libraries/frameworks
  - Signing binaries
  - Creating DMG file
  - Notarizing your bundle

First step: Getting Qt

- Using Homebrew

```
brew install qt5
brew install cmake
```

- Using official installer

CMake: Creating MacOS bundle

```
add_executable(<target> MACOSX_BUNDLE ${SOURCES} path/to/icon.icns)

set_target_properties(<target> PROPERTIES
    OUTPUT_NAME “App name”
    MACOSX_BUNDLE TRUE
    MACOSX_BUNDLE_INFO_PLIST path/to/Info.plist
    MACOSX_DEPLOYMENT_TARGET 10.9)

set_source_file_properties(pat/to/icon.icns PROPERTIES
MACOSX_PACKAGE_LOCATION “Resources)
```

Information property list (Info.plist)

- Contains data for bundle
- You can use your own template or the one provided by CMake
- Format is XML:

```
<dict>
    <key>CFBundleIconFile</key>
    <string>${MACOSX_BUNDLE_ICON_FILE}</string>
    <key>CFBundleLongVersionString</key>
    <string>${MACOSX_BUNDLE_LONG_VERSION_STRING}</string>
</dict>
```

CMake: Completing MacOS bundle 1/2

- Using macdeployqt tool:

```
get_target_property(QMAKE_EXECUTABLE Qt5::qmake IMPORTED_LOCATION)
get_filename_component(QT_BIN_DIR “${QMAKE_EXECUTABLE}” DIRECTORY)
find_program(MACDEPLOYQT_EXECUTABLE macdeployqt HINTS “${QT_BIN_DIR}”)


add_custom_command(TARGET <target> POST_BUILD
                   COMMAND “${MACDEPLOYQT_EXECUTABLE}”
                           “path/to/App Name.app”
                           -qmldir=”path/to/qml/files”
                           -executable=”path/to/App Name.app/Contents/MacOS/foo
                   COMMENT “Deploying Qt..”)
```

Completing MacOS bundle 2/3

- Using macdeployqt will:
  - Bundle all Qt libraries, Qt plugins and QML imports
  - Fix path to bundled libraries:

```
otool -L QtGui.framework/QtGui

Output:
/path/to/Qt/lib/QtGui.framework/Version/5.15/QtGui
(compatibility version 5.15.0, current version 5.15.2)
/usr/lib/libstdc++.6.dylib
/system/Library/Frameworks/ApplicationServices.framework/Versions/A/Appl
icationServices
```

Completing MacOS bundle 3/3

- Copying additional libraries manually:

```
cp -R path/to/library “App Name.app/Contents/Frameworks”

install_name_tool -change ${library} “@executable_path/../Frameworks/${basename
library}/ “App Name.app/Contents/MacOS/app_binary”
```

Signing binaries

- Let users know where your application comes from
- Sign all binaries (including libraries/frameworks):

```
codesign -s “$DEVELOPER_ID” --deep -v -f “$library -o runtime

macdeployqt -codesign=$DEVELOPER_ID
```

Creating DMG file

- DMG is a file container for apps on MacOS
- Create DMG file:

```
macdeployqt -dmg

hdiutil create -srcfolder “App Name.app” -format UDCO -imagekey zlib-level=9
-scrub -volname AppName-osx AppName-osx-$VERSION.unnotarized.dmg
```

Notarization

- Checks for malicious components
- Checks for code-signing issues
- Send your app for notarization with:

```
macdeployqt -sign-for-notarization=$DEVELOPER_ID

xcrun altool --notarize-app --primary-bundle-id “org.example.AppName”
--username “$USERNAME” --password “@keychain:NOTARIZATION_PASSWORD” --file
“AppName-osx-$VERSION.unnotarized.dmg”
```

Final step: Completed notarization

- After you receive an email that your app has been notarized

```
xcrun stapler staple “App Name.app”

hdiutil … to create DMG file again
```

Ship It!

Questions?

Thank you