**Using CMake to Generate Plugin Projects (8C)**
Will Pirkle

This document explains how to install and run CMake on your exported projects that RackAFX has generated for you. It also includes specifics to each API including the location of your final plugin.



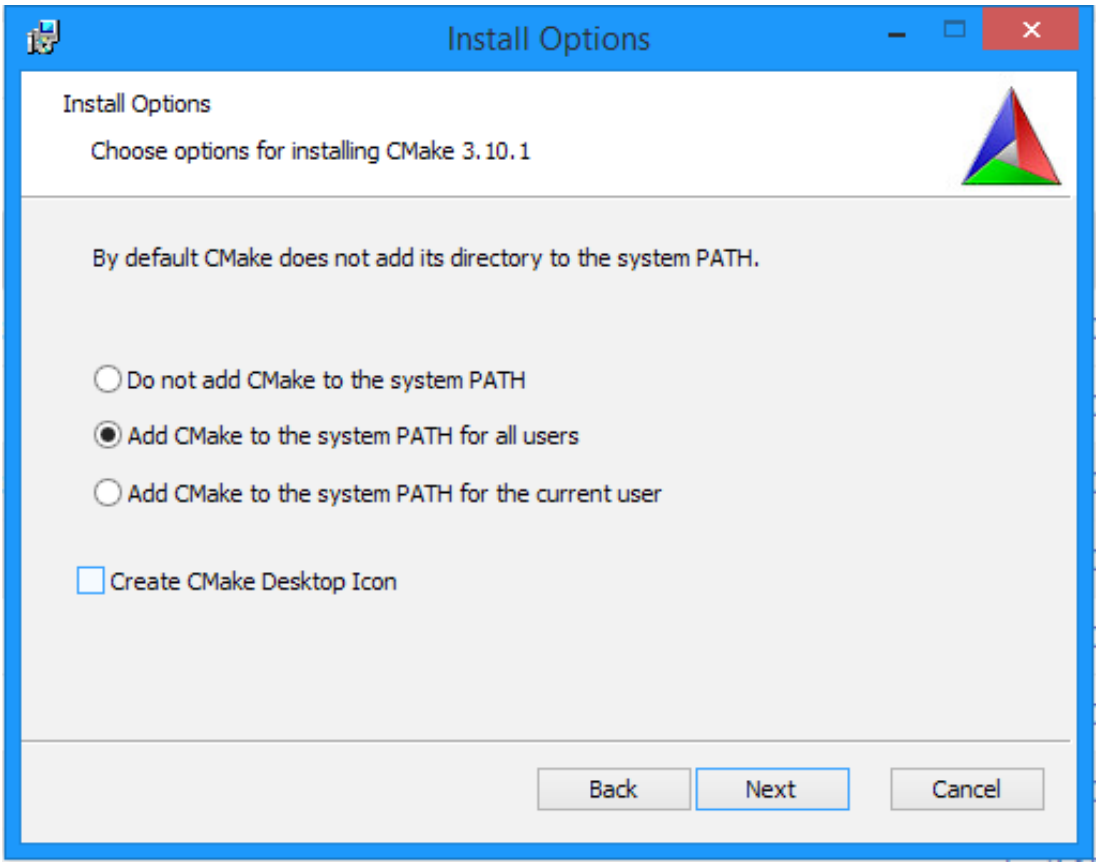
## 6. Installing CMake

In order to generate the RackAFX ported project compiler files, you need to have CMake installed on your OS. For ease, you should add the CMake executable to your default path.

Download the latest version of CMake at http://cmake.org/

### 6.1 Windows

During installation, you have the option of adding CMake to your default path list.




Choose this option and you are done.

### 6.2 MacOS

Open Terminal and type:

*echo $PATH*

which produces a colon : separated list of folders you can run directly with terminal;

*/usr/local/bin* should be in this list

Next, add a SymLink to the CMake executable and place it in the */usr/local/bin* folder in Terminal by running:

*sudo ln -s /Applications/cmake.app/Contents/bin/cmake /usr/local/bin*

After that, you are done.

## 7. Running CMake

In each case (Windows vs MacOS), you first navigate to the <OS>_build folder inside of the project folder. For Windows it is named *win_build* and for MacOS it is named *mac_build*. You will notice that these folders are empty when you navigate to them. This is correct for a freshly created project.

Running CMake will populate the build folder with the newly created XCode or Visual Studio project. After that you can open the compiler project and rebuild your solution as usual. To run CMake:

1. start your engines:

- **MaxOS**: open a Terminal shell
- **WinOS**: open the Command Prompt with admin privileges (aka "Command Prompt (Admin)")

2. Navigate and find your *myprojects* folder inside a particular SDK (individual) or in the ALL_SDK folder (universal).

3. Navigate into your ported project folder and then again into either the *win_build* or *mac_build* sub-folders according to your OS. You must run CMake from folder *inside* of the one that contains the outer CMakeLists.txt file. This is a quirk of CMake.

4. Run CMake as follows:

### 7.1 Windows

- VS 2015 (for VST3 and AAX) 64-bit:

  *cmake -G"Visual Studio 14 2015 Win64" ../*

- VS 2015 (for VST3 and AAX) 32-bit:

  *cmake -G"Visual Studio 14 2015" ../*

- VS2017 (for VST3 only - currently not supported by AAX as of SDK 2p3p0, but may be supported in the future) 64-bit:

  *cmake -G"Visual Studio 15 2017 Win64" ../*

- VS2017 32-bit:

  *cmake -G"Visual Studio 15 2017" ../*

You can find the CMake generator (G) codes for other Visual Studio compilers here:

https://cmake.org/cmake/help/v3.10/manual/cmake-generators.7.html
NOTE: the VSTGUI SDK requires C++11 so make sure your compiler is 100% C++11 compliant. Many older versions of Visual Studio are NOT 100% compliant so beware.

### 7.2 MacOS

*cmake -GXcode ../*

There are no version-specific or bit-depth specific Xcode generators.

## 8. Building Your Exported Project

You can find the resulting VS or Xcode project located right inside your *win_build* or *mac_build* subfolders respectively. Double-click on the .xcodeproj or .sln file to start your compiler.

Your ported projects will contain multiple targets (XCode) or projects (Visual Studio). These include:

- **ZERO_CHECK**: this will rerun cmake. You can/should execute this after changing something in your CMake files due to adding new files to your project (see below)

- **ALL_BUILD**: is simply a target which builds all projects in the active solution; you will usually just build this target

### 8.1 Compile Targets

Xcode calls each project a “target” while Visual Studio uses “Project” - each of these compiles to produce either your final plugin or a library (or multiple libraries) needed to compile it.

For individual projects, you will only have target(s) associated with one particular API. For universal projects, there will be targets for everything required to build for all APIs.

**VST3:**
The VST3 compiler project will include targets for two executables: *validator* and *editorhost*. These are automatically added by a CMakeLists.txt file that is internal to the VST3 SDK and that we do not want to modify. The *validator* project is required in the solution. The *editorhost* is optional and you can safely ignore it. See the VST3 section below for more information about the *validator*.

### 8.2 Notes on API Specifics

There are some specifics regarding each API that you need to be aware of. These are properties of the APIs/SDKs themselves, and have nothing to do with the RackAFX projects directly.

#### 8.2.1 AAX:

1. you must compile the AAX Base Library first. You only need to do this once per SDK release (or if you update your compiler to a newer version).

You can find the XCode and Visual Studio generated library projects in these folders:

AAX_SDK*/Libs/AAXLibrary/WinBuild*

or

AAX_SDK*/Libs/AAXLibrary/MacBuild*

Open the project file and compile your project using the same compiler that you will use to compile your project. Anytime you change compilers, you must re-compile the library with your new compiler. The result of the library compilation will be:

**MacOS**
AAX_SDK/Libs/Debug/libAAXLibrary.a
AAX_SDK/Libs/Release/libAAXLibrary.a

**Windows x64**
AAX_SDK/Libs/Debug/AAXLibrary_x64_D.lib
AAX_SDK/Libs/Release/AAXLibrary_x64.lib

2. Open the compiler project for your newly generated project. You can now re-build the solution to finish the ported project.

3. Your finished plugin will be located in the following sub-folder:

*AAX_SDK/myprojects/<project name>/**mac_build**/**AAX**/<config>*

or

*AAX_SDK/myprojects/<project name>/**win_build/AAX**/<config>*

where *<config>* is either Debug or Release, depending on how you built the project.

4. Copy the .aaxplugin into your AAX plugin folder. You must to this manually because in both operating systems, the target folder is admin-protected which prohibits the copy mechanism from being done programmatically during compilation.

Copy the plugin into the folder here:

**MacOS**

*/Library/Application Support/Avid/Audio/Plug-Ins*

**Windows**

*C:\Program Files\Common Files\Avid\Audio\Plug-Ins*

Now you can run your Pro Tools Developer's Build to test your plugin and generate the Presets for it.

### 8.2.2 AU

1. Open the compiler project for your newly generated project. You can now re-build the solution to finish the ported project.

2. Your finished plugin will be located in the following sub-folder:

   *AU_SDK/myprojects/<project name>/**mac_build/AU**/<config>*

where *<config>* is either Debug or Release, depending on how you built the project.

3. For AU, your plugin is AUTOMATICALLY copied to the proper location in your MacOS device:

   *~/Library/Audio/Plug-Ins/Components/<plugin>.component)*

4. run *auval* from Terminal to validate your plugin prior to use. The RackAFX ported projects will pass validation by default, however your customizations could create issues, especially if you are still new to plugin development. You received the *auval* terminal code when you ported the project in RackAFX. If your plugin fails validation, it will be black-listed in Apple Logic and perhaps other clients as well so you should correct any errors before testing.

### 8.2.3 VST

1. Open the compiler project for your newly generated project. You can now re-build the solution to finish the ported project.

2. Your project will be validated automatically with the validator project that is built-into your compiler project. If your plugin fails validation, the compiler will NOT create the final VST plugin. This is generally good as a plugin that fails validation will likely crash the VST client which may result in it being black-listed from future loads! Fix any validation problems (see the compiler output window).

3. Your finished plugins will be located in the following sub-folder:

   *VST_SDK/VST3_SDK/myprojects/<project name>/**mac_build/VST3**/<config>*

   or

   *VST_SDK/VST3_SDK/myprojects/<project name>/**win_build/VST3**/<config>*

where *<config>* is either Debug or Release, depending on how you built the project.

**MacOS**
Your VST2 and VST3 plugins are AUTOMATICALLY copied to the proper locations in your MacOS device:

VST2:

*~/Library/Audio/Plug-Ins/VST/<plugin>.vst)*

VST3:

*~/Library/Audio/Plug-Ins/VST3/<plugin>.vst3)*

**Windows**

You must manually copy the VST plugins to the proper locations - these will vary depending on your setup.

VST3:

There are generally accepted "default" locations for 32 and 64 bit builds. These are located in admin-protected folders and can't be copied programmatically:

32-bit

*C:\Program Files (x86)\Common Files\VST3*

64-bit

*C:\Program Files\Common Files\VST3*

You should copy your .vst3 plugin to the appropriate folder as well as any other DAW-specific VST3 folder that exist on your system.

VST2:

There is no default location for VST2 plugins.

You should copy your .vst3 plugin into your VST2 folder(s) for your DAW(s). THEN, rename the suffix from ".vst3" to ".dll" to convert it from a VST3 to a VST2 plugin.

## 9. Changing from Individual to Universal Exports

When you export a project from RackAFX, you have the option of creating individual exports, or the universal export pattern. But, you actually receive the same files for each option - only the outer most *CMakeLists.txt* file is different. If you open this file and check out the very top most portion, you will find the BOOLEAN variables that turn on and off the various API projects as well as the default locations for the SDKs when you are using the Universal API Export.

The individual projects may be enabled/disabled with the code below. If you enable the UNIVERSAL_SDK_BUILD, then all API projects will be generated regardless of the settings below it.

```
# --- Universal Build Flag
set(UNIVERSAL_SDK_BUILD TRUE)
```

To enable/disable individual projects when NOT using the universal build, alter the code below it:

```
# --- Individual project builds
set(AAX_SDK_BUILD TRUE)
```

```
set(AU_SDK_BUILD FALSE)
set(VST_SDK_BUILD FALSE)
```

Here, I've set CMake to generate only the AAX project.

## 10. Modifying your Project: CMake Ramifications

Ordinarily, you will export your projects from RackAFX once they are finally completed, or you will generate a series of ports up until the completion point. At this point, you will usually just build the plugin, test it and release it.

However you may decide to alter your existing exported project *after the fact*, and in this case, you will need to alter your CMake files accordingly if you plan on updating the project when a future version of a given SDK is released.

**Adding/Removing/Relocating Files**

If you add, remove, or relocate project files you will need to update the *CMakeLists.txt* file for each API that you are supporting or compiling against. Navigate into the appropriate folder within your project:

../*myprojects/<project name>/project_source/cmake*

Inside of this folder are a set of sub-folders, one for each API and each containing the *CMakeLists.txt* file for that specific API.

../*myprojects/<project name>/project_source/cmake/**aax_cmake***

../*myprojects/<project name>/project_source/cmake/**au_cmake***

../*myprojects/<project name>/project_source/cmake/**vst_cmake***

Your project source files are listed at the very top and are identical for each API's *CMakeLists.txt* file. NOTE: yes, there are ways to combine the portions of these files but in general I've found it safer to simply keep each file independent. Add, remove or relocate by altering the lines of text after the *set(rafx_<api>_sources)* chunk. ${RAFX_SOURCE_ROOT} is your rafx_source folder that contains the files you originally wrote in RackAFX:

```
set(rafx_aax_sources
    ${RAFX_SOURCE_ROOT}/DelayLine.h
    ${RAFX_SOURCE_ROOT}/DynamicsProcessor.h
    ${RAFX_SOURCE_ROOT}/GUIViewAttributes.h
    etc…
```

Save the files and re-run CMake. You may need to delete the existing files from your *mac_build* or *win_build* folders first if CMake complains that something has changed in the cache file. After re-running CMake, your project will reflect the changes properly.

**Deep Changes**

If you decide to add 3rd party components, libraries, or other stuff to your project which requires that you alter the project settings (in Xcode these are in the target's *Build Settings* while in Visual Studio they are the Project's *Properties)*, then you have a bit of work to do to alter the CMake files to use the new compiler settings. This may be simple or it may be very complicated depending on the changes you have made. There is no way to tell in advance what these changes may be, so the only help I can give you is to use the information at cmake.org as well as other sites like stackoverflow.com which has scores of help topics related to CMake compiler settings.