# Tutorial: creating an Usine user module on MacOS X

Version 1.0 (29th April 2018) by Benoit Bouchez (alias iModularSynth)

Version 2.0 (20th November 2023): Overhaul

## Introduction

This document is a short tutorial explaining how to create "from scratch" a user module for Usine using Xcode on MacOS X platform. It is not meant to explain how Usine modules are working, the SDK User Manual made by Martin Fleurent is perfect for that.

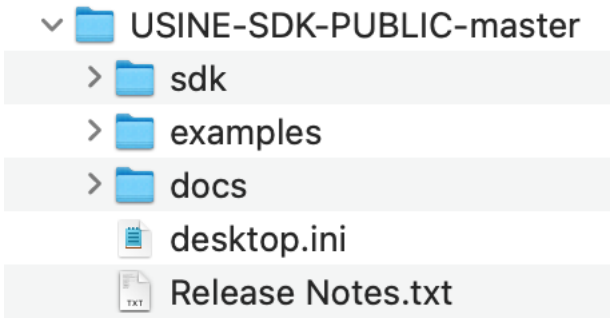
This tutorial was made using Usine Hollyhock's SDK 6 and Xcode 14.

## Preparing the computer

First, Xcode must be installed on your computer. Xcode is available for free on Apple website.

Download the Usine SDK BrainModular's website and copy the whole directory on your hard disk. The SDK contains excellent examples to understand how Usine works and what you must implement to make your own modules. We will concentrate here on the steps to follow to create modules from scratch (you can then copy source code from the projects in the SDK to have a good starting point).

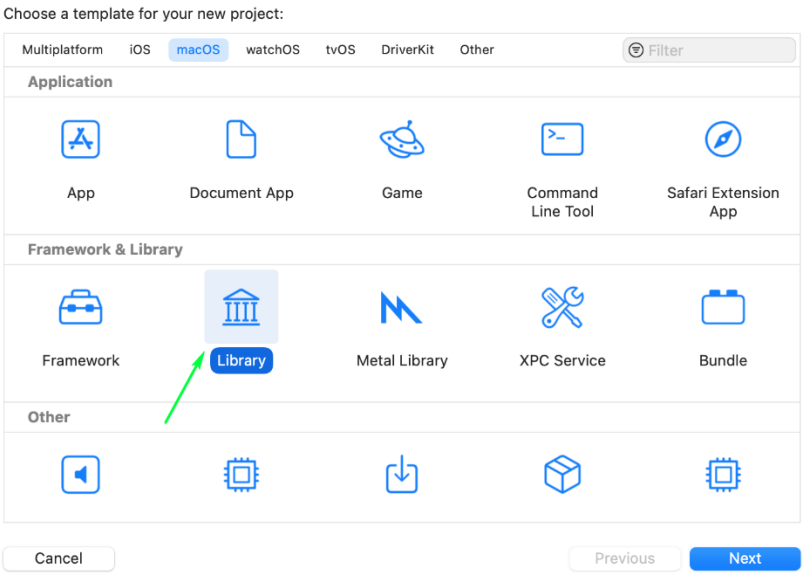
Once the SDK is copied on your hard disk, you should see the following structure in the SDK folder:



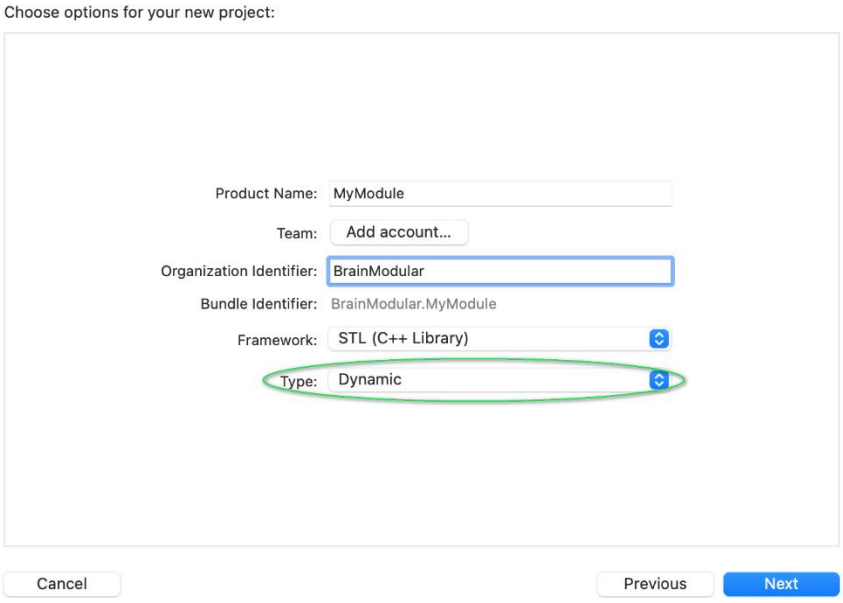

Technically, Usine user modules are nothing else than a dylib (Dynamic Library), but you need to define some specific properties in the project to make sure that Usine can load and use them.

## Creating the project in Xcode

Start Xcode. Click on "Create a new Xcode project". On the next page navigate to the "MacOS" tab, click on "Library" under the "Framework and library" header:



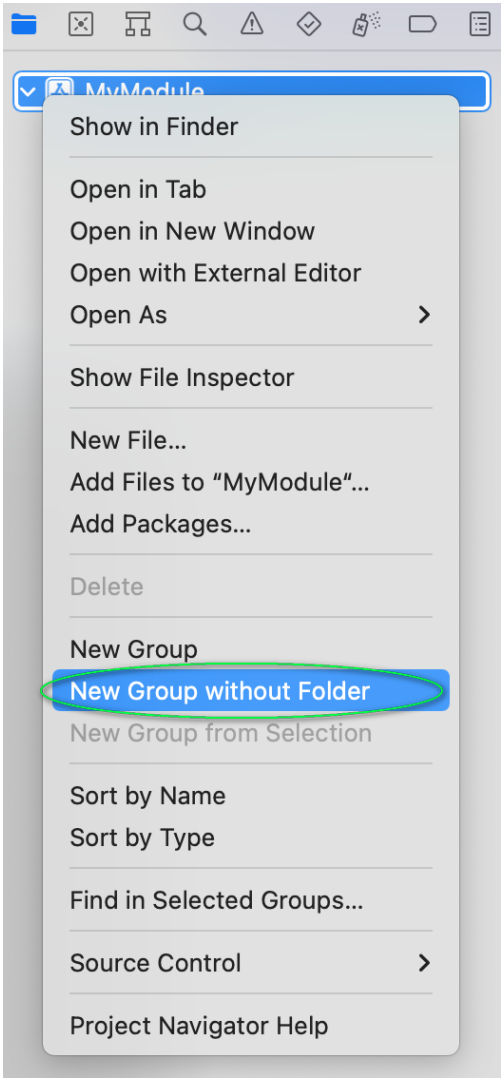

Next, choose the wanted name of the module as well your company's name and make sure the type at the bottom of the page is set to "dynamic":



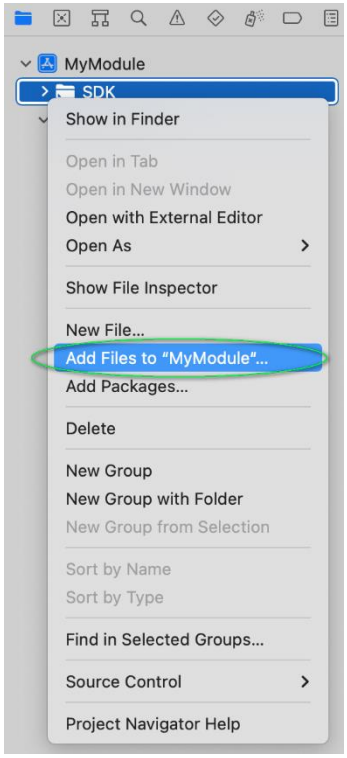

Xcode will create some dummy files. Feel free to keep or discard them.

## Including Usine's SDK

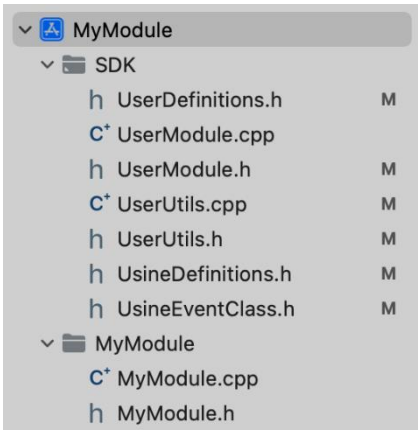
Right click on your project at the top of the project navigation bar at the left of the Xcode window then select "New group" and change its name to "SDK" (this step is not mandatory per say but tidy code is important):



Right click on the newly created “SDK” group then select “Add files to “YourProject”…”:




Navigate to the location of the Usine SDK on your computer and select every file in the “sdk” subfolder (shift + left click) and add them. Now add your own class to the project and you should now have a similar architecture to this:



The structure of your project is now complete however you now need to configure build settings before you can use your module in Usine.

## Setting the build settings

Due to the need to build for two different architectures on MacOS (Intel and Silicon) you may need to set up multiple targets:

- **Universal**, works for both architectures.
- **x86_64**, for machines using an Intel processor (deprecated).
- **arm64**, for machines using an Apple Silicon processor (deprecated).

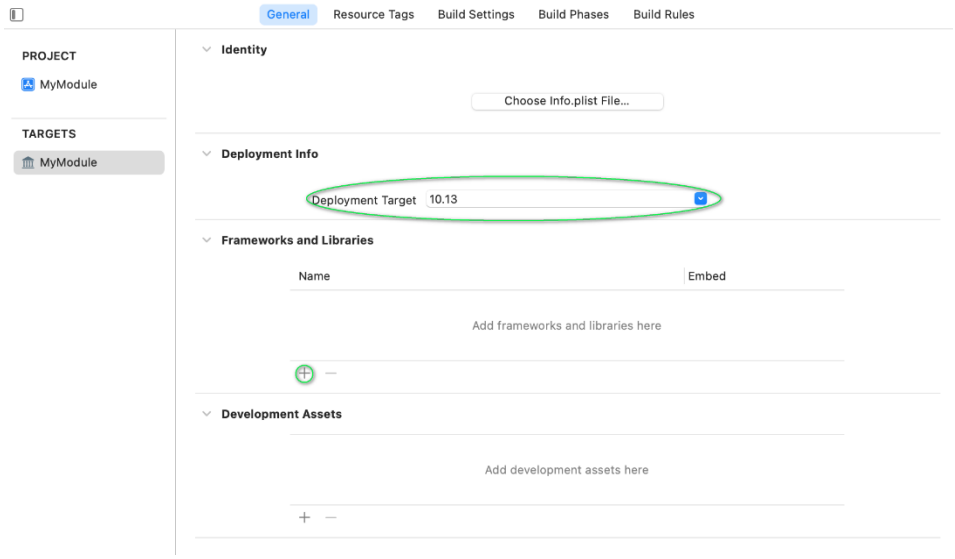
Universal binaries are a newish concept in MacOS development, they allow you to compile only once, do not require multiple targets and simplify distribution. This is obviously the preferred way to build modules currently, but as of now and for the foreseeable future, library support for universal binaries is sloppy to nonexistent and building universal binaries with external libraries can prove to be simply impossible.

Point being, you should always try to build universal binaries.

In the coming example, we will set up a universal binary build, additional instructions for architecture specific targets will be given thereafter.

## Building a universal binary

Left click on your project at the top of the project navigation bar and look at the "TARGETS" header on the left. Xcode should have created one target by default, click on it. The first step is to change the deployment target, select "10.13", you can also select your needed libraries here ("Frameworks and libraries" > click on "+" > "Add other...") if your project requires dilybs you must also copy them to the bin folder in Usine's contents:




You'll then need to change a few settings in the "Build Settings" tab, in this tab select "All" at the top of the tab to display all setting, then change the following:

- "Build Active Architecture Only", set to "No":

  **Build Active Architecture Only** **No**

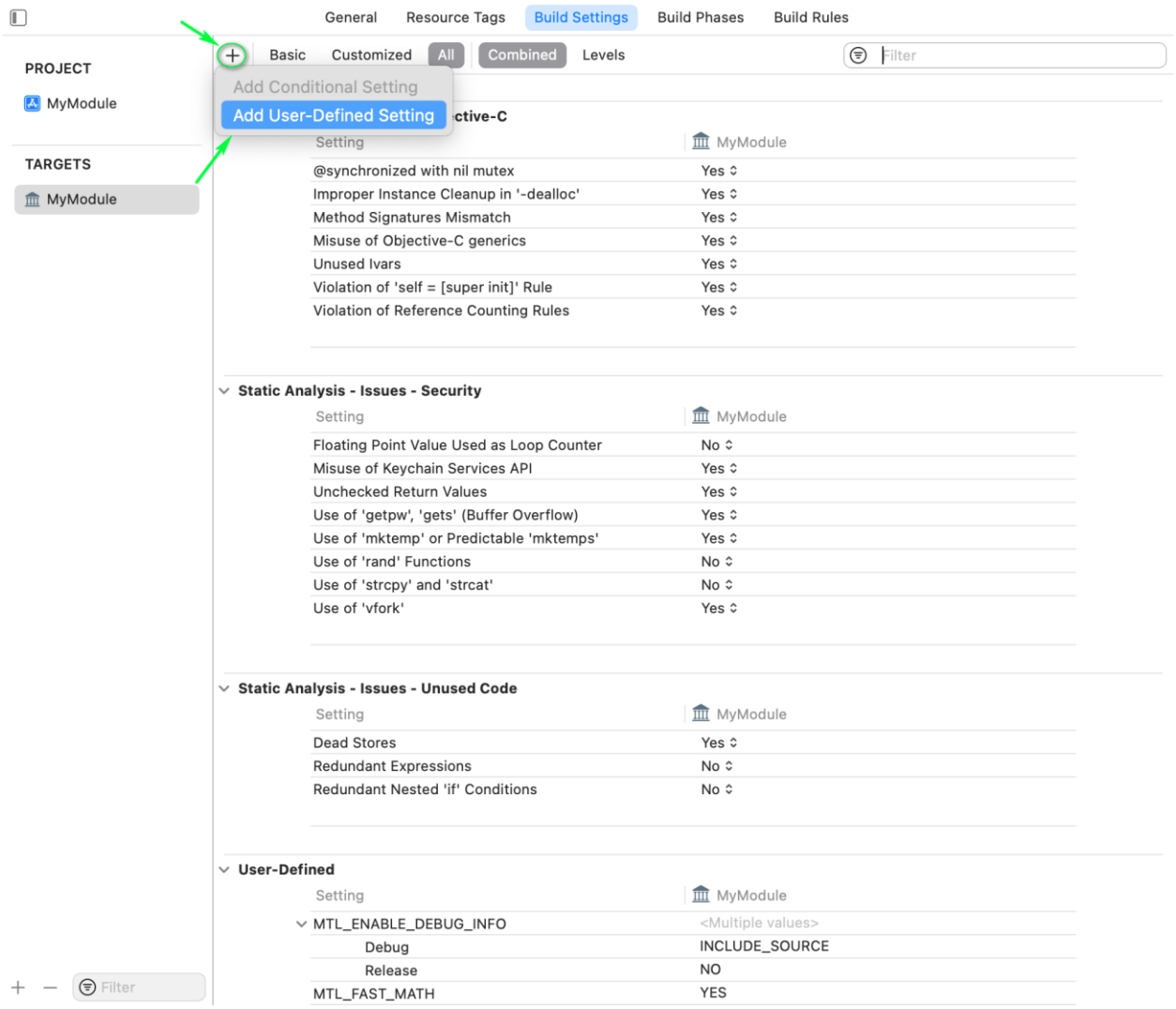
- "Executable Extension", set to "usr-macos64uni":

  **Executable Extension** **usr-macos64uni**

- "Executable Prefix", clear the line to remove "lib":

  **Executable Prefix**

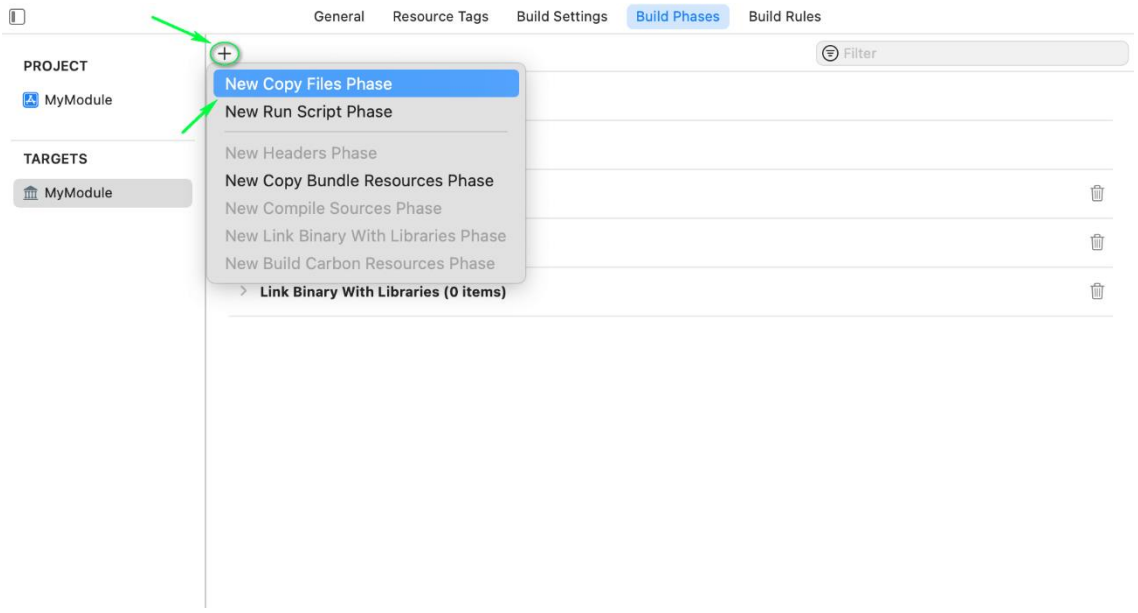
You must also create a new user-defined setting by clicking on the "+" at the top of the page and clicking on "Add User-Defined Setting":



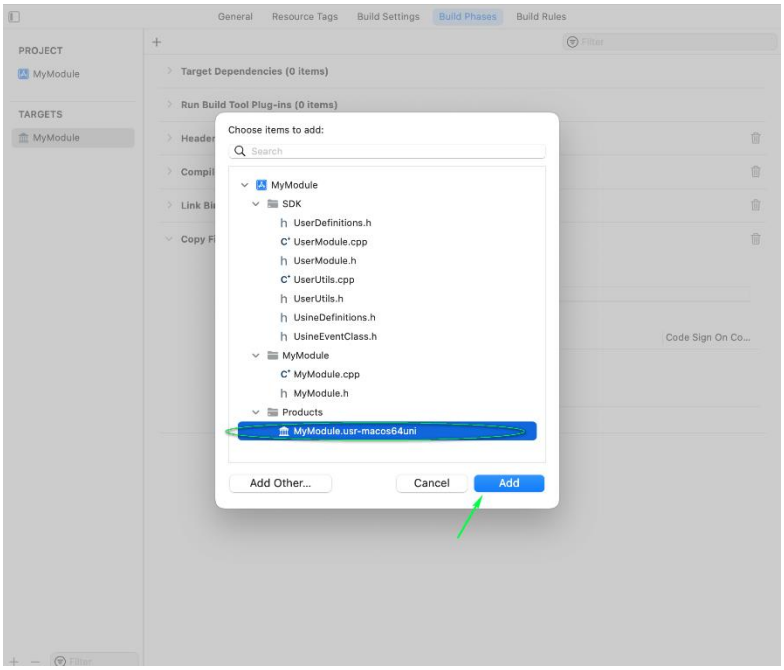

Set the newly created setting's name to "VALID_ARCHS" and its value to "x86_64 arm64":

**VALID_ARCHS** **x86_64 arm64**

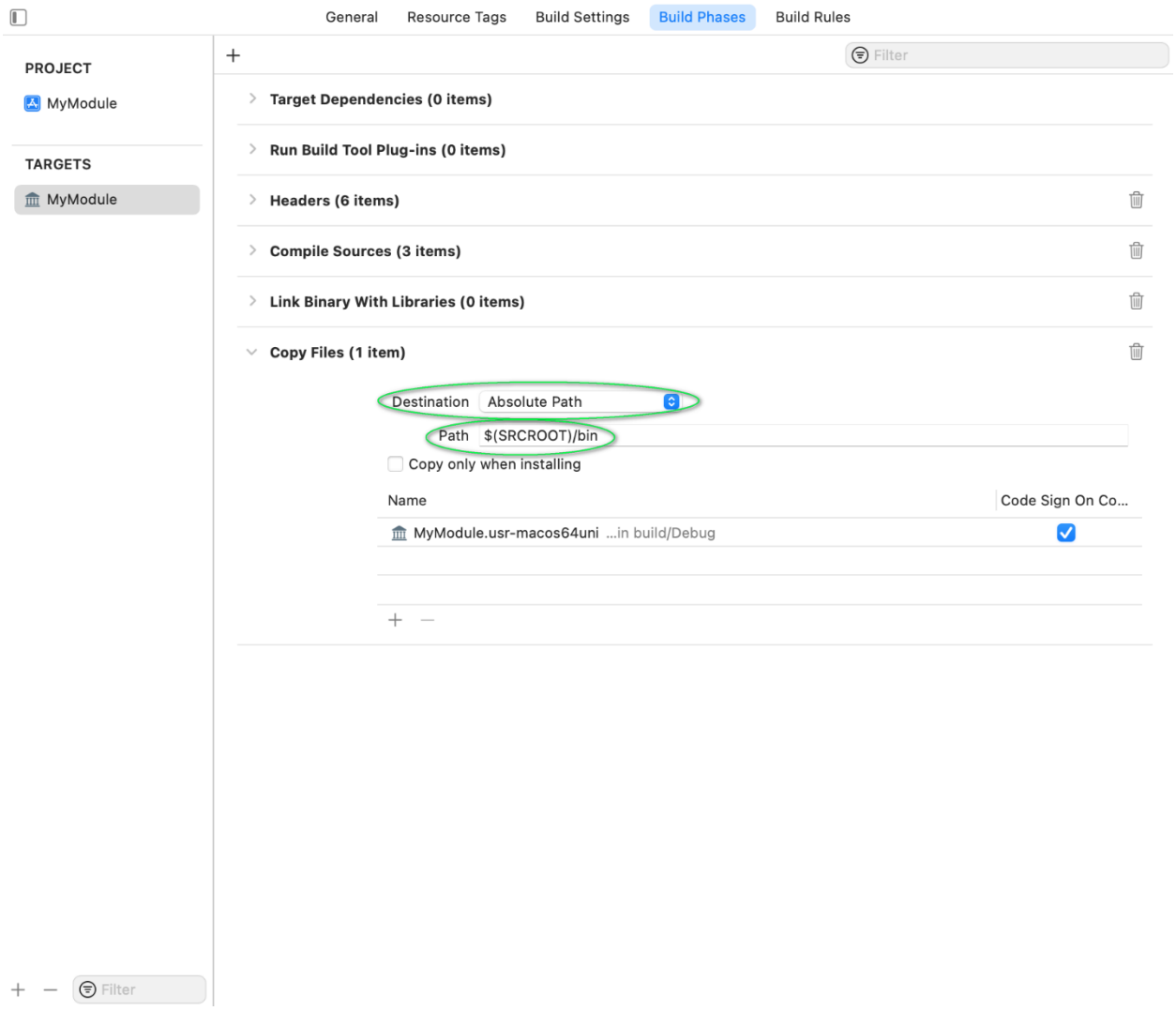
Finally, to make the built binary easily accessible you should tell Xcode to move it at the end of the build. To do so, navigate to the "Build Phases" tab and click the "+" at the top of the page and "New Copy Files Phase":




Hit the "+" button at the bottom of the new "Copy Files" section and select your product:



Set the destination to “Absolute Path” and enter the wanted path below, to make the path relative to the project’s root you can use $(SRCROOT), for example “$(SRCROOT)/bin” will output the binary in a “./bin” subdirectory. Your copy file phase should now look something like this:



## Other targets

Only a few settings must be changed to build for Intel or Silicon specifically instead of universal, as such we will only go over the needed settings here. The settings omitted here can be inferred from the above example. Also note that to add another target to a project on which you've already set up a target you can right click a target to duplicate it.

Beware that Xcode doesn't allow same name targets and that the binary will inherit the name of the target that built it. If you don't want this (which is probably the case), navigate to the build settings of your target and change the "Product Name" to reflect your wanted name.

## Building an Intel only binary

First, set the extension to "usr-osx64":

| | |
|---|---|
| **Executable Extension** | **usr-osx64** |

Then, go to the "VALID_ARCH" user-defined setting we created in the universal build and set its value to "x86_64":

| | |
|---|---|
| **VALID_ARCHS** | **x86_64** |

## Building an arm64 binary

First, set the extension to "usr-osxarm64":

| | |
|---|---|
| **Executable Extension** | **usr-osxarm64** |

Then, go to the "VALID_ARCH" user-defined setting we created in the universal build and set its value to "arm64":

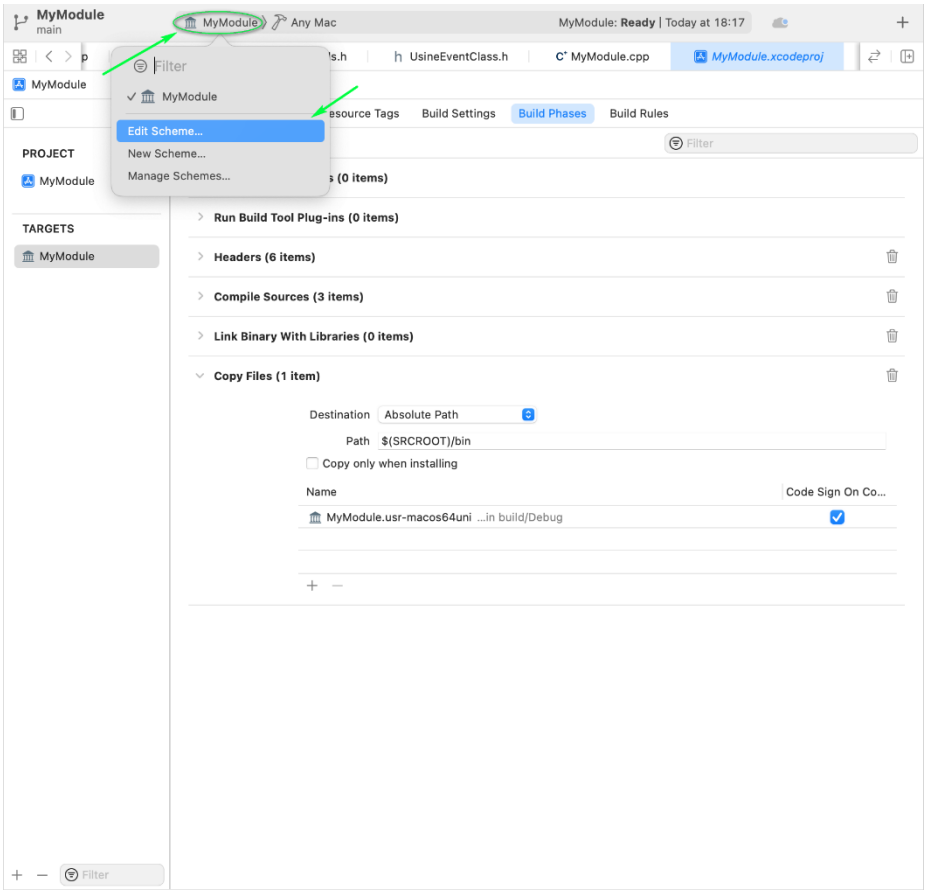
| | |
|---|---|
| **VALID_ARCHS** | **arm64** |

## How to test and debug your module

You can compile your module and debug it directly in Xcode while Usine is running. You can then put breakpoints, go step-by-step, etc... in your code and debug it efficiently.

To use Xcode debugger, you need to declare the application used to start your module (don't forget that a user module in Usine is nothing than a dylib, it then can't run by itself)

Go to Product menu, then select Scheme, then Edit Scheme:




Click on Run tab on the left side. Click on Executable list, then select "Other". A file selection dialog will open. Locate the Usine application on your disk and click on "Choose" button when the application is selected.

Once your module is compiled in Xcode, click on the "Run" arrow on top of Xcode window. Usine will then start. You can use all Xcode's debugging features (breakpoints, step by step, etc...).