Table of Contents

JUCE Cookbook

WARNING: WORK IN PROGRESS. SOME SECTION MIGHT NOT HAVE ANY CONTENT OR ONLY A LIST WITH KEYWORDS FOR ME AS A TODO LIST.

A collection of examples & workflow tips related to the C++ library JUCE.

Read Online

Latest PDF (direct download)

GitHub Repository

Intro

Why did I write this

I started using JUCE because I wanted to make my own audio plug-ins and after doing a little bit research JUCE seemed to be the way to go.

Over the last couple of years, I have collected all of the resources related to JUCE that I could find. The list of bookmarked pages & code snippets have grown to quite a collection, so I decided to publish them as a cheat sheet for myself and everybody else who uses or wants to use JUCE.

About myself

I started with programming at the age of 10. C++ was my first and primary language for the first couple of years. I have since then used Python, JS & Golang. I currently studying computer science with a focus on embedded systems in Berlin.

Disclaimer

I'm definitely not an expert in JUCE or C++. I do have a job as a C++ developer, but unfortunately not using JUCE. So if you find any problems in my examples, feel free to open an issue.

Code license

JUCE is published under a dual license, it is free for open source & small projects. You can find their license for JUCE here.

All of my example code in this repository is published to the public domain under the Creative Commons CC0 1.0 license.

Contribute

If you find any bugs or design problems in my examples feel free to open up an issue on GitHub.

If you want to add examples or resources to this collection you can either push a pull request directly or open up an issue first, if you have any questions. Please make sure that your topic is not already covered by one of the official JUCE tutorials. If you want to add to one of the official tutorials make sure you include a link to that page, so we can keep duplicate code as minimal as possible. Maintaining the same tutorial twice doesn't make much sense.

Getting Started

How to run examples

Each chapter can be read individually, feel free to jump around and find sections that are of interest to you.

All of the example projects included in this repository are normal `Projucer` projects, so running them is the same as any other project created with the `Projucer`

- Download JUCE
- Build the Projucer

Windows & macOS

- Open example projects `.jucer` file & save to generate the build files
- Open the project in your IDE (Visual Studio / XCode)
- Build

Linux

Make sure you add `Projucer` to your `PATH`.

```
cd $PROJECT_ROOT
Projucer --resave $PROJECT_NAME.jucer
cd Builds/LinuxMakefile
make config=[Release, Debug] -j8
```

Why JUCE

Features

Long story short, JUCE is a library which solves many common problems a developer might face during the creation of any kind of application. It hides away a lot of complexity, which normally is not very fun to work with. This includes cross platform window creation, file I/O, networking and so on. JUCE also comes with a collection of user interface widgets, such as buttons, combo boxes, menus, tabbed views and many more. A complete list can be found in the JUCE documentation. `Components` as they are called in JUCE form the basis of all user interaction in a JUCE application. The appearance is easily customizable. If the ones provided by JUCE don't fit your needs you can also create your own, just create subclass from the `Compoent` you want to customize and of you go.

Audio

- Wraps all common plug-in types.
- Basic DSP & Analysis
- Vector instructions for x86 and ARM

Mobil

- Easily use C++ for both Android & iOS
- Use native features

Other Libraries

Of course JUCE is not the only library that helps you build desktop & mobile applications. There are a lot of open source frameworks & libraries that try to achieve similar goals.

The following list is definitely not complete, but should give you a good overview on what is available in the C++ ecosystem.

QT

If you have been programming with C++ for a while, you probably heard of Qt. It's been around since the mid 90's and is currently being developed by The Qt Company.

The goal of Qt is to do it all. It runs on almost every platform including BSDs and embedded platforms. Because of this the whole framework is huge. While JUCE is around a couple of hundred megabytes, Qt is more in the range of a couple gigabytes.

Qt is also not really designed to help you write audio applications. There is no easy way to wrap a Qt application in an AudioUnit or VST plug-in for example.

If your goal is to write a desktop application in the style of Gimp, FreeCAD or Blender and you need a lot of premade desktop widgets, Qt is probably the way to go.

Qt is free for open source projects, but a license for closed source application can become pretty expensive.

SFML

- very simple
- intended for games

IMGUI

- very simple
- intended for simple widgets
- works well with SFML
  - cpp_box

Online resources

Tutorials

- JUCE tutorials

Docs & Forum

The two most important resources when developing with JUCE are the official JUCE documentation & the JUCE forum.

The API documentation for JUCE is very good in my opinion. All of the classes & functions are clearly documented.

The same goes for the forum, every time I asked a question, it was answered on the same day, which compared to other communities is simply awesome.

YouTube

If you like to learn using videos, you should be able to go from beginner to advanced using the resources found mainly on YouTube.

ADC Talks

The JUCE team has their own YouTube channel, where yo can find all of the talks from the yearly `Audio Developer Confrence (ADC)`. The content goes from general audio development to JUCE basics to advanced, companies showing of their workflow using JUCE and much more. I will link to specific talks in later sections.

The Audio Programmer

`The Audio Programmer` is a YouTube channel which almost only does JUCE related development tutorials. He has videos for the basics on getting up & running with the library, introduction to `Components`, an introduction to audio plug-in development and interviews with people working in the audio developer industry. Each video is `~20-60min` long.

If you are new to C++ and audio this is a create place to start.

If you are already pretty good in the language and have a basic understanding of audio in the digital world these videos will probably be a little to slow for you.

Discord

`The Audio Programmer` also runs a Discord channel for everything related to audio development. I hang around sometimes as well.

C++

To use all of the latest JUCE features you should at least use C++14 as your standard. Any newer standard works as well.

Idioms

If the list of idioms & language features below seems familiar to you, you're at a great starting point. If not, I have provided some resources below.

- Composition
- Inheritance
  - `virtual` / `override` / `final`
  - CppCon 2017: Louis Dionne “Runtime Polymorphism: Back to the Basics”
- Lambdas
  - CppCon 2019: Arthur O'Dwyer “Back to Basics: Lambdas from Scratch”
- RAII
  - CppCon 2019: Arthur O'Dwyer “Back to Basics: RAII and the Rule of Zero”
- Constexpr
  - CppCon 2015: Scott Schurr “constexpr: Introduction”
  - CppCon 2015: Scott Schurr “constexpr: Applications"
- Atomic/Lock
  - CppCon 2019: Rainer Grimm “Atomics, Locks, and Tasks (part 1 of 2)”
  - CppCon 2019: Rainer Grimm “Atomics, Locks, and Tasks (part 2 of 2)”
- Smart Pointer (unique & shared)
  - CppCon 2019: Arthur O'Dwyer “Back to Basics: Smart Pointers”
- Exceptions:
  - CppCon 2019: Ben Saks “Back to Basics: Exception Handling and Exception Safety”
- Where possible: `noexcept` / `const`
  - CppCon 2019: Dan Saks “Back to Basics: Const as a Promise”
- Container
  - array
  - vector
  - map

IDE

Visual Studio

- Only available on Windows
- Native `CMake` / `clang-format` support
- `clang-tidy` via plug-in

XCode

- Only available on macOS
- No good `clang-format` or `clang-tidy` integration

CLion

- Available on Linux/macOS/Windows
- Native `CMake` / `clang-format` / `clang-tidy` support

Visual Studio Code

- Available on Linux/macOS/Windows
- `CMake` / `clang-format` / `clang-tidy` support via plug-ins
- Has plug-ins for literally anything
- Uses a lot of memory

Tools

compiler

- warnings are your friend
- clang from source
- cross platform different warnings
- old versions in distros

clang-format

Automatic formatting of source code. A must have in my option. It doesn't matter what configuration you pick, the goal is for your complete code base to look as similar as possible.

```
# Formats all files with endings .h .hpp .cpp inplace (overrides).
find . -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' | xargs clang-format -i
```

clang-tidy

Static analysis tool. Takes a lot of CPU to run, but finds a lot of valid issues.

compiler-explorer

https://godbolt.org

Online compiler. Great for testing small code snippets. Does not currently have `JUCE` installed unfortunately.

coverage

- gcc + gcov & lcov
- clang + llvm-profdata & llvm-cov

Makefile/Scripts

I usually wrap all the common commands in a Makefile. Just to save some typing.

```
CONFIG ?= Release
BUILD_DIR ?= build
GENERATOR ?= Ninja

.PHONY: config
config:
    cmake -S. -B$(BUILD_DIR) -G$(GENERATOR) -DCMAKE_BUILD_TYPE=$(CONFIG)

.PHONY: build
build:
    cmake --build $(BUILD_DIR) --config $(CONFIG)

.PHONY: test
test:
    cd $(BUILD_DIR) && ctest -C $(CONFIG)

.PHONY: clean
clean:
    rm -rf $(BUILD_DIR)

.PHONY: format
format:
    find . -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' | xargs clang-format -i
```

Resources

YouTube

Must watch

- ADC 2016: The Golden Rules of Audio Programming, Pete Goodliffe
- CppCon 2014: Mike Acton "Data-Oriented Design and C++"
- CppCon 2015: Timur Doumler “C++ in the Audio Industry”
- CppCon 2016: Timur Doumler “Want fast C++? Know your hardware!"
- CppCon 2017: Kate Gregory “10 Core Guidelines You Need to Start Using Now”
- CppCon 2018: Jason Turner “Applied Best Practices”

Conferences

- CppCon
- c++Now
- ACCU
- Meeting Cpp
- Pacific++

CppWeekly

Run by Jason Turner. Each video focuses on one language feature or oddity.

- Channel

TheCherno

Focuses on Game & Graphics programming. Good basic C++ concept series.

- Channel
- C++ Series

Books

- Scott Meyers: Effective C++

Setup

Project Types

- Official JUCE: Choosing the right Projucer template for your application

Project Management

If you have multiple projects on your machine that use JUCE, you get to a problem where you update to a new release and now you have to check all of your projects if they still compile & work. Using git submodules solves this problem. You essentially have a copy of JUCE for each of your projects and the exact version is saved in your git history. Know updating to new JUCE version can be done on a per project basis.

```
cd $PROJECT_ROOT
mkdir 3rd_party
git submodule add https://github.com/juce-framework/JUCE 3rd_party/JUCE

# optional
cd 3rd_party/JUCE
git checkout juce6
```

Projucer

- Update module paths to new location
- Make sure it's done for each exporter

CMake

- Update path in `add_subdirectory`

macOS

Dependencies

Program	Description	Source	Comment
XCode	Compiler & IDE	AppStore	
CMake	Build file generator	brew / download	Optional
Ninja	Build system similar to Makefiles, but faster.	brew / download	Optional
xcpretty	Pretty prints XCode command line output	brew / download	Optional

```
brew install cmake ninja-build
gem install xcpretty
```

Install

```
git clone https://github/juce-framework/JUCE.git
cd JUCE/extras/Projucer/Builds/MacOS
xcodebuild --project Projucer.xcodeproj --configuration Release | xcpretty
```

Tools

xcpretty

```
xcodebuild [flags] | xcpretty                     # Pretty print
xcodebuild [flags] | tee xcodebuild.log | xcpretty  # Pretty print, but save raw output to file.
```

Windows

Dependencies

Program	Description	Source	Comment
Visual Studio	Compiler & IDE	Download from Microsoft	
CMake	Build file generator	Scoop / Choco / Download	Optional

Install

```
git clone https://github/juce-framework/JUCE.git
```

- Open `JUCE/extras/Projucer/Builds/VisualStudio2019/Projucer.sln`
- Build in `Release` mode

Linux

Dependencies

Program	Description	Comment
x11	Unix windowing system.	
xinerama	Multi display extension to x11.	
LibXext	More extensions to x11.	
ALSA	ALSA sound library.	
freetype	Font library.	
glu1-mesa		
webkit2gtk	WebKit Browser Engine	Optional `JUCE_WEB_BROWSER=0`
curl4-openssl	CURL library	Optional `JUCE_USE_CURL=0`
CMake	Build file generator. Replaces Projucer	Optional
Ninja	Build system similar to Makefiles, but faster.	Optional
Clang	Compiler.	Optional

Ubuntu

```
# Required
sudo apt install libx11-dev libxinerama-dev libxext-dev libfreetype6-dev libasound2-dev libglu1-mesa-dev
# Optional
sudo apt install libwebkit2gtk-4.0-dev libcurl4-openssl-dev
```

ToDo

- Arch
- Fedora
- Raspberry PI

Install

```
git clone https://github/juce-framework/JUCE.git
cd JUCE/extras/Projucer/Builds/LinuxMakefile
make config=Release -j8
```

Projucer vs. CMake

Prior to JUCE version 6, the Projucer is the only official way of creating projects. CMake support is currently available on the `juce6` preview branch.

If you already know how CMake works, this will be your best option. The preview branch is very stable.

Projucer

- Quick Setup
- Perfect IDE integration
- Hard to link external code/libraries

The Projucer is an application that comes with the JUCE library. It handles the creation if IDE projects and Makefiles depending on the platform. All of your configuration is stored in a `.jucer` file, which internally is xml. Compiler flags, defines & includes can be set for each platform independently. Limitations come when you want to link against third party code. If you get to this point you should probably switch to CMake.

This is definitely the fasted way of creating & running a project.

CMake

CMake support is coming in JUCE version 6. This will let you simply write `add_subdirectory(path/to/JUCE)` in your CMake configuration.

A guide can be found in the JUCE repository: github.com/juce-framework/JUCE/tree/juce6/examples/CMake

Example plug-in projects can be found here:

- tobanteAudio/juce-6-demo
- tobanteAudio/modEQ

Create Project

Projucer

- Offical JUCE: Projucer Part 1: Getting started with the Projucer
- Offical JUCE: Projucer Part 2: Manage your Projucer projects

CMake

- github.com/juce-framework/JUCE/tree/juce6/examples/CMake

Debugging

- AudioPluginHost
    - IDE Integration

Documentation

Doxygen

- Example: tobanteAudio/modEQ

GitHub Pages

static files

readthedocs

gitbook

UI

- Basic Components
    - Button
    - Slider
    - Label
    - Combobox
- Projucer live build
    - Tricks
- Animations
- LookAndFeel
    - Colour IDs
    - Override functions
    - Look at JUCE implementations
- Custom components
    - Look&Feel methods

DSP

- DSP module
    - Gain example
    - Compressor example
- AudioProcessor
- Plug-ins
    - VST/AU
- ValueTree
    - Parameters
    - Undo
    - Generic Editor
    - Attachments

Misc

Model View Controller

- Why
- Example
- Talk

File IO

- Record/Playback
- Sampler

Network IO

- Open Sound Control
  - Arduino

OpenGL

- Text for 3 examples
- How to use GLEW
  - If not, how to get function pointers using `openglcontext.extensions` .

JUCE Modules

Third Party Modules

Small and definitely not complete list of Third party JUCE modules.

Name	License	Description	Comment
ATK	BSD 3	An audio digital processing toolbox	Haven't used yet.
FF Meters	BSD 3	Plug and play component to display LED meters for JUCE audio buffers	Using it in modEQ. Love it.
Tracktion Engine	GPL/Commercial	A DAW framework	Haven't used yet.
Foleys Video Engine	GPL/Commercial	A video engine to load, play, assemble and write video	Haven't used yet.

If you know of any other JUCE modules, fell free to open an issue so we cam add it to the list.

Example Projects

OpenGL

- OpenGL Basic
- OpenGL Model
- OpenGL Shader

Code snippets

Plug-in recall

Variable `parameters_` is of type `juce::AudioProcessorValueTreeState`:

```
void YourPluginProcessor::getStateInformation(juce::MemoryBlock& destData)
{
    juce::MemoryOutputStream stream(destData, false);
    parameters_.state.writeToStream(stream);
}

void YourPluginProcessor::setStateInformation(const void* data, int sizeInBytes)
{
    juce::ValueTree tree = juce::ValueTree::readFromData(data, static_cast<size_t>(sizeInBytes));
    jassert(tree.isValid());
    if (tree.isValid())
    {
        parameters_.state = tree;
    }
}
```

Testing

Unit Tests

Catch2

github.com/catchorg/Catch2

- Link against CMake shared code target
- Register tests
- CTest

JUCE Unit Tests

- API Documentation
- Example
- How to run them in a plug-in build

pluginval

github.com/Tracktion/pluginval

The `pluginval` team describes there software as:

```
pluginval is a cross-platform plugin validator and tester application.
It is designed to be used by both plugin and host developers
to ensure stability and compatibility between plugins and hosts.
```

You can run `pluginval` both in command-line or GUI mode. So it's very easy to integrate into your `CI` pipeline.

Checks

- Randomly automate UI/parameters
- Recall parameters
- Call audio callback with different sample rates & buffer sizes
- Checking for memory allocations on audio thread (macOS AU only)
- And more...

Install

The simplest way to get pluginval is to download it directly from GitHub. See the releases page.

macOS

```
curl -L "https://github.com/Tracktion/pluginval/releases/download/latest_release/pluginval_macOS.zip" -o pluginval.zi
p
unzip pluginval
cp -r pluginval.app ~/Applications
```

Windows

TODO

Linux

```
curl -L "https://github.com/Tracktion/pluginval/releases/download/latest_release/pluginval_linux.zip" -o pluginval.zi
p
unzip pluginval
cp pluginval /usr/local/bin
```

GUI

Should be the same an all platforms:

- Launch `pluginval`
- Scan for plug-ins
- Set strictness
- Select plug-in to test
- Test

Command Line

macOS

```
~/Applications/pluginval.app/Contents/MacOS/pluginval --validate-in-process --strictness-level 10 --validate "path/to
/your.vst3" || exit 1
```

Windows

Somehow `pluginval` doesn't like `powershell`, so `cmd` should be used or run it in GUI mode.

Linux

```
pluginval --validate-in-process --strictness-level 10 --validate "path/to/your.vst3" || exit 1
```

Sanitizers

Address

Undefined Behaviour

Memory

Thread

Profile

Linux

perf

Set `-fno-omit-frame-pointer` for best results.

```
perf record -g path/to/exe
perf report -g 'graph,0.5,caller'
```

Benchmark

- `Google Benchmark`
- Generate test data
  - Noise
  - Sample
- Simple example

Continuous Integration

Explaining the concepts of `continuous integration` and `continuous deployment` is far out of scope for this document, but TLDR: CI helps you find problems in your code faster. Every time you push to your version control server of choice (e.g. GitHub or GitLab) builds on various platforms start up. For example in my plug-in project modEQ, I have builds for Windows, macOS & Linux. Since all of the platforms are using different compilers I get different warnings on each. Fixing all those little things as you go will save you a lot of time compared to developing exclusively on one platform and then trying to release it for a additional platform at the end.

There are many CI services online for free if your project is publicly available on GitHub or other platforms. I will focus on `travis-ci` and `appveyor` because those to combined will cover all desktop platforms JUCE supports (Windows, macOS & Linux).

Travis CI

- basics
- install dependencies
    - Linux fake Xorg
- platform matrix
- push to gh-pages
- run pluginval
- full example

Appveyor

Example Configuration

JUCE + CMake:

```yaml
version: 0.1.0.{build}
clone_folder: C:/projects/project
branches:
  only:
    - master

image: Visual Studio 2019
platform: x64
configuration:
  - Release

install:
  - cd %APPVEYOR_BUILD_FOLDER%
  - git submodule update --init --recursive

build:
  parallel: true

build_script:
  - cd C:/projects/project
  - mkdir build
  - cd build
  - cmake -G "Visual Studio 16 2019" ..
  - cmake --build . --config Release
```

Publish

- GitHub releases
- zip
- installer/package

Deploy

- docs
- app/plug-in

My wish list for JUCE

- Build system (solved in JUCE version 6)
  - CMake
  - Multiple targets in one project
- Better Graphics API integration
  - Metal (coming in JUCE version 6)
  - Vulkan
- Available in compiler-explorer
- FreeBSD support (Working on it, see tobanteAudio/juce-freebsd-example)

What next

- Read the docs
- Read the source
- Read other app written in JUCE
  - modEQ
  - helm
  - temper

Related resources

- Faust
  - YouTube
- `std::audio`

License